Hocheffiziente und skalierbare Software für die Simulation turbulenter Strömungen in komplexen Geometrien

Förderkennzeichen 01IH08010
Laufzeit des Projekts: 1.1.2009 – 30.6.2012

Gesamtschlussbericht

Koordinator:
Prof. Dr. Sabine Roller
Angewandtes Supercomputing im Maschinenbau
German Research School for Simulation Sciences GmbH
Schinkelstr. 2a
52062 Aachen

Tel.: 0241 / 80-99741
Fax.: 0241 / 80-6 99741
Email: s.roller@grs-sim.de

Projektpartner:

German Research School for Simulation Sciences

AIA RWTH Aachen

BOSCH Technik fürs Leben

HLRS

IAG

TRUMPF

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01 IH 08010 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

Liste der Zuwendungsempfänger:

- Universität Stuttgart
  Förderkennzeichen: 01IH08010 A
  Anschrift: Universitätsbereich Vaihingen, 70550 Stuttgart

  - Höchstleistungsrechenzentrum Stuttgart
    Anschrift: Nobelstr. 19, 70569 Stuttgart

  - Institut für Aerodynamik und Gasdynamik
    Anschrift:Pfaffenwaldring 21, 70569 Stuttgart

- RWTH Aachen
  Förderkennzeichen: 01IH08010 B
  Anschrift: Templergraben 55, 52056 Aachen

  - Lehrstuhl für Strömungslehre und Aerodynamisches Institut
    Anschrift: Wüllnerstr. 5a, 52062 Aachen

- Robert Bosch GmbH
  Förderkennzeichen: 01IH08010 C
  Anschrift: Robert-Bosch-Platz 1, 70839 Gerlingen-Schillerhöhe

- TRUMPF GmbH + Co. KG
  Förderkennzeichen: 01IH08010 D
  Anschrift: Johann-Maus-Straße 2, 71254 Ditzingen

- German Research School for Simulation Sciences GmbH
  Förderkennzeichen: 01IH08010 E
  Anschrift: 52425 Jülich

Abstract

Dieses Dokument ist der Schlussbericht (gem. NKBF 9 8) zum Projekt „STEDG — Hocheffiziente und skalierbare Software für die Simulation turbulenter Strömungen in komplexen Geometrien“, BMBF-Förderkennzeichen 01IH08010.

Das STEDG-Konsortium bestand aus den Partnern German Research School for Simulation Sciences GmbH (GRS), dem Höchstleistungsrechenzentrum Stuttgart (HLRS), dem Institut für Aerodynamik und Gasdynamik (IAG) an der Universität Stuttgart, dem Aerodynamischen Institut Aachen (AIA) an der RWTH Aachen, der Robert Bosch GmbH und der Trumpf Werkzeugmaschinen GmbH + Co. KG, sowie als assoziierter Partner EADS Deutschland GmbH.

Der Bericht stellt unter anderem die Ziele, den Ablauf, die Ergebnisse und die zukünftige Verwertung der Resultate des Projekts vor.

Inhaltsverzeichnis:

I. Kurze Darstellung

1. Aufgabenstellung

Die numerische Simulation von Strömungen ist eine unverzichtbare Methode für Forschung und Entwicklung in allen Ingenieurbereichen. Sie wurde zu einer Schlüsseltechnologie für die Verbesserung der Wirtschaftlichkeit, Umweltverträglichkeit und Sicherheit und trägt entscheidend zur Wettbewerbsfähigkeit der deutschen Industrie bei. Allerdings gründet sie sich häufig noch auf vereinfachte Methoden, um unter den vorgegebenen zulässigen Rechenzeiten zu Ergebnissen zu kommen. Die Berechnung turbulenter Strömungen um Realkonfigurationen beispielsweise basiert im Allgemeinen immer noch auf den Reynolds-gemittelten Navier-Stokes Gleichungen (Reynolds-averaged Navier-Stokes (RANS)), wobei die turbulenten Vorgänge durch algebraische oder Ein- bzw. Zwei-Gleichungs-Modelle modelliert werden. Es werden somit in der Regel Verfahren angewandt, die in der Zeit gemittelte stationäre Lösungen liefern. Für viele reale Industrieanwendungen spielen jedoch die zeitabhängigen Phänomene eine wichtige Rolle und die RANS Lösungen sind unzureichend. Zwar werden in den Forschungs- und Entwicklungsabteilungen erste Berechnungen, zum Beispiel basierend auf Large-Eddy-Simulationen (LES), ausgeführt. Es werden dazu aber kommerzielle Programmpakete eingesetzt - mit numerischen Methoden, die für die herkömmlichen stationären Simulationen mit Turbulenzmodellen geeignet sind und ein breites Anwendungsspektrum haben, für instationäre Vorgänge aber viel zu rechenzeitintensiv sind. Eine effiziente Simulation instationärer Vorgänge mit höherwertiger Turbulenzmodellierung bedingt aber eine umfassende Erneuerung aller Komponenten: Numerische Methoden und Algorithmen und deren effiziente Implementierung auf modernen Hardware-Architekturen.

Dieses Projekt adressierte die Änderungen, welche die Softwareentwicklung aufgreifen muss, um mit den Entwicklungen der Hardware Schritt zu halten. Es wurden hochaktuelle numerische Methoden, die von ihrer Struktur her genau auf diese Rechner-Architektur passen, zu effizienten Werkzeugen in der Strömungssimulation weiterentwickelt. Dabei ging es einerseits um die numerischen Methoden und Algorithmen der Zukunft als auch deren Mapping auf die Hardware-Architekturen von heute (multi-core) und morgen (many-core Architekturen). Es wurden reale Anwendungen aus der industriellen Forschung und Entwicklung (F+E) betrachtet und deren heutige Umsetzungsmöglichkeiten mit den aktuellen und zukünftigen Entwicklungen korreliert.

Hierzu wurden im Einzelnen folgende Ergebnisse angestrebt:

(1) Numerische Modellierung turbulenter Strömungen mit zeitgenauen numerischen Verfahren.

(2) Methoden zur Analyse und Optimierung der Rechen-Codes mit Anpassung an Hierarchien in der Hardware-Architektur in Bezug auf Netzwerk und/oder Memory.

(3) Verbesserung der Skalierbarkeit der Algorithmen innerhalb wie zwischen den Knoten.

(4) Evaluierung und Bestimmung von Cost-Faktoren für ein optimiertes Load-Balancing der in Raum, Zeit und Genauigkeitsordnung hoch-adaptiven Verfahren.

(5) Auswertung und Validierung anhand industrie-relevanter großer Testfälle, die bislang nicht oder nicht in akzeptablen Turn-Around-Zeiten berechenbar waren.

2. Voraussetzungen, unter denen das Vorhaben durchgeführt wurde

Projektpartner und Vorarbeiten

Arbeitsgruppe an der German Research School for Simulation Sciences GmbH
Der Lehrstuhl Angewandtes Supercomputing im Maschinenbau (ASE) beschäftigt sich mit der Simulation großer technischer Systeme. Dabei liegt der Forschungsschwerpunkt auf der Entwicklung moderner Simulationsmethoden für Mehr-Skalen- und Multi-Physik-Anwendungen und deren effizienter Implementierung auf Supercomputern. Dies erfordert die gesamte Prozesskette von der numerischen Entwicklung (Mathematik), der effizienten Implementierung (Informatik) bis hin zur Produktion (Maschinenbau). Die Anwendungen entstammen dabei überwiegend Fragestellungen der Projektpartner aus Industrie oder Medizin aus realen industriellen Aufgabenstellungen. Um diese realen Situationen simulieren zu können, ist der Einsatz von Höchstleistungsrechnern erforderlich. Aus diesem Grund liegt ein Fokus der Arbeitsgruppe, neben der Modellierung und Analyse von physikalischen Phänomenen, auch auf der effizienten Umsetzung von Algorithmen auf Supercomputern. Der interdisziplinäre Forschungsansatz der Gruppe zeigt sich in der Zusammenarbeit mit Partnern aus den Bereichen Mathematik, Maschinenbau und Informatik sowie mehreren europäischen Supercomputing-Zentren.

Arbeitsgruppe am Höchstleistungsrechenzentrum Stuttgart
Am Höchstleistungsrechenzentrum Stuttgart (HLRS) sind durch die Vielzahl unterschiedlicher Architekturen breite Erfahrungen in der Optimierung von Simulationscodes aus vielen Bereichen vorhanden. Entwicklungen wie die im Projekt zu verwendende Bibliothek „Abstract Data and Communication Library“ ADCL, ebenso wie eigene Werkzeuge, beispielsweise PACX-MPI für hybride Parallelisierung zwischen zwei oder mehr Maschinen oder Marmot, einem MPI Analyse und Checking Tool, unterstützen die Anwender. In vielen Kooperationen und Aktivitäten werden den Nutzern Konzepte und Methoden zur effizienten Nutzung moderner Architekturen im Hoch- und Höchstleistungsbereich vermittelt und gemeinsam umgesetzt.

Arbeitsgruppe am Institut für Aerodynamik und Gasdynamik der Universität Stuttgart
Die Arbeitsgruppe von Prof. Munz am Institut für Aerodynamik und Gasdynamik befasst sich seit vielen Jahren mit der Konstruktion numerischer Verfahren in der Strömungsmechanik, mit Implementierungsfragen und Anwendungen. Die Entwicklungen der letzten Jahre waren Finite-Volumen- und Discontinuous-Galerkin-Verfahren hoher Ordnung auf unstrukturierten Gittern, die zunächst zur Simulation der Schallausbreitung in komplexen Geometrien eingesetzt wurden. In den letzten fünf Jahren wurden beide Klassen von Verfahren auch für kompressiblen Strömungen weiter entwickelt und insbesondere die Kopplung mit der Lärmberechnung untersucht.

Arbeitsgruppe am Aerodynamischen Institut der RWTH Aachen
Die Arbeitsgruppe am AIA in Aachen ist in Deutschland die Arbeitsgruppe, die im Rahmen von DES, LES und Grobstruktursimulation für kompressible Strömungen mit die größte Expertise hat. Aufgrund der jahrelangen Entwicklung steht ein Lösungsalgorithmus für Large-Eddy Simulationen zur Verfügung, der auf blockstrukturierten Gittern basiert, Diskretisierungen bis zur 6. Ordnung enthält und verschiedene Feinstrukturmodelle besitzt.

Arbeitsgruppe der Robert-Bosch GmbH
Die Arbeitsgruppe der Robert-Bosch GmbH hat ihren Tätigkeitsschwerpunkt im Bereich von hydrodynamischen und thermodynamischen Problemstellungen. Behandelt werden sowohl Grundlagen- als auch Anwendungsthemen. Ein sehr wichtiges Gebiet in den kommenden Jahren sind gasführende Dosier- und Injektionssysteme, die z.B. in Methangas-betriebenen Kfz-Motoren Anwendung finden. Entsprechende Untersuchungen erfolgen dabei sowohl theoretisch als auch experimentell. Im theoretisch-numerischen Bereich werden aufbauend auf kommerziellen Tools wie Ansys CFX, Fluent etc. augenblicklich schon numerische Simulationen von Teilbereichen der Strömung durchgeführt. Neben normalen Workstations steht für Berechnungen auch ein Parallel-Cluster aus ca. 250 Opteron-Prozessoren zur Verfügung. Während die verwendeten Methoden prinzipiell in der Lage sind, Aussagen über stationäre und instationäre Phänomene zu liefern, bedeuten diese einen hohen Aufwand an Vorbereitung und Rechenzeit. Aufgrund dieser Schwierigkeiten kann z.B. keine systematische Auslegung von Komponenten basierend auf numerischen Simulationen erfolgen.

Arbeitsgruppe der TRUMPF Werkzeugmaschinen
TRUMPF beschäftigt sich seit über 20 Jahren mit der Entwicklung von Laserschneidköpfen für die Blechbearbeitung. Der Trend geht beim Laserstrahlschneiden von Blechen zu höheren Schneidgeschwindigkeiten und zu stärkeren Blechstärken. Um diesen Anforderungen nachzukommen, werden immer leistungsstärkere CO_2 Schneidlaser entwickelt. Diese Schneidlaser benötigen ein neues Schneidkopfkonzept, bei dem die Fokussierlinse durch einen Spiegel ersetzt wird. Der Wegfall des transmissiven Elementes in einem Spiegelschneidkopf sorgt jedoch dafür, dass die heute eingesetzten Lochdüsen nicht verwendet werden können. Dies macht die Entwicklung sogenannter Ringspaltdüsen notwendig. Zum Einstellen einer Schneidgasströmung, die obige Punkte erfüllt, gibt es eine Vielzahl von Stellschrauben, angefangen über Winkel- und Durchmesservariationen bis zu unzähligen Längenverhältnissen. Dabei müssen die Stellschrauben nicht nur für das Brennschneiden und das Schmelzschneiden, sondern auch für die jeweiligen Blechdicken angepasst werden. Für eine experimentelle Düsenentwicklung müssten nicht nur eine Vielzahl von Düsen gefertigt, sondern auch gasdynamisch und schneidtechnisch untersucht werden. Allein die schneidtechnischen Untersuchungen sind sehr zeitintensiv und mit erheblichen Kosten verbunden. Daher sind neben experimentellen Düsenuntersuchungen auch gasdynamische Simulationen zwingend notwendig.

Insgesamt verfügen die Antragsteller in den Bereichen Grobstruktursimulation mit und ohne Mischungsvorgänge sowie der LES auf zeitlich unabhängigen und zeitlich abhängigen Netzen als auch in dem Forschungsfeld Computational Aeroacoustics durch die Entwicklung und Anwendung der akustischen Störungsgleichungen auf Strömungs- und Verbrennungslärm über in vielen Jahren aufgebaute Expertise.

3. Planung und Ablauf des Vorhabens

Arbeitspakete

AP 0: Projektmanagement
- AP 0.1: Administrative Koordination

AP 1: Code-Optimierung für Multi- und Many-Core Architekturen
- AP 1.1. Analyse des vorhanden Strömungslösers
- AP 1.2. Single CPU und Intra-Node Optimierung, SMP-Parallelisierung
- AP 1.3. Inter-Node Kommunikation
- AP 1.4. Abstract Data and Communication Library (ADCL)

AP 2: Entwicklung der Numerischen Modelle
- AP 2.1: Weiterentwicklung der numerischen Verfahren unter Berücksichtigung der Hardware
- AP 2.2: Vergleich der numerischen Codes auf unterschiedlichen ArchitekturenAP
- AP 2.3: Infrastruktur für die Rechenprogramme

AP 3: Load Balancing und Cost Faktoren
- AP 3.1: Instrumentierung der Codes
- AP 3.2: Adaptive Numerik, Analyse und Anpassung der Parallelisierung zur Laufzeit
- AP 3.3.: Adaption an das Hardwaresetup bzgl. Memory Hierarchien, Architekturen, Spezialeinheiten

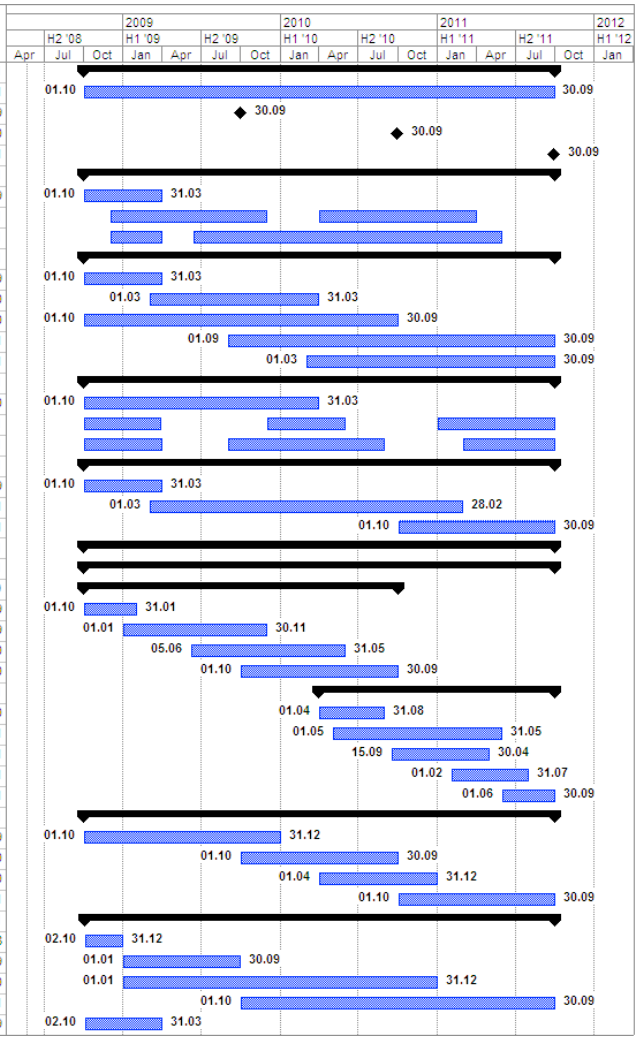
AP 4: Industrielle Anwendungen
- AP 4.1: Gaseinspritzung für benzin- und gasgetriebene Motoren
- AP 4.2: Dreidimensionale Hochauftriebskonfigurationen bei Flugzeugen
- AP 4.3: Optimierung des Gasstrahls beim Laser-Schneiden
- AP 4.4: Tools für vereinfachte Ausführung von Rechenjobs in externen Umgebungen.

Laufzeit

Das Projekt war auf eine Laufzeit von 36 Monaten angelegt. Aufgrund der Kurzarbeit bei den Industriepartnern, des Wechsels von Prof. Roller an die RWTH Aachen, den schwangerschafts-bedingten zeitlichen Ausfall und die Einarbeitung neuer Mitarbeiter war eine Verlängerung um 6 Monate erforderlich.

Gantt-Chart

ID	Task Name	Start	Finish
1	**0 Projektmanagement**	**Wed 01.10.08**	**Fri 30.09.11**
2	Projektmanagement	Wed 01.10.08	Fri 30.09.11
3	Jahresbericht Jahr 1	Wed 30.09.09	Wed 30.09.09
4	Jahresbericht Jahr 2	Thu 30.09.10	Thu 30.09.10
5	Abschlussbericht Jahr 3	Fri 30.09.11	Fri 30.09.11
6	**1 Code-Optimierung für Many-Core Architekturen**	**Wed 01.10.08**	**Fri 30.09.11**
7	1.1 Analyse der Codes	Wed 01.10.08	Tue 31.03.09
8	**1.2 Single CPU und Intra-Node Optimierung**	**Mon 01.12.08**	**Thu 31.03.11**
11	**1.3 Optimierung der Inter-Node Kommunikation**	**Mon 01.12.08**	**Tue 31.05.11**
14	**1.4 Weiterentwicklung der ADCL**	**Wed 01.10.08**	**Fri 30.09.11**
15	Erweiterung der Datenstrukturen	Wed 01.10.08	Tue 31.03.09
16	Einbau in den Code	Sun 01.03.09	Wed 31.03.10
17	Erweiterung der Kommunikationsmuster	Wed 01.10.08	Thu 30.09.10
18	Überlappende Berechnung / Kommunikation	Tue 01.09.09	Fri 30.09.11
19	Verhalten bei Adaptivität	Mon 01.03.10	Fri 30.09.11
20	**2 Entwicklung der numerischen Modelle**	**Wed 01.10.08**	**Fri 30.09.11**
21	2.1 Weiterentwicklung der Verfahren 1	Wed 01.10.08	Wed 31.03.10
22	**2.2 Vergleich auf verschiedenen Architekturen**	**Wed 01.10.08**	**Fri 30.09.11**
26	**2.3 Infrastrukture, Pre- und Postprocessing**	**Wed 01.10.08**	**Fri 30.09.11**
30	**3 Load Balancing und Cost Faktoren**	**Wed 01.10.08**	**Fri 30.09.11**
31	3.1 Instrumentierung der Codes	Wed 01.10.08	Tue 31.03.09
32	3.2 Adaptive Numerik	Sun 01.03.09	Mon 28.02.11
33	3.3 Adaption an Hardware-Setup	Fri 01.10.10	Fri 30.09.11
34	**4 Industrielle Anwendungen**	**Wed 01.10.08**	**Fri 30.09.11**
35	**4.1. Gaseinspritzung für Benzin- und Gas-Motoren**	**Wed 01.10.08**	**Fri 30.09.11**
36	**Arbeitsschritt 1**	**Wed 01.10.08**	**Thu 30.09.10**
37	Definition von Problemen/Anforderungen	Wed 01.10.08	Sat 31.01.09
38	Anpassung numerischer Codes	Thu 01.01.09	Mon 30.11.09
39	Validierung der DG-Codes	Fri 05.06.09	Mon 31.05.10
40	Benchmarking/Vergleich	Thu 01.10.09	Thu 30.09.10
41	**Arbeitsschritt 2**	**Thu 01.04.10**	**Fri 30.09.11**
42	Definition komplexer Beispielsysteme	Thu 01.04.10	Tue 31.08.10
43	Simulation / Ermittlung von Wirkzusammenhänge	Sat 01.05.10	Tue 31.05.11
44	Erweitertes Benchmarking	Wed 15.09.10	Sat 30.04.11
45	Optimierung für ausgewählte Beispiele	Tue 01.02.11	Sun 31.07.11
46	Abschließende Evaluation / Transfer	Wed 01.06.11	Fri 30.09.11
47	**4.2 3D Hochauftriebskonfiguration bei Flugzeugen**	**Wed 01.10.08**	**Fri 30.09.11**
48	LES der Strömung	Wed 01.10.08	Thu 31.12.09
49	CAA der Lärmerzeugung	Thu 01.10.09	Thu 30.09.10
50	Quell-Analyse	Thu 01.04.10	Fri 31.12.10
51	Design zur Lärmreduktion	Fri 01.10.10	Fri 30.09.11
52	**4.3 Optmierung des Gasstrahls beim Laser-Scheic**	**Thu 02.10.08**	**Fri 30.09.11**
53	Definition der Düsen-Geometrien	Thu 02.10.08	Wed 31.12.08
54	Experimentelle Messungen	Thu 01.01.09	Wed 30.09.09
55	Simulation der Gasströmung	Thu 01.01.09	Fri 31.12.10
56	Vergleich Fluent/STEDG	Thu 01.10.09	Fri 30.09.11
57	4.4. Tools für vereinfachte Nutzung (Unicore)	Thu 02.10.08	Tue 31.03.09



4. Wissenschaftlicher und technischer Stand, an den angeknüpft wurde

Ausgangssituation

Simulation von turbulenten Strömungen und Strömungslärm
Turbulente Strömungen für technische Anwendungen werden üblicherweise mit statistischen Turbulenzmodellen simuliert. Im Allgemeinen berücksichtigen diese Ansätze die Boussinesq-Hypothese und gehen im Wesentlichen von einem turbulenten Gleichgewicht aus. Je nach Strömung kann diese Modellierung große Fehler liefern wegen der Schwierigkeit der Modellierung anisotroper Strömungsstrukturen. Dagegen werden bei der Large-Eddy Simulation (LES) die größeren und energiereicheren Strukturen des Turbulenzspektrums auf einem adäquaten Rechengitter aufgelöst und nur die kleinen Turbulenzstrukturen werden modelliert. Damit werden die großen Strukturen ohne Modellierung erfasst, so dass die Voraussetzungen für das isotrope Turbulenzmodell besser erfüllt sind. Basierend auf Large-Eddy-Simulationen werden in den Forschungs- und Entwicklungs-abteilungen erste Berechnungen mit kommerziellen Programmpaketen ausgeführt, die zwar einen großen Anwendungsbereich haben, aber kaum auf die nötigen instationären Simulationen optimiert sind. Bei der aeroakustischen Lärmberechnung ist jedoch eine instationäre Strömungssimulation mit hoher Auflösung die Grundvoraussetzung, um sicher zu sein, dass die Schallquellen richtig erfasst werden. Kommt der Lärm aus dem turbulenten Strömungsfeld, müssen die relevanten turbulenten Strukturen der Strömung richtig und genau genug erfasst werden. Solche direkten Simulationen von Lärm durch ein turbulentes Strömungsfeld wurden ansatzweise schon durchgeführt – allerdings nur für einfache Geometrien und mit Rechenzeiten, die einen Einsatz im Design-Stadium nicht erlauben.

Numerische Methoden
Robuste numerische Simulationen von Strömungsproblemen in komplizierten Geometrien werden zurzeit vor allem mit Finite-Volumen (FV)-Verfahren 2. Ordnung ausgeführt. In kommerziellen Rechen-Codes werden *implizite* Druck-Korrektur-Verfahren benutzt, die sich dadurch auszeichnen, dass sie sowohl stationäre (RANS-) als auch instationäre Lösungen berechnen können und somit einen sehr breiten Anwendungsbereich haben. Die Effizienz der Berechnung von instationären Lösungen für turbulente Strömungen ist dabei nicht sehr hoch, da die Auflösung der Zeitentwicklung der physikalischen Vorgänge kleine Zeitschrittweiten erfordert, implizite Verfahren jedoch nur bei großen Zeitschrittweiten ihren Vorteil ausspielen können. Hier sind explizite Verfahren im Vorteil, da sie pro Zeitschritt deutlich weniger Rechenzeit benötigen. Um stabil zu sein, benötigen explizite Verfahren immer zwingend kleine Zeitschritte, insbesondere im inkompressiblen Strömungsregime, was ihre allgemeine Anwendbarkeit einschränkt, im konkreten Fall der instationären Turbulenz aber mit der physikalischen Zeitentwicklung übereinstimmt und daher keine Restriktion bedeutet. In kommerziellen Codes sind sie trotzdem kaum zu finden, da dort die Hauptzielsetzung ist, mit einem einzigen Code ein möglichst breites Spektrum an Anwendungen abzudecken. Dem Ziel der breiten Anwendbarkeit wird daher eine schlechte Performance für einzelne Anwendungsbereiche untergeordnet. Für den industrierelevanten Bereich der instationären turbulenten Strömungen ist dies aus den genannten Gründen leider der Fall. Im akademischen Umfeld werden daher andere Wege beschritten. Am AIA werden seit Jahren Forschungsarbeiten zur Simulation turbulenter Strömungen mit Grobstruktur-Modellen ausgeführt. Das Rechenprogramm ist dabei ein *explizites* Finite-Volumen (FV)-Verfahren. Neben den FV-Verfahren gibt es in der Numerik-Forschung neue Ansätze mit Verfahren höherer Ordnung, um die Effizienz zu erhöhen. So kombinieren Discontinuous-Galerkin (DG)-Verfahren die hohe Ordnung von Finite-Element (FE)-Verfahren mit der Möglichkeit der Auflösung starker Gradienten der FV-Verfahren.

Effizienz der Implementierungen

Im Rahmen von Vorarbeiten konnten Erkenntnisse in den typischen Implementierungsstatus von HPC-Anwendungen gewonnen werden. Dabei zeigte sich, dass insbesondere der für die zukünftigen Multi- und Many-Core Prozessoren wichtige Bereich der Shared Memory Parallelisierung bisher wenig Beachtung fand. Auch die Skalierung für den Bereich von Clustern mit großen Knotenzahlen ist i.d.R. nicht optimal. Je nach Anwendungen sind Speed-Up (dasselbe Problem in kürzerer Zeit) oder Scale-Up (größere Probleme in derselben Zeit) Betrachtungen wichtiger. Strategien, um Hierarchien in der Architektur, insbesondere im Netzwerk, oder Hierarchien im Code auszunutzen, finden sich selten. Dagegen sind Cache-Optimierung oder Vektorisierung häufig gut.

Limitierende Faktoren für die Simulationen sind häufig im Bereich der Pre- und Postprocessing-Tools zu finden, im Bereich kommerzieller Software auch beim Lizenzmanager. Gittergeneratoren sind häufig nicht in der Lage, Gitter mit sehr hohen Knotenzahlen zu erzeugen. Dies bedeutet für die Simulationscodes, dass die Gitter zu grob sind und im Initialisierungsschritt der Simulation aufbereitet und verfeinert werden müssen. Postprocessing-Tools, insbesondere Visualisierungstools lesen die zu visualisierenden Daten oft als ganzes ein und können daher sehr große Datenfiles nicht handlen. Lizenzmanager wie flexLM erlauben meist nur eine begrenzte Anzahl von Lizenzen gleichzeitig zu ziehen. Auf den derzeitigen Clustern sind die Prozessor-Zahlen noch nicht so hoch, dass diese Einschränkung spürbar wäre. Bei Clustern von mehr als 1000 Knoten und 8 Prozessoren pro Knoten wird dies aber spürbar. Diese Cluster können von kommerzieller Software dann nicht ausgenutzt werden, weil der Lizenzmanager nicht skaliert.

Wichtig für Nutzer sind üblicherweise nicht Flops-Zahlen oder CPU-Zeit, sondern Turn-Around Zeiten für den gesamten Prozess, von der Vorbereitungszeit inkl. Vernetzungsaufwand, Wartezeiten in der Queue, Ausführen der Simulation bis hin zu Datenübertragung und Visualisierung. Die Entwicklung der Codes findet dabei üblicherweise auf dem eigenen Desktop statt. Dort werden auch kleinere Testfälle gerechnet, um die aktuellen Implementierungen zeitnah validieren zu können. Systematisch getestet wird dann meist auf einem am Institutscluster . Dort werden mittlere Testfälle, auch parallel, simuliert. Die eigentlich interessierenden großen Anwendungen werden dann auf einer HPC-Maschine wie beispielsweise am HLRS durchgeführt. Optimierungen für alle Kombinationen von Maschine, Compiler, MPI-Library bedeuten Zeitaufwand und werden deshalb häufig nicht konsequent genug durchgeführt, insbesondere wenn die Anforderungen der Maschinen widersprüchlich sind.

Literatur

Resch, M., Roller, S., Lammers, P., S., Furui, T., Galle, M., Bez, W. (Eds.) High Performance Computing on Vector Systems 2007, Springer, 2007.

Harald Klimach, Sabine P. Roller, Jens Utzmann and Claus-Dieter Munz: Parallel Coupling of Heterogeneous Domains with KOP3D using PACX-MPI, Track No: 1126, Proceedings Parallel CFD 2007 May 21-24, Antalya Turkey

Sabine Roller: InGrid - Innovative Grid Developments for Engineering Applications, inside, Vol. 4 No. 1, Spring 2006

E. Gabriel, S. Feki, K. Benkert, M. Chaarawi. The Abstract Data and Communication Library, submitted for publication in Journal of Algorithms and Computational Technology, Special Issue on Computational Science for Medicine, Energy, and Environment Applications.

Bettina Krammer, Katrin Bidmon, Matthias S. Müller, Michael M. Resch. "MARMOT: An MPI Analysis and Checking Tool", *Parallel Computing 2003* , Published in PARALLEL COMPUTING: Software Technology, Algorithms, Architectures & Applications, Ed. Joubert, Nagel, Peters, Walter, pp. 493-500, Elsevier, 2004.

Edgar Gabriel and Shuo Huang, Runtime Optimization of Application Level Communication Patterns, 12th International Workshop on High-Level Parallel Programming Models and Supportive Environments, held in conjunction with IPDPS 2007, Long Beach, CA, March 26th 2007.

Katharina Benkert, Edgar Gabriel, and Michael M. Resch, 'Outlier Detection in Performance Data of Parallel Applications', in Proceedings of the 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, held in conjunction with the IPDPS 2008, Miami, Fl, USA, March 2008.

Edgar Gabriel, Saber Feki, Katharina Benkert, and Michael M. Resch, 'Towards Performance and Portability through Runtime Adaption for High Performance Computing Applications', accepted for publication at the International Supercomputing Conference, June 17-20, 2008, Dresden, Germany.

Rainer Keller, Markus Liebing, 'Using PACX-MPI in MetaComputing applications', 18th Symposium Simulationstechnique, Erlangen, Sept. 12.-15., 2005.

M. Dumbser and C.-D. Munz: ADER Discontinuous Galerkin schemes for aeroacoustics, CRAS Mécanique 333 (2005), 683-687

J. Utzmann, T. Schwartzkopff, M. Dumbser, C.-D. Munz: Heterogeneous domain decomposition for computational aeroacoustics, AIAA Journal 44 (2006), 2231-2250

M. Dumbser and C.-D. Munz: Building blocks for arbitrary high order discontinuous Galerkin schemes, Journal of Scientific Computing 27 (2006), 215-230

M. Dumbser and C.-D. Munz: On source terms and boundary conditions using arbitrary high order discontinuous Galerkin schemes, Int. J. Appl. Math. Comput. Sci 17 (2007), 101-114

D.S. Balsara, C. Altmann, C.-D. Munz, M. Dumbser: A sub-cell based indicator for troubled zones in RKDG schemes and a novel class of hybrid RKDG-HWENO schemes, J. Comput. Phys. 226 (2007), 586-620

C.-D. Munz, M. Dumbser, S. Roller: Linearized acoustic perturbation equations for low Mach number flow with variable density and temperature. J. Comput. Phys. 224 (2007), 352-364

F. Lörcher, G. Gassner, C.-D. Munz: A discontinuous Galerkin scheme based on a space-time expansion. I. Inviscid compressible flow in one space dimension, Journal of Scientific Computing 32 (2007), 175-199.

F. Lörcher, G. Gassner, C.-D. Munz: A discontinuous Galerkin scheme based on a space-time expansion. II. Viscous compressible flow in multi space dimensions, Journal of Scientific Computing 34 (2008), 260-286

G. Gassner, F. Lörcher, C.-D. Munz: A contribution to the construction of diffusion fluxes for finite volume and discontinuous Galerkin schemes. J. Comput. Phys. 224 (2007), 1049-1063

F. Lörcher, G. Gassner, C.-D. Munz: An explicit discontinuous Galerkin scheme with local time-stepping for general unsteady diffusion equations, J. Comput. Phys. 227 (2008), 5649-5670

Rütten, F., Schröder, W., and Meinke, M., LES of Low Frequency Oscillations of the Dean Vortices in Turbulent Pipe Bend Flows, Physics of Fluids 17, 2005.

Guo, X., Schröder, W., and Meinke, M., Large-eddy simulations of film cooling flows", Computers and Fluids 35, 587-606, 2006.

Renze, P., Schröder, W., and Meinke, M., Large-Eddy Simulation of Film Cooling Flows at Density Gradients, to appear in Int. J. Heat and Fluid Flow, 2008.

Ewert, R. and Schröder, W., Acoustic perturbation equations based on flow decomposition via source filtering, J Comput Phys.188, 365-398, 2003.
Bui, T. Ph., Schröder, W., and Meinke, M., Acoustic perturbation equations for reacting flows to compute combustion noise, Int. Journal of Aeroacoustics 6, 335-355, 2007.
Gröschel, E., Schröder, W., Renze, P., Meinke, M., and Comte, P., "Noise prediction for a turbulent jet using different hybrid methods, in press in *Computers and Fluids,* 2008.
Meinke, M., Schröder, W., Krause, E., Rister, Th.: A Comparison of Second- and Sixth-Order Methods for Large-Eddy Simulations, Computers & Fluids, Vol. 31, pp. 695-718, 2002.
Rütten, F., Schröder, W., Meinke, M.: LES of Low Frequency Oscillations of the Dean Vortices in Turbulent Pipe Bend Flows, Physics of Fluids, Vol. 17, Issue 2, 035107, 2005.
Schröder, W., Meinke, M., Schvorak, A.: Large-Eddy Simulations of Accelerated Pipe Flows, Computational Fluid Dynamics JOURNAL, 2001.
Alkishriwi, N., Meinke, M., Schröder, W.: A Large-Eddy Simulation Method for Low Mach Number Flows Using Preconditioning and Multigrid, Computers and Fluids, 35, 1126-1136, 2005.
Ewert, R., Schröder, W.: On the Simulation of Trailing Edge Noise with a Hybrid LES/APE Method, Journal of Sound and Vibration, Vol. 270, pp. 509-524, 2004.
El-Askary, W.A., Schröder, W., Meinke, M.: LES of Compressible Wall-Bounded Flows, AIAA Paper 2003-3554, 2003.
Ewert, R., Meinke, M., Schröder, W.: Aeroacoustic source terms for the linearized Euler-equations, in Proceedings 6th AIAA/CEAS Aeroacoustics Conference, 2000, AIAA Paper 2000-2046.
Ewert, R., Schröder, W., Meinke, M., El-Askary, M.A.: LES as a Basis to Determine Sound Emission, AIAA Paper 2002-0568, Jan., 2002.
Ewert, R., Zhang, Q., Schröder, W., Delfs, J.: Computation of Trailing Edge Noise of a 3D Lifting Airfoil in Turbulent Subsonic Flow, AIAA Paper 2003-3114, 2003.
Bui, T.Ph., Meinke, M., Schröder, W.: A hybrid method for combustion noise based on LES and APE, AIAA Paper 2005-3014, 2005.

5. Zusammenarbeit mit anderen Stellen

Im Projekt wurde intensiv mit dem Lehrstuhl Parallele Programmierung von Prof. Felix Wolf zusammengearbeitet und Ergebnisse der Projekte SILC und LMAC verwendet. Im Vordergrund stand hier die Benutzung von Tools wie Scalasca, Vampir und Tau, um Schwachstellen der Codes zu identifizieren. Diese Analyse bildet die Basis für Optimierung der Anwendungen.

Die Zusammenarbeit mit den Rechenzentren des Gauss-Centers for Supercomputing und der Gauss-Allianz war immer ein wichitger Aspekt im Projekt. Die stetige Fortentwicklung der Software-Umgebung auf den Maschinen, insbesondere bei Neuinstallationen, erfordert eine entsprechende kontinuierliche Anpassung der Simulationscodes. So mussten beispielsweise Compilervergleiche durchgeführt werden, da sich unterschiedliche Compiler auf die Rechengeschwindigkeit, aber auch auf das Zusammenspiel mit der jeweiligen MPI-Library ausgewirkt haben. Hier bestand regelmäßiger Kontakt sowohl mit den Rechenzentren, den Herstellern der Systeme, als auch mit den Herstellern von Analyse-Tools wie Debuggern und Performance-Analyse.

II. Eingehende Darstellung

1. Verwendung der Zuwendung und des erzielten Ergebnisses im Einzelnen

Die erzielten Ergebnisse sind fachlich detailliert im englischsprachigen Report beschrieben, der diesem Bericht als Anlage beigefügt ist. Deshalb erfolgt hier nur eine knappe Aufzählung mit Gegenüberstellung der angestrebten zu den erreichten Ergebnissen.

AP 0: Projektmanagement
Im Bereich des Projektmanagements wurden die regelmäßigen Projekttreffen koordiniert, die in der Regel 3x im Jahr stattfanden. Zum Projektende wurde ein Abschlussmeeting abgehalten, zu dem die nicht direkt am Projekt beteiligten Nachbarabteilungen der industriellen Partner eingeladen und die Ergebnisse mit ihnen diskutiert wurden. Die Beteiligung an der 1. HPC Statuskonferenz in Schwetzingen, der 2. HPC Statuskonferenz in Darmstadt und Beiträge für den Infobrief der Gauss-Allianz wurden hier koordiniert. Ein zentrales Repository für Code- und Datenaustausch sowie eine Projektwebseite wurden bereitgestellt und gepflegt.

AP 1: Code-Optimierung für Multi- und Many-Core Architekturen
Ziel dieses Arbeitspakets war die Analyse und Optimierung der Strömungslöser innerhalb eines Knotens sowie zwischen den Knoten. Für stark wechselnde Lasten sollte die Abstract Data and Communication Library (ADCL) zur Optimierung zur Laufzeit eingesetzt werden. Kapitel 4 des technischen Reports beschreibt die Ansätze und Ergebnisse der Optimierung im Detail. In Kürze zusammengefasst lässt sich sagen, dass die Performance pro Knoten um bis zu 57% reduziert werden konnte. Starke Verbesserungen konnten im Bereich der parallelen IO erzielt werden. Insbesondere das Einlesen der Gitter- und ggfs. Restart-Daten für einen Neustart war anfangs so aufwändig, dass es große Rechnungen u.U. sogar verhindern konnte. Dieser Flaschenhals konnte behoben werden, so dass es hier keine Einschränkungen mehr gibt. Ebenso konnten die Output-Routinen deutlich verbessert werden. Die Arbeiten an der ADCL wurden abgeschlossen und mündeten in der Dissertation von Frau Benkert. Im Rahmen des Projektes wurde sie allerdings nicht weiterverfolgt, da sich zeigte, dass die Charakteristika der Anwendungen keinen weiterer Vorteil durch Verwendung der ADCL bringen würden. Andererseits wurde sehr viel stärker auf den Memory-Bedarf der Anwendung fokussiert, der sich als starke Einschränkung herausstellte. Insbesondere in der Startup-Phase der Simulation, in der Gitter eingelesen und Nachbarschaftsbeziehungen aufgebaut werden, zeigten sich implizite Serialisierungen, die sich teils auf die Rechenzeit, aber vor allem auf den Arbeitsspeicherbedarf auswirkten. Zu Beginn des Projekts führte dies zu Programmabstürzen, im Laufe des Projektes wurde dies gelöst, u.a. durch Entwicklung einer vollständig parallelen Implementierung der Startup-Phase.

AP 2: Entwicklung der Numerischen Modelle
Ziel dieses Arbeitspaketes war die Weiterentwicklung der numerischen Verfahren so, dass sie die vorhandenen Hardware-Architekturen effizient nutzen konnten. Neben dem Vergleich der Verfahren auf unterschiedlichen Computesystemen, die an den GCS-Sites in Stuttgart und Jülich, beim Gauss-Allianz-Mitglied Rechenzentrum der RWTH Aachen sowie an den einzelnen beteiligten Instituten als auch bei der Robert Bosch GmbH verfügbar sind, wurden Erweiterungen und Anpassungen der Numerik in Hinblick auf die Hardware untersucht.
Zum einen standen Verfahren hoher Ordnung im Vordergrund. Es werden bei hoher Ordnung weniger Gitterzellen benötigt, die Gittererzeugung wird einfacher, aber sowohl der Rechen- als auch der Speicheraufwand pro Zelle steigen. Es zeigten sich hier sowohl das Potential als auch die Grenzen dieser Verfahren: Durch den erhöhten Aufwand je Zelle ergibt sich ein weiterer Freiheitsgrad, der für die Optimierung genutzt werden kann, außerdem werden mehr Rechenoperationen pro Dateneinheit durchgeführt (Byte per Flops Ratio), was zu einer verbesserten Single-CPU Performance und somit höherer Effizienz führt. Andererseits enthält das Rechengitter weniger Zellen, so dass die Gebietszerlegung und damit die starke Skalierung auf Rechensystemen eingeschränkt wird, wenn zu wenige Zellen je Knoten anfallen. Durch die Tendenz zu mehr Cores pro Rechenknoten und somit stärker anwachsender Intra-Node-Performance bei eher gleich bleibender oder zumindest langsamer

wachsender Inter-Node-Performance ist dies jedoch eher ein Vorteil, da sich der Bedarf der numerischen Verfahren in derselben Richtung entwickelt wie die Tendenz in der Hardware.
Zum anderen stand die Turbulenzmodellierung selbst im Fokus. Komplexe fluidmechanische Probleme erfordern in der Regel Large-Eddy-Simulation (LES), welche aber im Vergleich zu den in kommerziellen Lösern üblicherweise verwendeten Reynolds-Averaged Navier-Stokes (RANS) Verfahren sehr viel rechenintensiver sind. Durch die effiziente Implementierung des LES-Verfahrens konnten zum einen detaillierte Untersuchungen mit LES-Verfahren durchgeführt und so die notwendigen Informationen für die Evaluierung bestehender Probleme von RANS-Ansätzen zu erhalten. Zum anderen konnten auf Basis dieser Simulationen dann Kopplungsmechanismen für den zonalen RANS-LES Ansatz weiterentwickelt werden, bei dem der LES-Ansatz nur dort verwendet wird, wo es notwendig ist, während andere Bereiche des Simulationsgebiets mit RANS modelliert werden. Durch die lokale Anwendung einer LES konnte zum einem die Unsicherheit bezüglich der Qualität des Gesamtergebnisses auf ein Minimum reduziert und zum anderen der Rechenaufwand hinsichtlich Gesamtsimulationsdauer und Speicherbedarf optimiert werden. Gerade im Kontext der numerischen Optimierung ist eine Verkürzung der Rechenzeiten wesentlich für die Anwendbarkeit der Verfahren unter industriellen Bedingungen.

AP 3: Load Balancing und Cost Faktoren
In diesem Arbeitspaket wurde die Lastverteilung zur Laufzeit betrachtet. Starke Imbalancen, in denen viele Prozesse auf wenige warten müssen, reduzieren die Effizienz der Simulationen. Um dynamisch auf Last-Imbalancen reagieren zu können, müssen diese zunächst entdeckt und bewertet werden können. Im ersten Schritt ist dafür die Instrumentierung des Codes notwendig, um zur Laufzeit die Ausführungszeiten und ggfs. Speicheranforderungen jeder Zelle in Abhängigkeit von Zeitschritt, Element-Typ und Verfahrensordnung bestimmen zu können. Wurden diese Abhängigkeiten zu Beginn des Projektes im Wesentlichen heuristisch ermittelt, erfolgt dies nun durch Messungen zur Laufzeit. Diese Werte werden im nächsten Schritt verwendet, um eine Re-Partitionierung des Rechengebiets vorzunehmen. Hier wurde mit dem Sparta-Algorithmus ein effizientes Verfahren nicht nur zur Bestimmung der neuen Aufteilung, sondern auch zum Durchführen der Umverteilung, also zum Versenden von Gitterelementen und ihrer Daten an andere Prozesse im Rechengebiet implementiert. Besonderes Augenmerk musste auf den Algorithmus gelegt werden, in dem die Prozesse bestimmen, ob der Austausch abgeschlossen ist, oder noch weitere Nachrichten empfangen werden müssen. Hier wurden in Zusammenarbeit mit Prof. Felix Wolf die Möglichkeiten des neuen MPI-3 Standards untersucht. Hier werden nun sog. Non-Blocking Collectives verwendet. Dies hat nicht nur zu einer optimalen Routine im STEDG-Projekt geführt, sondern war gleichzeitig auch der für die Aufnahme in den neuen MPI-Standard notwendig Einsatz in der Praxis.

AP 4: Industrielle Anwendungen
Dieses Arbeitspaket betrachtete die Anwendung der Verfahren auf reale industrielle Testcases. Besonders im Vordergrund stand die Zeit vom Aufsetzen der Simulation bis zum Ergebnis. Besonders für den Einsatz in der sog. Design of Experiment Phase, also der Phase, in der Voruntersuchungen auf Basis von Simulationen darüber entscheiden, welche Prototypen gebaut und experimentell validiert werden sollen, verlangt die Durchführung von relativ vielen, detaillreichen Simulationen. Die wurde für drei unterschiedliche Anwendungen, a) Gaseinspritzung für benzin- und gasgetriebene Motoren, b) dreidimensionale Hochauftriebskonfigurationen bei Flugzeugen, c) Optimierung des Gasstrahls beim Laser-Schneiden durchgeführt. In allen Anwendungen konnten sowohl wissenschaftlich-technischen Erkenntnisse aus der Simulation gewonnen werden, eine ausführliche Beschreibung der Ergebnisse ist im (englischsprachigen) Projektbericht dargestellt. Auch die Handhabung der Simulationen, also der gesamte Workflow von Gittergenerierung, Preprozessing, Simulation und Postprocessing, verbessert werden. Insbesondere der Zugang für Industrieanwender auf die Höchstleistungsrechner am HLRS wurde ermöglicht, was weniger eine technische als eine Frage der Policies und Berechtigungen war. Hierfür wurde mit der entsprechenden IT-Abteilung zusammengearbeitet.

2. Wichtigste Positionen des zahlenmäßigen Nachweises

Summarische Darstellung der Mittel:	geplant	verwendet	
Personalbedarf:	189 PM	190,75 PM	
Finanzierungsbedarf des Projektes:	1.416.990 €	1.318.714€	= 93%
Förderbedarf:	1.040.248 €	987.729 €	= 95%
Industrieanteil am Gesamtfinanzierungsbedarf:	53%	50%	
Industrieanteil am Fördervolumen:	36%	34%	
Akademischer Anteil an Gesamtfinanzierung:	47%	50%	
Akademischer Anteil am Fördervolumen:	64%	66%	

Die Gesamtkosten wurden zu 75% durch Fördermittel, zu 25% durch Eigenmittel der Industriepartner geleistet. Der Eigenanteil der Industriepartner beläuft sich auf 330.986 €. Der Kostenrahmen wurde dem Antrag entsprechend eingehalten.
Der Zeitplan musste aufgrund der Kurzarbeit bei den Industriepartnern, des Wechsels von Prof. Roller an die RWTH Aachen, den schwangerschaftsbedingten zeitlichen Ausfall und die Einarbeitung neuer Mitarbeiter um 6 Monate verlängert werden.

3. Notwendigkeit und Angemessenheit der geleisteten Arbeit

Im Projekt wurde die detaillierte Simulation turbulenter Strömungen im Kontext realer industrieller Anwendungen und ihre Praktikabilität im Kontext realer industrieller Randbedingungen, insbesondere vorgegebener Turn-Around-Zeiten untersucht. Ausgangspunkt war die Diskrepanz zwischen modernen Methoden, wie sie in Universitäten und Forschungseinrichtungen im Rahmen akademischer Forschungscodes untersucht wurden, und den Lösungsverfahren, wie sie aufgrund von Rechenzeitbeschränkungen in kommerziellen Softwarepaketen in der Industrie im Einsatz sind. Akademische Verfahren bilden die physikalischen Phänomene richtig ab, sind aber zu langsam. Standardverfahren in kommerziellen Lösern sind dagegen schnell, erreichen dies aber durch Verwendung unzureichender Modellierungsansätze.

Im Rahmen des STEDG-Projektes konnte diese Diskrepanz überwunden werden: es konnte gezeigt werden, dass moderne, bei den akademischen Partnern entwickelte Modellierungs- und Lösungsverfahren so implementiert werden können, dass sie die Möglichkeiten aktueller und zukünftiger Supercomputer nutzen können, und dadurch auch in der Lage sind, die Vorgaben der industriellen Anwender, wieviele Simulationen in welcher Zeit durchzuführen sind, erfüllen können. Parallel dazu wurden auch Hürden beseitigt, die nicht in erster Linie technischer Natur sind, sondern auf Policies und Zugangsvoraussetzungen beruhen, und bisher das Nutzen der am HLRS installierten und für industrielle Anwender zugängliche Supercomputer behindert hatte. Das Projekt hat dadurch direkt zum Austausch zwischen Industrie und Forschung beigetragen und stützt damit sowohl den Forschungs- als auch den Industriestandort Deutschland.

Über diese direkten Projektziele und –ergebnisse hinaus trug das Projekt auch zur Community-Bildung und insbesondere zur Zusammenarbeit zwischen (akademischen) Nutzern und den Rechenzentren des Gauss Centers for Supercomputing (GCS) und der Gauss Allianz bei. Projektpartner portierten und analysierten die Software auf unterschiedlichen Systemen, verglichen so die Leistungsfähigkeit der verschiedenen Hardware-Architekturen, berichteten den Zentren und deren Herstellern über Probleme und Bugs in Systemsoftware wie Compilern, Bibliotheken wie MPI, Tools wie Vampir, Tau und Scalasca, und Konfigurationen wie Modulumgebungen, Pfade, aber auch Zusammenspiel beispielsweise von paralleler IO und parallelem Filesystem. Auch die Industriepartner können für ihre nächsten Computecluster-Beschaffungen auf diese Ergebnisse zurückgreifen.

Im Projekt wurden 5 Doktoranden, mehrere Masterstudierende und etliche wissenschaftliche Hilfskräfte ausgebildet. Ebenso flossen die Ergebnisse in die Vorlesungen und Labore der akademischen Partner ein. Das Projekt trug so auch zur Ausbildung des Nachwuchses im Bereich Simulation Sciences bei.

4. Voraussichtlicher Nutzens, insbesondere Verwertbarkeit des Ergebnisses im Sinne des fortgeschriebenen Verwertungsplans

Der Nutzen und die Verwertbarkeit der Ergebnisse ergeben sich auf unterschiedlichen Ebenen.

Auf akademischer Seite steht der wissenschaftliche Fortschritt im Vordergrund, der sich in über 30 Veröffentlichungen in wissenschaftlichen Journals, Konferenzen, Buchbeiträgen, Postern und Vorträgen darstellt. Daneben sind 5 Dissertationen und eine Masterarbeit entstanden. Die Ergebnisse fließen in weitere Projekte sowie die zukünftige Softwareentwicklung ein. Auch in die Lehre in Form mehrerer Vorlesungen für den Studiengang Simulation Sciences sowie Trainingsveranstaltungen wie den seit Jahren in Kooperation von GRS, HLRS und IAG durchgeführten CFD-Kurs flossen die Erfahrungen ein.

Auf Seiten der Industriepartner stehen zwei Aspekte im Vordergrund. Einerseits natürlich die direkte Optimierung von Produkten, die vor allem auf dem Verständnis der internen Vorgänge beruht, welche Simulationen mit feiner Auflösung und hohem Detailgrad erfordern, was mit kommerziellen Lösern auf Standard-Rechensystemen nicht erreichbar ist. In diesem Sinne fliessen die Ergebnisse der Simulationen selbst direkt in die Entwicklung ein. Zum anderen besteht der Gewinn des Projektes auch im Wissenstransfer, sowohl was die Hintergründe der Simulationsverfahren angeht, aber noch viel stärker, was die Nutzung und Nutzbarkeit von Supercomputern im industriellen Umfeld angeht. Das Klären von Zugangsvoraussetzungen (Firewalls, Windows2Linux, Lizenzproblematik) ermöglicht die Verwendung von Supercomputern außerhalb des Unternehmens.

Über diesen spezifischen Nutzen und Verwertungen für die direkt beteiligten Partner war die Zusammenarbeit mit den Rechenzentren ein wichtiger Aspekt im Projekt. Die stetige Fortentwicklung der Software-Umgebung auf den Maschinen, insbesondere bei Neuinstallationen, erfordert eine entsprechende kontinuierliche Anpassung der Simulationscodes. So mussten beispielsweise Compilervergleiche durchgeführt werden, da sich unterschiedliche Compiler auf die Rechengeschwindigkeit, aber auch auf das Zusammenspiel mit der jeweiligen MPI-Library ausgewirkt haben. Hier bestand regelmäßiger Kontakt sowohl mit den Rechenzentren, den Herstellern der Systeme, als auch mit den Herstellern von Analyse-Tools wie Debuggern und Performance-Analyse.

Die Ergebnisse des Projektes fließen somit auf unterschiedlichen Ebenen in die HPC-Community zurück: Mit den industriellen Partnern wird an den Anwendungen und dem Erkenntnisgewinn durch die Simulation gearbeitet. Hierfür sind effiziente Codes und numerische Algorithmen notwendig, die insbesondere im Austausch mit den anderen akademischen Partnern erfolgen. Einerseits wirkt dies über die Verbesserungen im Code nach, die natürlich auch über Projektende hinaus zur Verfügung stehen. Andererseits wird auf der Seite der akademischen Partner das Bewusstsein für mögliche Problemstellen geschaffen und Knowhow vermittelt, damit zukünftige Erweiterungen der Algorithmen und Software von vornherein unter Berücksichtigung der Performance und Effizienz erfolgen können. Im Bereich der Informatik erfolgt die Zusammenarbeit mit Tools-Entwicklern und Rechenzentren, um die Bedürfnisse der Nutzer einzubringen.

Für die tiefergehende Details sei auf die Erfolgskontrollberichte der einzelnen Partner verwiesen.

5. Während der Durchführung des Vorhabens dem ZE bekannt gewordenen Fortschritts auf dem Gebiet des Vorhabens bei anderen Stellen

Während der Durchführung des Vorhabens sind keine Fortschritte bei anderen Stellen bekannt geworden, die die Durchführung des STEDG-Projekts behindert und hinfällig gemacht hätten.

Weiterentwicklungen in einzelnen Bereichen finden sowohl auf der Seite der Forschungsentwicklung als auch auf der Seite der kommerziellen Software-Pakete wie Ansys CFX, Fluent, Gambit, ANSA u.a. statt. Diese Entwicklungen wurden aufgenommen, sowohl durch Teilnahme am wissenschaftlichen Austausch, beispielsweise auf Konferenzen, durch Papers und individuelle Zusammenarbeit als auch durch Verwendung und Abgleich mit der jeweils aktuellen Softwareversion, die auf den verfügbaren Installationen zugänglich ist.

Die Diskussion der im Projekt erzielten Ergebnisse wurde durch Veröffentlichungen und Vorträge, ebenso wie in anderen Kooperationsprojekten von den Konsortialpartnern in die wissenschaftliche Community getragen. Ebenso wurden die Ergebnisse den Fachabteilungen der Industriepartner zugänglich gemacht und die Anwendung in anderen Bereichen angesprochen. Die sich aus diesen Diskussionen ergebenden Erkenntnisse stellen in beide Richtungen (in das Projekt und aus dem Projekt heraus) eine Bereicherung des Projekts und der HPC-Community dar. Sie dienen als Grundlage für neue Forschungsansätze und Realisierungsideen.

6. Erfolgten oder geplante Veröffentlichungen des Ergebnisses

Veröffentlichungen:

- Harlacher, D.F., Siebert, C., Klimach, H., Roller, S., Wolf, F. **Dynamic Load Balancing for Unstructured Meshes on Space- Filling Curves**. In Proceedings of the Workshop on Large-Scale Parallel Processing, IPDPS 2012
- Harlacher, D.F., Klimach, H., Roller, S. **Turbulence Simulation at large Scale**. inSide
- Katharina Benkert, Bernhard Müller and Michael M. Resch. **Reducing Turn-Around times for supernova simulations**, Interdisciplinary Information Sciences, 15(1), 2009
- Edgar Gabriel, Saber Feki, Katharina Benkert, Michael M. Resch. **Towards Performance and Portability through Runtime Adaption for High Performance Computing Applications**, Concurrency and Computation: Practice and Experience 22(16): 2230-2246 (2010)
- K. Benkert, E. Gabriel. **Empirical Optimization of Collective Communications with ADCL**, High Performance Computing on Vector Systems 2010, 2010
- K. Benkert, E. Gabriel, S. Roller. **Timing Collective Communication in an Empirical Optimization Framework**, PARENG2011, The Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, P. Ivanyi and B.H.V. Topping (Eds), Ajaccio, Corsica, France, 12-15 April 2011, PENG11/2010/000052, ISBN 978-1-905088-44-7
- **Highly Efficient and Scalable Software for the Simulation of Turbulent Flows in Complex Geometries**, Daniel F. Harlacher, Sabine Roller, Florian Hindenlang, Claus-Dieter Munz, Tim Kraus, Martin Fischer, Koen Geurts, Matthias Meinke, Tobias Klühspies, Volker Metsch, Katharina Benkert, High Performance Computing in Science and Engineering '11, 2012, pp 289-307
- Daniel F. Harlacher, Manuel Hasert, Harald Klimach, Simon Zimny, Sabine Roller. **Tree Based Voxelization of STL Data**, In: Resch, M., Wang, X., Bez, W., Focht, E., Kobayashi, H., Roller, S. (Eds.) *High Performance Computing on Vector Systems 2011*, pp. 81-92, Springer, 2011
- Daniel F. Harlacher, Sabine Roller, Florian Hindenlang, Claus-Dieter Munz, Tim Kraus, Martin Fischer, Koen Geurts, Matthias Meinke, Tobias Klühspies, Yevgeniya Kovalenko, Uwe Küster **Industrial Turbulence Simulation at Large Scale.** , HLRS review 2013

- Sabine Roller, Jörg Bernsdorf, Harald Klimach, Manuel Hasert, Daniel Harlacher, Metin Cakircali, Simon Zimny, Kannan Masilamani, Laura Didinger, Jens Zudrop: **An Adaptable Simulation Framework Based on a Linearized Octree**, In: Resch, M., Wang, X., Bez, W., Focht, E., Kobayashi, H., Roller, S. (Eds.) *High Performance Computing on Vector Systems 2011*, pp 93-105, Springer, 2011, ISBN 978-3-642-22244-3, DOI 10.1007/978-3-642-22244-3_7
- Harald Klimach, Sabine P. Roller, Jens Utzmann, Claus-Dieter Munz: **Simulation of Automotive Injector Nozzle Noise with fully coupled CFD/CAA solver***,* Proceedings of the V European Conference on Computational Fluid Dynamics ECCOMAS CFD 2010 J. C. F. Pereira, A. Sequeira and J. M. C. Pereira (Eds) Lisbon, Portugal,14-17 June 2010. ISBN: 978-989-96778-1-4
- Roidl, B., M. Meinke and W. Schröder, **A zonal RANS/LES method for compressible flows**, Comp. Fluids, vol 67, p 1-15, 2012
- Florian Hindenlang, Jonathan Neudorfer, Gregor Gassner, Claus-Dieter Munz. **Unstructured three-dimensional High Order Grids for Discontinuous Galerkin Schemes**, 20th AIAA Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 27-30, 2011 , 2011.
- Gregor Gassner, Michael Dumbser, Florian Hindenlang, Claus-Dieter Munz. **Explicit one-step time discretizations for discontinuous Galerkin and finite volume schemes based on local predictors**, Journal of Computational Physics, 230, 11, Page(s): 4232 - 4247, 2011.
- G. Gassner, F. Hindenlang, C.-D. Munz. **A Runge-Kutta based Discontinuous Galerkin Method with Time Accurate Local Time Stepping**, Wang, (Ed.): Adaptive High-Order Methods in Computational Fluid Dynamics, World Scientific.
- F. Hindenlang, G. Gassner, T. Bolemann, C.-D. Munz. **Unstructured High Order Grids and their Application in Discontinuous Galerkin Methods**, Proceedings of ECCOMAS, 2010.
- F. Hindenlang, G. Gassner, C. Altmann, A. Beck, M. Staudenmaier, C.-D. Munz. **Explicit Discontinuous Galerkin methods for unsteady problems**), Computers&Fluids, 61, pp. 86-93, 2012, ISSN: 0045-7930.
- T. Kraus et al. „**Direct Aeroacoustic Simulation of near fid noise dring a gas injection process with a Discontinuous Galerkin Approach**, 18. AIAA/CEAS Aeroacoustics Conference in Colorado Springs (USA, CO)
-

Vorträge und Poster:

- Harlacher, Daniel. **Dynamic Load Balancing for Unstructured Meshes on Space-Filling Curves**. IPDPS 2012
- Harlacher, Daniel. **Highly efficient and scalable software for the simulation of turbulent flows in complex geometries**. HLRS review workshop 2011. Ausgezeichnet mit dem „Golden Spike Award“
- Roller, Sabine. **Simulation of turbulent flows under industrial constraints**. International Supercomputing Conference ISC 2012
- Harlacher, Daniel. **Comparison of stability and efficiency of high-order DG and WENO schemes for a super-sonic free jet**. Eccomas 2012
- K. Benkert, E. Gabriel. **Measuring Execution Times of Collective Communications in an Empirical Optimization Framework**, Poster at EuroMPI, 12.-15. Sept. 2010, Stuttgart, 2010
- T. Kraus et al. „**Direct Aeroacoustic Simulation of near fid noise dring a gas injection process with a Discontinuous Galerkin Approach**, 18. AIAA/CEAS Aeroacoustics Conference in Colorado Springs (USA, CO)
- S. Roller. **Multi-scale CFD for Aeroacoustics, Plasma Flows and Medical Physics**, Eingeladener Vortrag auf der ParCFD 2010, Taiwan
- S. Roller: **STEDG – Hocheffiziente und skalierbare Software für die Simulation turbulenter Strömungen in komplexen Geometrien**, Vortrag auf der 1. HPC Statustagung, Berlin, 2010
- S. Roller: **STEDG – Hocheffiziente und skalierbare Software für die Simulation turbulenter Strömungen in komplexen Geometrien**, Vortrag auf der 2. HPC Statustagung, Darmstadt, 2011
- S. Roller, Harald Klimach, Daniel Harlacher: **Fluid dynamics with multi-scale processes: modelling and HPC**, **Invited Talk**, Munic Multiphysics Meeting, 13 July 2011, Munich Centre of Advanced Computing (MAC)

- S. Roller, Harald Klimach, Daniel Harlacher: **Adaptive Multi-Scale Simulation of Turbulent Flow and Acoustics**, ECCOMAS Coupled Problems 2011, 20-22 June 2011, Kos Island, Greece
- K. Benkert, E. Gabriel, S. Roller: **Timing Collective Communication in an Empirical Optimization Framework**, PARENG2011, The Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, Ajaccio, Corsica, France, 12-15 April 2011
- Sabine Roller. **Strategien zur Effizienz-Optimierung – Widerspruch zwischen Memory- und CPU-Load Balancing**. Workshop "Developments in HPC", Leogang, 2011
- Sabine Roller: **Multi-Scale Methods for CFD-Applications in the Face of HPC, International Mini Symposium for Biomechanics of Intracranial Stent**, Tohoku University, Sendai, Japan, 20. Oktober 2010
-

Dissertationen:

- K. Benkert: **Adaptive parallel communications for large-scale computational fluid dynamics,** Online verfügbar: http://elib.uni-stuttgart.de/opus/volltexte/2012/7020/
- D. Harlacher: **Efficient HPC models for turbulent flow simulation**, in Vorbereitung, Einreichung August 2013
- T. Kraus: in Vorbereitung, Einreichung 2013
- F. Hindelang: in Vorbereitung, Einreichung 2013
- Y. Kovalenko: in Vorbereitung, Einreichung 2013

Masterarbeiten:

- Betreuer: Harlacher, Daniel. Student: Krupp, Verena. **Influence of artificial viscosity on the accuracy of LES modeling**. Oktober 2012

Technische Berichte:

- Projektabschlußbericht

Hocheffiziente und skalierbare Software für die Simulation turbulenter Strömungen in komplexen Geometrien

Wissenschaftlicher Teil des Schlussberichts

**Daniel F. Harlacher, Sabine Roller,
Florian Hindenlang, Claus-Dieter Munz,
Tim Kraus, Martin Fischer,
Koen Geurts, Matthias Meinke,
Tobias Klühspies, Volker Metsch,
Yevgeniya Kovalenko, Uwe Küster**

GEFÖRDERT VOM

25/06/2012

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IH08010 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Acknowledgment

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Färderkennzeichen 01 IH 08010 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

The project STEDG was funded by the German Federal Ministry for Education and Research (BMBF) in the call "HPC Software for scalable Parallel Computers". We thank the Gauss Centre for Supercomputing (GCS) which provided us with the necessary resources on different HPC systems.

Contents

1. Introduction and Outline of the Project

1.1. Motivation and Starting Point

Numerical simulation of fluid flows is nowadays an indispensable method for research and development in all areas of engineering. Over the years, it became a key technology to improve economic, environmental and security behavior of products, thereby vitally strengthening the competitiveness of German industry. Nevertheless, it often uses simplified methods to obtain results within the given turn-around times. The simulation of turbulent flows is often based on Reynolds-averaged Navier-Stokes equations (RANS). Here, the algorithms give stationary solutions, averaged in time. On the other hand, time-dependent phenomena often play an important role in industrial applications. Research and development departments carry out first calculations, based e.g. on Large-Eddy simulations (LES). But the commercial software packages used, work with numerical methods suited for traditional stationary simulations with turbulence model. They have a broad application spectrum, but are much to costly for in-stationary processes.

The efficient simulation of in-stationary fluid flows with higher order turbulence modeling requires an overall renewing of all components: numerical methods, algorithms and their efficient implementation on modern hardware architectures. While several groups work intensively on new numerical methods, the changes in hardware of future computer generations remain mostly unconsidered. Current developments in hardware architecture of high performance computers require dramatic changes in the programming model of simulation codes to be able to make use of these new architectures.

Necessary is a paradigm change in the comprehension of the users: in the past, users could expect an increase of performance of their simulations directly with the performance of the hardware. In the future, this is not directly the case. The power and performance of computer systems is no longer increasing by higher clock frequencies of single processors, but through a more and more increasing number of processors - with respect to increasing number of nodes as well as increasing number of cores within one node. Future architectures will be systems of shared memory nodes, which is currently hardly used by application codes. Also the increasing gap between processor speed and memory bandwidth is hardly ever a topic discussed at user conferences.

The STEDG project addresses the changes, software development has to catch up to keep track with hardware development. Current new numerical methods - by structure well suited for these computer architectures - are further developed to efficient flow

simulation tools. On the one hand, future numerical methods and algorithms, on the other hand their mapping to current (multi-core) and future (many-core) hardware architectures are considered. They are applied to real-life applications from industrial research and development (R&D) and their usability for industrial development cycles correlated with current and future developments.

1.2. Outline

This project considers the process of developing software for the numerical simulation of currents as a whole. Modern algorithms for problems that can not be numerically simulated yet are implemented in a way that allows them to take advantage of future hardware architectures. The simulation of real world applications with higher order turbulence modeling is only made possible by combining new numerical methods with an efficient implementation that is adapted to modern hardware architectures. The potential of this approach will be tested on topical real world problems provided by industry partners. In short: this project is about future numerical methods which are efficiently implemented on future hardware architectures and applied to future problems and questions.

The goals of this project contain:

- Numerical modeling of turbulent currents using time-precise numerical methods.
- Methods for analyzing and optimizing the code with respect to hierarchies in the memory and/or network of hardware architecture.
- Improving the scalability of the algorithms within nodes as well as between nodes.
- Determining and evaluating cost-factors to optimize the load-balancing of highly adaptive methods (with respect to space, time and precision).
- Evaluating and validating the developed methods on the basis of big test cases with relevance for the industry.
- High value turbulence models that are based on the concept of Large-Eddy-Simulation are being used to describe complex currents.
- The numerical methods are tuned to in-stationary currents and allow for an explicit discretization of time which decreases both the computing time and the amount of communication between parallel processes.
- Using highly local methods with minimal data exchange with neighboring elements the application should scale perfectly on new hardware architectures.
- This allows industrial applications to be simulated within acceptable amounts of time. Furthermore this will help compute problems that could not be simulated up to now.

A prototype for this kind of problem is the process of injection in a gas-powered combustion engine. Here, the gas enters the injector tube as a supersonic jet where it is deflected by the cross flow and hits the tube wall while interacting withe reflected shock waves. The numerical simulation of the whole process with conclusions about the creation and possibly the reduction of sound is a central application in this project. Up to now only parts of this problem can be simulated within a computing time of 10 000 CPUh which is too much for use in the design process.

Another problem concerns the aerodynamics and aeroacoustics of an elevator unit of a plane. If this problem is to be solved in three dimensions by a hybrid fluid mechanics/acoustic approach, a computational grid consisting of 5^8 cells is necessary. The amount of memory and computing time needed is considerably large.

The third application is the optimization of a nozzle for laser beam cutting. Here, a complex supersonic jet with shocks appears in a complex geometry and the turbulence is not as challenging as these shocks and the complex geometry.

To reduce the amount of computing time needed from multiple weeks to several days the use of HPC-architectures is inevitable. Modern architectures are dominated by dual-core and quad-core layouts. Soon, nodes with up to 32 cores will appear, eventually even with 100 cores or more. The deployment of these many-core processors in clusters will lead to systems with thousands of (heterogeneous) cores. In this project the programming models and especially the parallelization models will be adapted to the technological development. Challenges for the simulation tools will arise regarding the memory and the memory-hierarchies. The goal will be to prevent a decrease of performance caused by the difference between processor speed and memory bandwidth.

2. Outline

This report is structured as follows: In Chapter 3 the numerical schemes which are used in context of the STEDG project are introduced in short to give a more general understanding of the rest of the report. Both presented schemes are capable of capturing turbulent phenomena in a time-accurate manner. In Chapter 4 the evaluation and optimization of the above mentioned numerical schemes is presented. In this context not only serial performance was evaluate and optimized but also the behavior on highly distributed supercomputing systems were investigated. In detail the chapter deals with the following:

- An introduction and definition of a new performance metric is given, which is capable of relating the necessary information for the behavior of a code on a given system to the user
- A short overview over the tools that were used to debug and optimize the used codes with respect to memory and run-time is given
- An overview is given over the performance bottlenecks that where found within this project and how the code benefited from their removal.
- The performance for the used codes is evaluated in great detailed on supercomputing systems with up to 100.000 cores.
- A partitioning algorithm based on space-filling curves is introduced which was developed within this project to optimize the partitioning of the computational domain before and during run-time onto multiple processes. The benefits of this algorithm are introduced and the application to one of the project codes is shown. Additionally a new way to determine accurate cost factors for an efficient load-balancing is presented.
- Also based on space-filling curve and closely related to the above mentioned partitioning algorithm a preprocessing scheme is introduced in which a given mesh from a commercial mesh generation tool is converted into a format which becomes easily and efficiently readable for highly distributed systems.

Chapter 5 presents test-cases in which the general physical correctness of the results for the HALO code for turbulent cases is investigated. In Chapter 6 to 7 three large industry test-cases are, which so far could not have been simulated within reasonable time-frames are presented and evaluated.

3. Numerical Methods for Time-Resolved Turbulent Flows

3.1. LES with DG

The flow solver of the Institute for Aero- and Gasdynamics (IAG) at the University of Stuttgart is called HALO and solves the compressible Navier-Stokes equations for three-dimensional compressible flows. It is a discontinuous Galerkin scheme, the code is fully MPI parallelized [21] and runs on unstructured meshes consisting of hexahedra, prisms, pyramids and tetrahedra. The solution is represented on each element by a polynomial of arbitrary degree. A special explicit time discretization is employed, the so called time-consistent local time stepping [22, 8, 9].

Two types of LES approaches were implemented. An explicit Smagorinsky model, where the amount of added viscosity is computed from the local velocity gradient tensor. The second is an implicit approach, derived from a technique which is already used in HALO for shock capturing. An spectral indicator is evaluated locally for each grid cell, representing the amount of polynomial oscillation, which is an indicator for an under-resolved solution. Proportional to the oscillation, artificial viscosity is added to the viscous fluxes. The viscosity damps the oscillations and stabilizes the simulation. Both approaches were investigated in this project, whereas the implicit approach was used in the industrial test cases, since more stabilization was needed because the flow developed both shocks and turbulence.

3.2. Zonal RANS-LES

The flow solver of the Institute of Aerodynamics Aachen (TFS) solves the Navier-Stokes equations for three-dimensional compressible flows on a block-structured grid. A modified AUSM method as introduced in [20] is used for the Euler terms which are discretized to second-order accuracy by an upwind-based approximation. For the non-Euler terms, a centered approximation of second-order accuracy is used. The temporal integration is done by a second-order accurate explicit 5-stage Runge-Kutta method with coefficients optimized for maximum stability [25]. The sub-grid scale modeling for the large-eddy simulations is based on an implicit ansatz, i.e., the MILES (monotone integrated LES) approach of Boris *et al.* [4]. For the RANS zones the Spalart-Allmaras turbulence model was chosen to close the Reynolds-averaged Navier-Stokes equations. To reduce computation time, the solution methods based on RANS and LES can be combined into a zonal method. The LES regions are used to resolve the leading and trailing edge region where flow separation occurs, while the RANS zone is used for the

attached flow regions. The schematics of the overlapping zones is shown in Fig. 3.1 and 3.2 , where the values that have to be communicated back and forth between the two different approaches are indicated. In the overlapping region, where the flow is directed from a RANS to LES zone, synthetic eddies are introduced to accelerate the generation of coherent turbulent structures using the method of Jarrin [16]. Furthermore, control planes are used to drive the solution towards the correct turbulence level in the LES domain according to Spille and Kaltenbach [31]. When the flow is coming from the LES into the RANS domain, the RANS requires a definition of the eddy viscosity ν_t, the value of which is reconstructed from time and spatial averaging of the LES data.

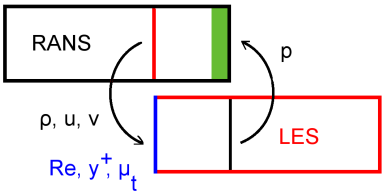


Figure 3.1.: RANS-to-LES transition

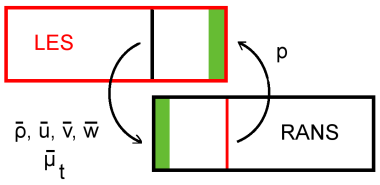


Figure 3.2.: LES-to-RANS transition

4. High Performance Computing

Programs like an academic fluid simulation code are developed over years and usually with a broad applicability spectrum in mind. So it happens that codes that where designed and/or optimized for a certain architecture might not run as efficient on current and/or future machines. In the worst case a paradigm shift in architectures (e.g. from NUMA- or vector-machines to distributed memory machines) inhibits the code to run efficiently at all, in these cases a complete rewrite of the code might become necessary. On the other hand, with the increasing demand of parallelism the problematic of efficient code writing gets even more complicated. Code parts that scaled very well to moderate number of processes might become a bottleneck when thinking about scaling to hundreds of thousands of processes. In addition to that even seemingly small changes in the supercomputing architectures might have a great impact on performance. Therefore a crucial part of software-engineering has become the performance analysis in the sense of both a way to find bottlenecks and remove them to enable higher parallel efficiency and the way in judging the performance of the code on a certain architecture and predict the performance on an other machines.

4.1. Performance Metrics

Numerical simulations are carried by floating point operations (FLOP). Therefore we introduce a performance metric called performance map which is based on the floating point operations per second (FLOPs) a code achieves for a given problem size on a given number of processes on a given architecture.

4.1.1. Serial Performance Map

For the serial performance map the FLOPs over the problem size are plotted. With this the behavior of the code on the processor can be judged. Figure 4.1 shows a typical serial performance map. On the y-axis the problem size is denoted while on the x-axis the performance of the code is plotted. Most notable perhaps is the fact that the smallest problems do not run the fastest. This is due to the execution overhead of the used program. With increasing problem size the performance therefore rises in the beginning as the relative overhead of the program becomes smaller. Overlapping this general effect different other behaviors can be identified, e.g. it becomes clear that the caches of the CPU are filled one after the other. Depending on the overhead of the application these effects can be seen in more detail or not. With most codes one sees a peak in performance followed by a notable drop in performance - once the problem size does not fit into the cache anymore. After dropping out of cache the performance

will continue to increase slightly due to ever decreasing overhead and more efficient filling of pipeline instructions. The end of the performance plot shows the maximum problem size that can be used on the investigated architecture.

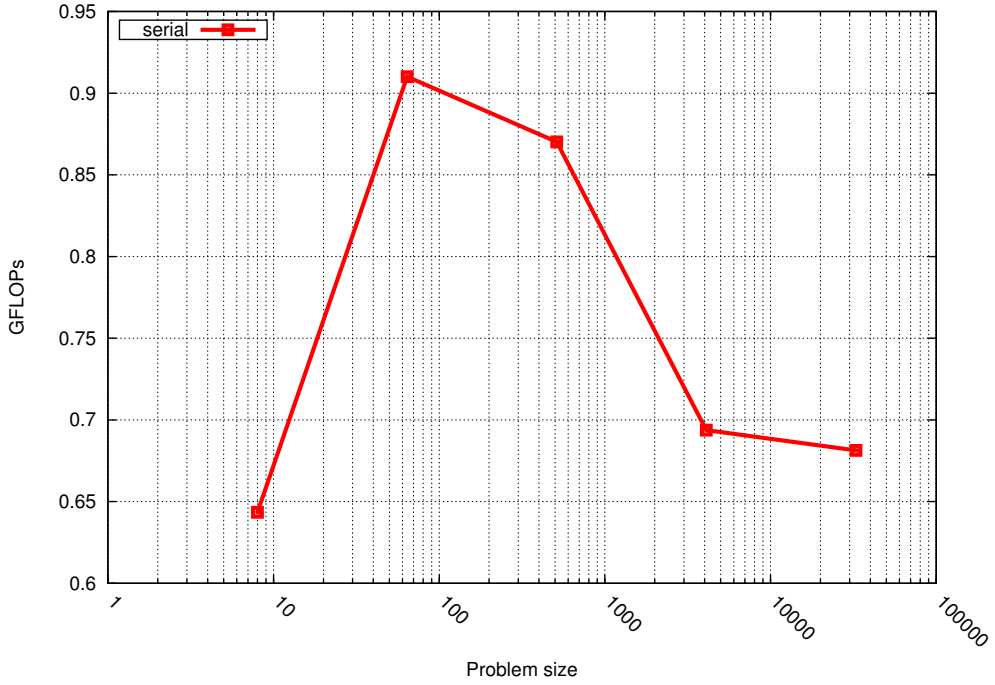


Figure 4.1.: Example for s serial performance map.

4.1.2. Parallel Performance Maps

Going from serial to parallel one would expect similar behavior when thinking of performance. But in general the parallel performance is not easily comparable to the serial due to additional influential factors in the code and the architecture. Therefore it can not be assumed that a serial code runs times X faster with X processes. These factors being communication on the code side which can not only lead to more application overhead but also may change the performance behavior. Additionally it has to be distinguished between inner-node and intra-node communication as those might be implemented separated from each other - OpenMP/MPI hybrid application - and therefore might have different individual impacts on performance. The architecture of a system can also influence the performance due to private caches on CPU level up to communication bandwidth between the compute nodes. With both code and architecture presenting different kinds of parallelism a pair of performance maps is

introduced in the following to separate the effects on performance.

Intra-node Performance Map

The assertion of the intra-node performance map is how many cores of a compute node can be used efficiently on a given computer architecture. Therefore the different curves in figure 4.2 represent the different number of processes on which the corresponding problem size was run. With this the metric directly reflects the gains and/or losses in performance when using a certain number of cores on a node. With the baseline being one process, which corresponds to the serial performance map the metric overall shows the same features (4.1.1) in each curve but they differ in distinctiveness and location.

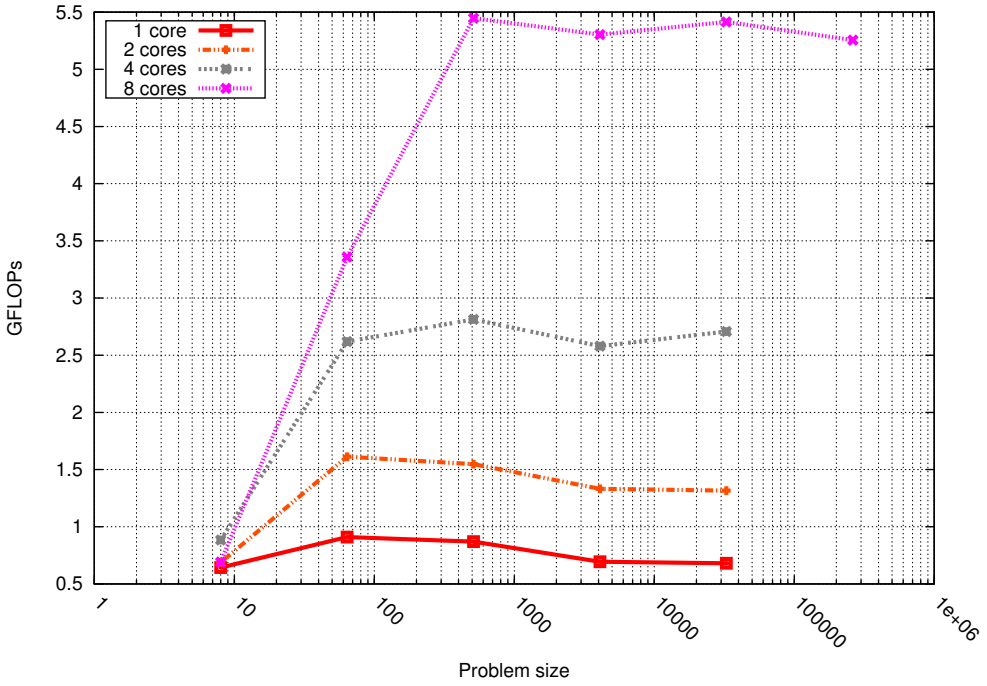


Figure 4.2.: Example for an intra-node performance map.

Inter-node Performance Map

Using the most efficient node setup resulting from the intra-node performance map (4.1.2) the inter-node performance map is generated. This metric reflects both the influence of the network of the machine and the inter-node parallelization of the code on the performance. Figure 4.3 displays this metric for a specific machine. Similar to the intra-node performance map the x-axis denotes the problem size but this time

it is the local problem size per node. On the y-axis the reached performance of the code is plotted - again FLOP/s are used. A typical inter-node performance map shows a common trend in the lines which gives back a lot of information. After a slope in which the problem size per node is not large enough for efficient execution on the architecture the trend reaches a plateau. The transition of the slope to the plateau marks the minimum problem size per node which should be reached to use the resources of the machine efficiently. After the plateau, which for perfect scaling codes would go on forever the map can show some (mostly negative) influences of code, network, etc ...

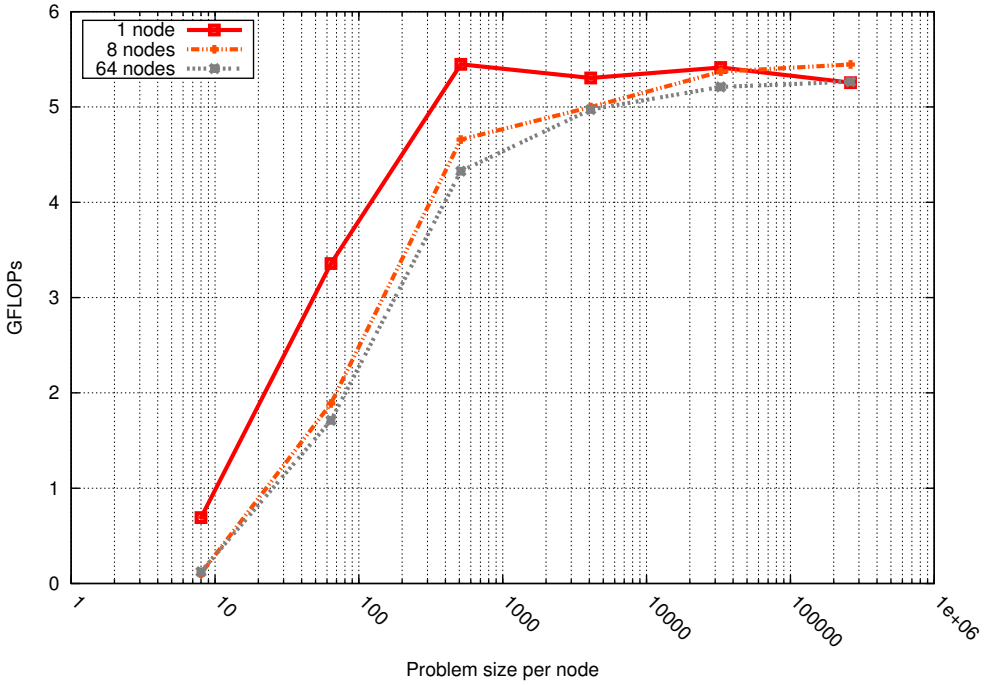


Figure 4.3.: Example for an inter-node performance map.

To extract information on weak scaling from the performance map one simply has to compare the performance of a given problem size between the different curves. A code which shows perfect weak scaling would show the same performance for a given problem size independent of the number of cores, therefore leading to an overlap of all curves in that point. For strong scaling one has to pick the problem size on the "one node" curve and then follow the chart from right to left while changing to the curve of the desired number of nodes.

4.2. Performance and Debugging Tools

A number of tools are introduced which help to analyze the behavior of codes on different supercomputing architectures. Some of them are used to measure performance, others to visualize the communication or to detect potential problems.

Likwid

Likwid consists of a set of command line tools which enable easy access to hardware-counter information such as FLOPS, clock-frequency, etc ... In this work the tool Likwid was used to count the floating point operations of applications.

Vampir

Vampir is a parallel profiling tool which uses a code instrumentation in the form of compiler wrappers to extract information like communication and others. With its time-line view over all processes indicating the communication using a visual representation in form of lines the tool vampir can be used to investigate communication-heavy parts of the code visually. Therefore enabling a quick assertion of the code and a fast detection of possible sections in the code that might impact parallel performance.

Valgrind

The Valgrind toolsuite is a collection of tools which help you analyze a code on a very detailed level. Valgrind tools are based on the emulation of a CPU and can therefore give information about cache-alignment and cache-misses as well as memory-demand and other information. In the following only a few of the many tools offered by Valgrind are introduced.

massif The tool massif allows you to track the memory consumption over the elapsed simulation time. The consumed memory is subdivided into the responsible functions and subroutines in which the memory is allocated. Therefore tracking of a memory leak gets more comfortable.

memcheck The tool memcheck is useful to track down memory errors like accesses to uninitialized values or missing deallocation. It can also detect some failures in the MPI usage.

4.3. Performance Optimizations

In section 4.1 metrics where introduced by which the behavior of a code can be evaluated on a certain architecture. If that behavior is unexpected or unsatisfying the code needs to be investigated more thoroughly. After identifying a potential bottleneck in the code using performance analysis tools or otherwise one needs to go into the code and exchange the code-part which is responsible. This section shows how performance

bottlenecks that where identified within this work, how they were assessed and how resolving these bottlenecks led to higher code performance.

4.3.1. General optimizations

Serial execution efficiency of the HALO-code was analyzed on the NEC Nehalem Cluster. The analysis has shown that the timing relationships between different functions depend heavily on the input parameters. Some functions change their influence according to the chosen computational order. Thus, several code examples were analyzed for different parameters (input parameter spacial order of the scheme: 2, 4, 6). With the help of Intel Vtune Amplifier XE 2011 the bottlenecks of the application could be identified.

Most sequences are inefficient due to the complicated nested data structures including pointers, which are not transparent to the compiler. Therefore compilers are not able to optimize the code to an acceptable degree. Global changes in the data model would result in considerable changes to the overall program itself. Thus, the data model was modified only in the most time-consuming places, which were consuming about 80% of the running time of the example using the LES model. In those functions the intrinsic MATMUL-function was replaced by unrolled loops, data types were changed from Fortran pointers to Fortran allocatables.

Those changes allow the result for both better hand and more aggressive compiler optimizations. Improvements in the running time of these functions of up to 70% have been achieved. The reduction of the total execution time by using Intel 12.1.3 compiler was 57% and by using GNU 4.6 compiler was 40%. The run-times pro function for described example with LES model enabled for order of the scheme 6 and number of DOFs 1512 can be found in table 4.1.

Functions	Original Code $[s]$	Modified Code $[s]$	Run-time reduction
rkck	212.66	76.61	63.98%
scaleseparatetderivs	142.42	44.86	68.50%
evalu_t_fast	12.95	8.54	34.09%
finalizedof	11.3	10.83	4.18%
calcvolint	10.9	9.78	10.28%
calcsurfint	8.8	6.16	30.02%
evalutilde_t_fast	8.2	7.71	6.01%
evalflux	8.07	6.4	20.70%
...	...	...	...
Total	439.51	186.2	57.63%

Table 4.1.: Comparison of the serial run-time for the original and modified codes for the example with enabled LES, order 6, number of DOFs 1512, compiled with the Intel 12.1.3 compiler

Some more optimizations were targeted to minimize the negative impact of chosen

data model. With this model mostly computational loops have small number of iterations, but loops are three or more times nested. That limits the full usage of the vector registers, the loop overhead becomes more significant and implies longer run-times. In those cases some code modifications were done to help the compiler generating more efficient code. Thus on some places loop collapsing techniques were used, to replace multidimensional by single dimensional addressing and nested short loops by a single long loop. With this optimization technique we achieved run-time reduction for certain loops of 90%.

As already mentioned the usage of nested pointer data structures (derived types in Fortran) is common for the HALO code. It's expensive to calculate the address of an array in such data structure, even more if it is used within the loop. So we have tried to minimize the addressing costs by making a local copy of the innermost array or by passing the address of such array, instead of the pointer to the structure by calling the function. Owing to this the pointer array will be handled as usual array.

Also some essential loops were switched between different sizes, unrolled by hand where possible, fused or simplified. Short arrays were replaced by variables. On some places the calling overhead of short procedures was significantly more expensive, as the procedure itself. Thus such short procedures were inlined with the help of compiler directive or by hand, when the compiler was not able to inline itself.

Those code modifications enabled better compiler optimization. Table 4.2 shows the comparison of run-times between original and modified code for different scheme orders and number of DOFs for examples with enabled and disabled LES. Due to existing dependency between timing relationships of different functions and input parameters, some functions change their influence according to the chosen computational order. So accordingly some optimizations of functions change their influence too. If two first functions (rkck and scaleseparatetderivs) for example with enabled LES, order 6 were taken 80% of all execution time, then for order 4 their part is already 48%, and for order 2 — only 18%. So essential run-time reduction of application for order 6 by 57% reduces to 33% for order 4 and to 21% for order 2. For example with disabled LES model described optimizations allowed to decrease the run time by 25%. Further improvements would affect the data model and would imply large changes of the entire application.

4.3.2. Parallel I/O

One great challenge in parallel computing is the I/O. In most cases accesses to input/output devices (e.g. reading from and writing to disc-storage) are much slower than memory accesses. Parallel I/O may require additional synchronizations of the hardware and the operating system. As a result whenever possible, we should avoid to read data back that was previously written, especially if multiple processes are involved in the I/O operation.
Furthermore for calculations on architectures with distributed memory, one process can probably only read a part of a huge data set due to memory restrictions. Even if the data set would fit in the memory of a single process, reading data with only one process and distributing it afterwards may create a bottleneck.

Example	Order	DOFs	Original Code [s]	Modified Code [s]	Run-time reduction
LES model enabled	6	1512	439.51	186.2	57.63%
	4	2500	209.04	139.09	33.46%
	2	4000	71.87	56.22	21.78%
LES model disabled	6	1512	86.96	69.32	20.29%
	4	2500	78.92	59.18	25.01%
	2	4000	42.94	35.5	17.33%

Table 4.2.: Comparison of the serial run-time for the original and modified codes, compiled with the Intel 12.1.3 compiler

Example: Parallel mesh input

A common strategy for years was to have one process read in the entire mesh, partition it with an algorithm (e.g.ParMetis, SPartA) and distribute the mesh to all calculating processes. This strategy is no longer applicable when thinking of always increasing problem sizes and the parallel trend of decreasing memory per core in the supercomputers. Therefore a strategy needs to be deployed that enables a fully parallel read-in from disk.

Example: Record-point output

The HALO code allows you to obtain continuous information about the state at specific points in the computational domain. These points are called *record-points*. The code needs to output the state vector at each record-point for each (local) time-step. In contrary the state of all elements in the computational domain is only written to disc in a specified output interval.
The record-points were implemented in the HALO code in the following way: Each record-point stores a list of state values and the points in time when these state values were recorded. Every time the state of an element with record-points is updated, the state vector is evaluated at the specific record-points and the result is appended to the corresponding list.
At the next output time all recorded data is written to disc. The previous implementation stored the data of all record-points on all processes in a single CGNS-file. For this purpose each process with active record-points opened the CGNS-file, appended the locally recorded data and closed it, one at a time.
On many processes this procedure becomes very slow, because each process must wait until all previous processes (processes with smaller id) finish writing their data. The fact that all processes also need to read a part of the CGNS-file may intensify the problem due to the necessary synchronization (handled by the operating system).
The new implementation uses one binary file per process, so each process just needs

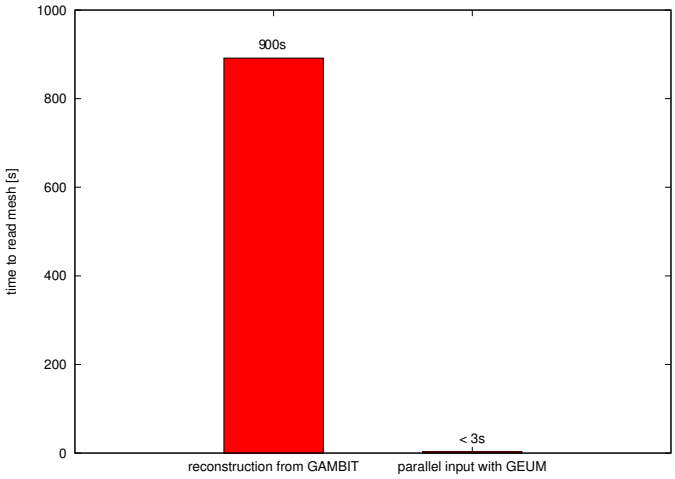


Figure 4.4.: Different methods to setup the internal mesh data structure (test-case with 4 million elements, calculation with 1024 processes on the JUROPA, for the parallel reconstruction 2048 cores were reserved to increase the available memory per process (explanation see communication matrix/infiniband buffer))

to append the locally recorded data to a file without reading from it. This minimizes the overhead of the record-point output during the calculation. In a post-processing step the user can convert the binary files to the desired format (e.g. CGNS) for visualization.

4.3.3. Parallel algorithms

The development of algorithms that are suited for many-core architectures depict another great challenge. Even if an HPC program runs fine with a specific number of processes, there may occur problems when the number of processes is increased.

Example: Record-point input

An example consists in the input of record-points in the HALO code. The setting is quite simple: The input file lists record-point coordinates. Then we need to map each record-point to a specific element on a specific process that contains these coordinates. In the previous implementation a 'marching search' algorithm was used. In short: The first process selects its first element and checks if the desired coordinates lie inside the element. If not, it selects a neighboring element in the direction of the given coordinates and so forth. If the neighboring element lies on another process, it notifies this process, so the 'marching search' can be continued on the next process. As a result, all processes except one are waiting for a notification.

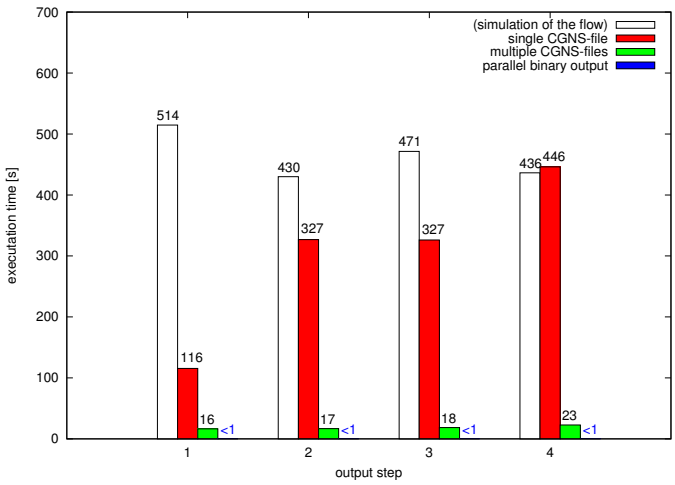


Figure 4.5.: Different methods to output record-point data (test-case with 4 million elements and ∼240 record-points, calculation with 1024 processes on the JUROPA)

This means the whole search algorithm is serialized and the communication overhead increases with the number of processes. For a low to a medium number of processes this is not a problem: The algorithm is only executed once when the calculation is started, so in this case its run-time is negligible in relation to the total computation costs.
However for massively parallel computations it may dominate the whole calculation; we observed a significant overhead for calculations with more than 512 processes: For example on a mesh with 0.7 million elements and 230 record-points it takes ∼1 minute to insert these points in the mesh with 512 process on the JUROPA. With 1024 processes only about 20 points are read withing ∼2 hours, with 2048 processes only about 10 points in several hours.
In the new implementation all processes read the record-point coordinates from the input file and check if the coordinates lie inside the bounding box of the local mesh. Then each process determines the specific elements of the record-points on their part of the mesh using a coarse background mesh (a bucket search approach in 3D). This way, the search algorithm is parallelized without any communication and due to the bounding box test each process just has to search for a small part of the record points. Depending on the element distribution, the new implementation should scale nearly perfectly. More importantly, the computation time needed to setup the record-points is now insignificant (several seconds on 2048 processes) compared to the total computational costs of a calculation.

Example: Element redistribution after dynamic load-balancing

For dynamic load-balancing an algorithm is needed that redistributes the elements between the different processes. The problem to determine a good distribution is discussed in section 4.6.
In the following we describe the task to exchange the data of a set of elements: In the case of the HALO code, the data per element consists in the elements coordinates, the state vector from the last time-step, information about boundaries, etc... The size of the data of different elements varies, because of different element types, different order of the local solution or boundary conditions. That is why we cannot easily solve the task by a single all-to-all communication statement.
In the previous implementation, each process determines the data of the elements that need to be sent to another process in a first step. Then all processes exchange the necessary data pairwise in a predefined ordering. This approach has similarities to a soccer-tournament; each process communicates in each round with a single other process. During this pairwise communication step the two processes send at first the size of the data that the other process will receive and allocate corresponding buffers. Afterwards they exchange the necessary data.

Figure 4.6 shows the resulting communication of the algorithm: At the beginning each process determines which data to send to which process (until ∼8.45 s). In the middle we can clearly recognize the round-based communication steps (from ∼8.45 s to ∼9.85 s). Afterwards the processes need to setup the newly received elements for the calculation and wait for all other processes to finish at the end.
This approach leads to several problems:
Due to the fixed order of the communication partners, a process must probably wait a long time until previous rounds of the tournament have finished.
This creates idle times that increase highly with the number of processes.
Additionally, every process needs to communicate with every other process, even if they do not have to exchange any data. This may cause a large memory consumption for communication buffers (see section 4.3.5).

In the new implementation these problems are avoided: In a first step every process determines the number of elements that it needs to send to every other process and to receive from every other process.
When the number of elements to send to each other process is known, we can achieve this using a single MPI_ALLTOALL operation (which may be implemented efficiently for small message sizes, see section 4.3.5). In the following there is only communication between processes that really need to exchange any data: Every process opens incoming, non-blocking connections to receive the necessary buffer sizes. Then it determines the size of the buffers needed for its local elements and sends these to the corresponding other processes. This step is necessary, because the size of the data depends on the properties of the elements. So all processes can allocate buffer storage for the incoming data and open non-blocking, incoming connections. Afterwards they serialize their data and send them to the target processes.
In Figure 4.7 we can see the results of the new algorithm: In the left part (until 9.7 s) the processes determine the number of elements to send to each other and to receive

Figure 4.6.: VAMPIR communication time-line of the element redistribution using a round-based approach with 128 processes on 64 hyperthreaded cores on the JUROPA

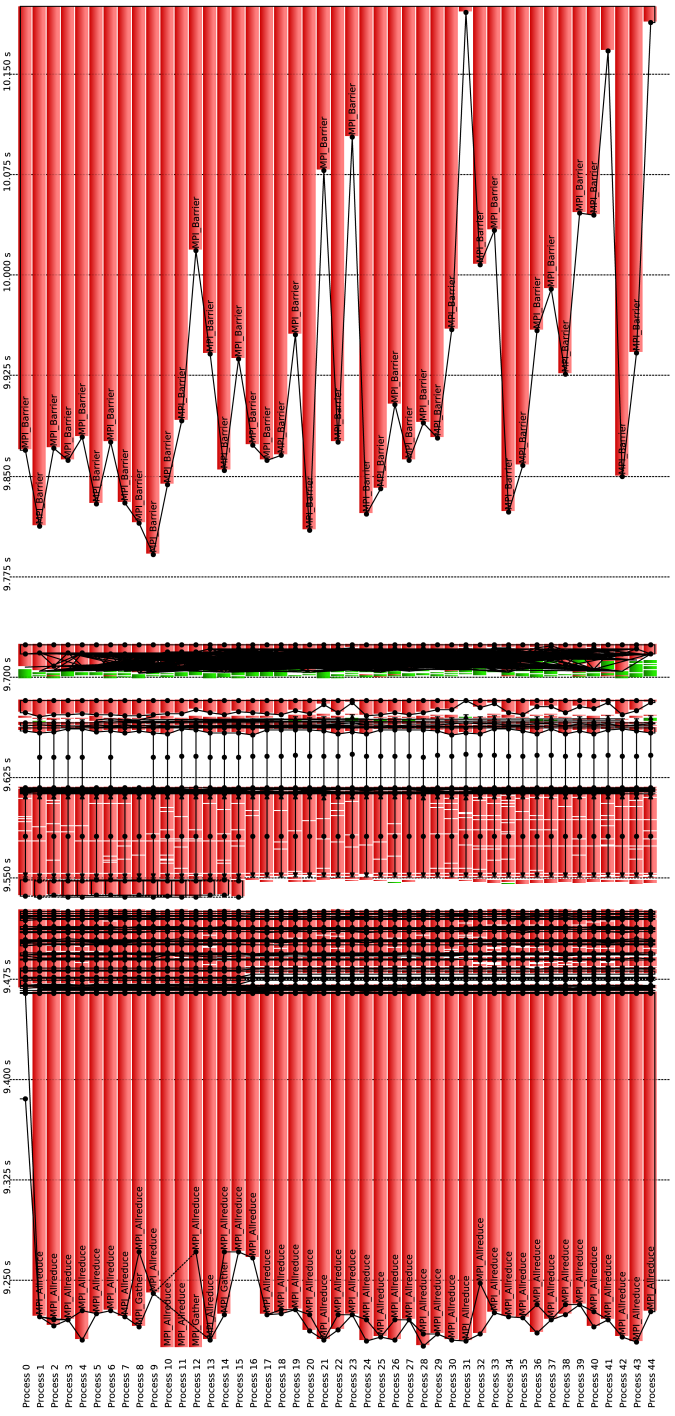
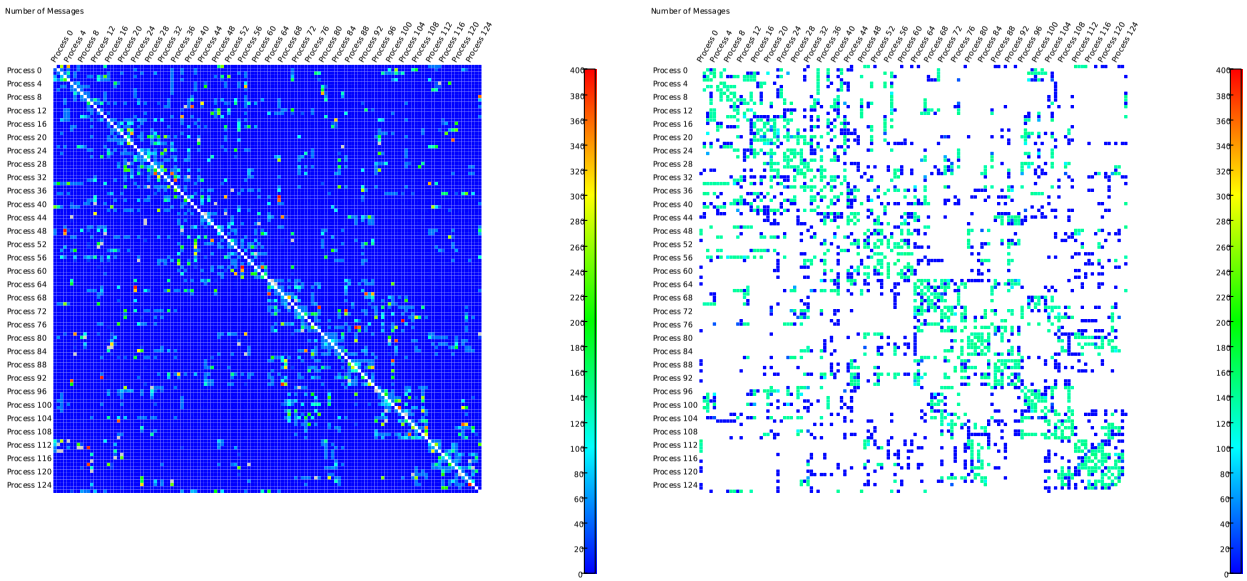


Figure 4.7.: VAMPIR communication time-line of the element redistribution using sparse non-blocking communication with 128 processes on 64 hyperthreaded cores on the JUROPA



(a) previous implementation using a round-based approach

(b) new implementation using sparse non-blocking communication

Figure 4.8.: VAMPIR visualization of the communication matrix of the element redistribution with 128 processes on 64 hyperthreaded cores on the JUROPA

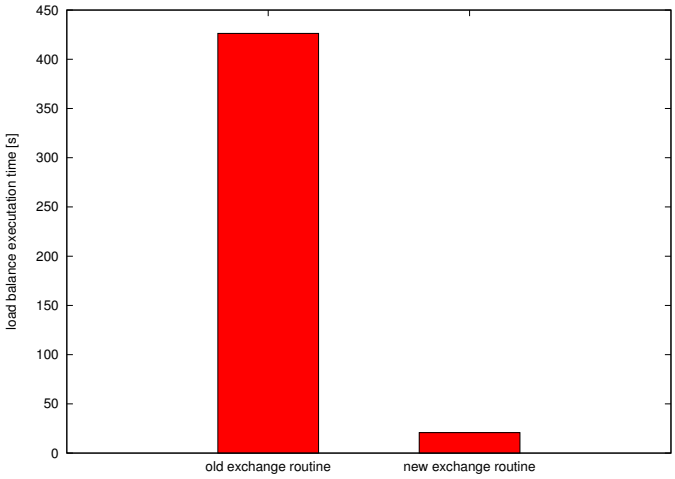


Figure 4.9.: Execution time of the load-balance routine (test-case with 4 million elements, calculation with 1024 processes on the JUROPA)

from each other. If a space-filling is used for load-balancing (see section 4.6), then most of the communication can be omitted: In this case we can easily derive the number of elements to send and receive from the element-offsets of the processes.
In the middle (at about 7.5 s) we see the result of the actual redistribution of the elements. Since all processes open the necessary incoming connections in advance using appropriate buffers, we can send all data at once. This prevents unnecessary idle times. Additionally there is no communication between processes that do not need to exchange any data (see Figure 4.8).A possible drawback of the new implementation lies in the fact that it needs to allocate all buffers at once. In the case of the HALO code this should not cause any problems, because the memory consumption during the actual calculation is much higher due to the storage needed for intermediate values.

As a result the new implementation prevents problems due to self-implemented collective communications (see subsection 4.3.5) and it is much faster, especially on many processes (see Figure 4.9).

4.3.4. Memory performance

On architectures with distributed memory the available memory per process may be another problem. So a good memory scaling of the application is crucial due to the previously mentioned trend of decreasing memory per core (for an increasing number of available cores).
In this work we discovered essentially two possible sources of bad memory scaling: communication buffers and the serial preparation of necessary data on a single process.

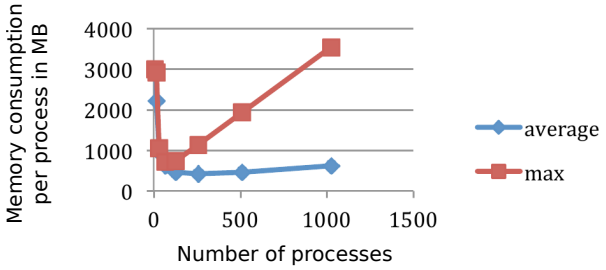


Figure 4.10.: Memory consumption of the HALO code during the startup-phase

Communication buffers

The first problem is caused by the underlying MPI implementation; it allocates a communication buffer for each other process under certain circumstances (point-to-point communication with each other process). For a detailed discussion see section 4.3.5. As a result we can run the quarter of a test-case (Bosch injector with 0.75 million elements, fourth order) on 512 cores on the JUROPA, but it does not work on 1024 cores. Moreover we can not run the full configuration of the test-case with 3 million elements on the JUROPA except when we reserve additional cores without using them to increase the available memory per process.

Example: Memory consumption of the HALO code during the beginning of a calculation

The second problem can be observed during the startup-phase of calculation with the HALO code. To get a rough overview of the total memory consumption see Figure 4.10. Apparently the mean and the maximum memory consumption of the processes differ considerably. Even worse, for more than 256 processes the memory consumption increases with the number of processes. This is partially caused by the communication buffers. Further investigation with the tool massif from the tool-suite valgrind shows that the first process is always the process with the maximal memory consumption as you see comparing Figure 4.11 and Figure 4.12 (the scale of the two figures does not match). In the following we describe the problem that caused the high peak of the memory consumption:
Before reading the mesh the first process generates a background grid that grows with the number of processes. This background grid is then distributed using Parmetis. The idea is to obtain a preliminary distribution of the computational domain in order to read the mesh in parallel. As we see in Figure 4.10 the implementation does not scale well on more than $\sim$ 128 processes (so it worked well when the method was implemented for multi-core architectures).
The problem does not occur when the calculation is restarted, because the restart-files already define a preliminary distribution, so the above method is not necessary in this case (see Figure 4.13).

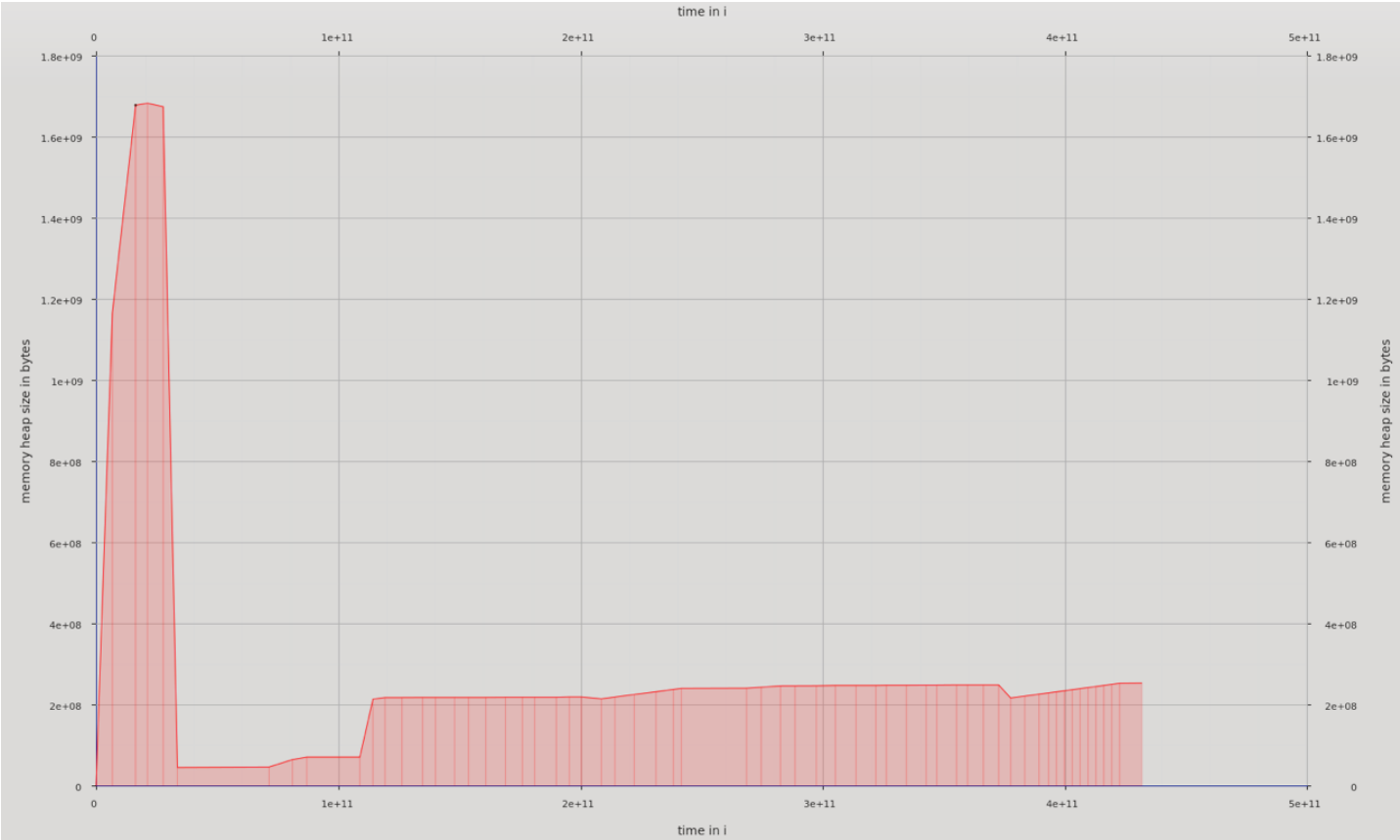


Figure 4.11.: Memory consumption of the HALO code at the startup of the calculation on process 1 of 512

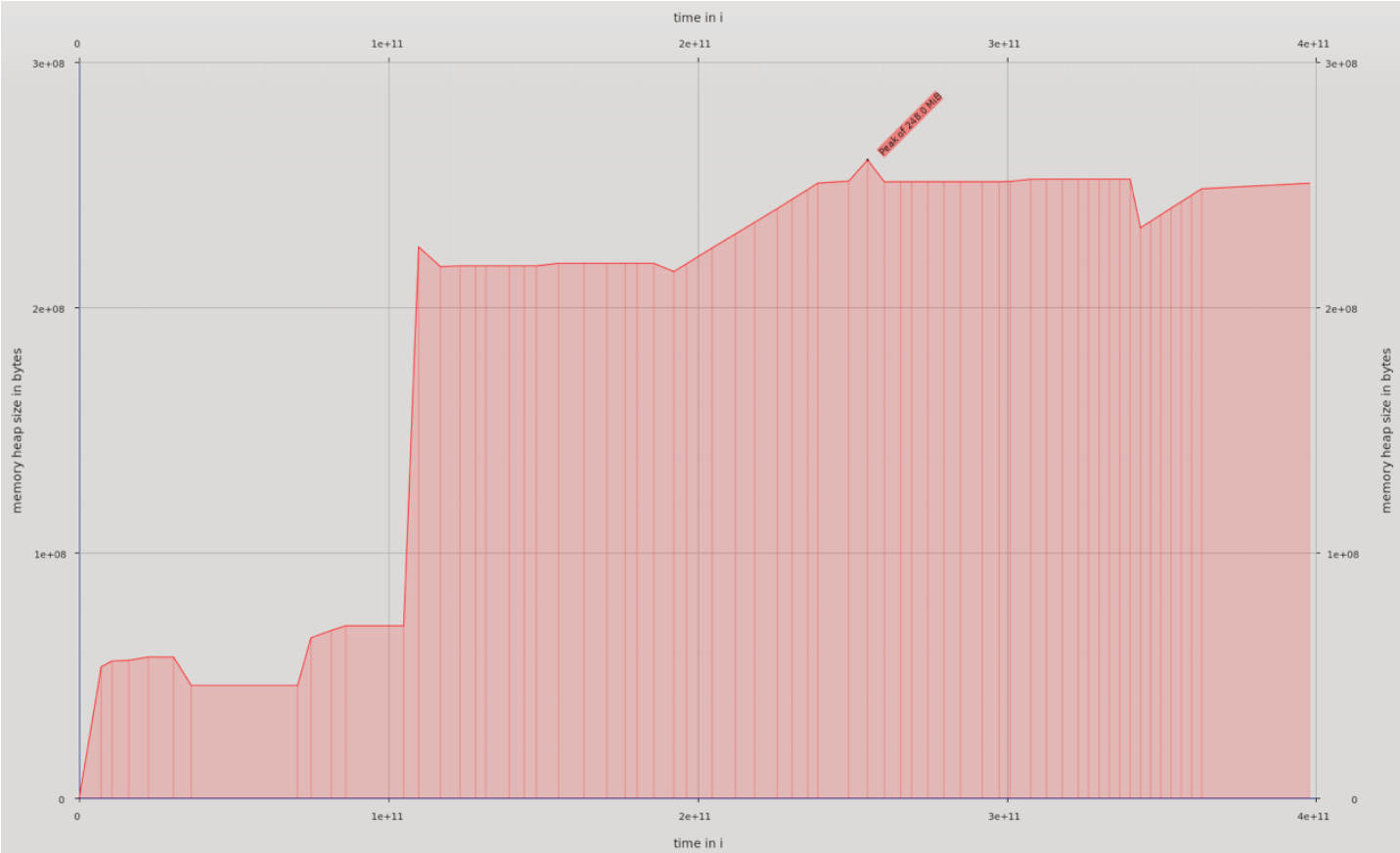


Figure 4.12.: Memory consumption of the HALO code at the startup of the calculation on processes 2 to 512

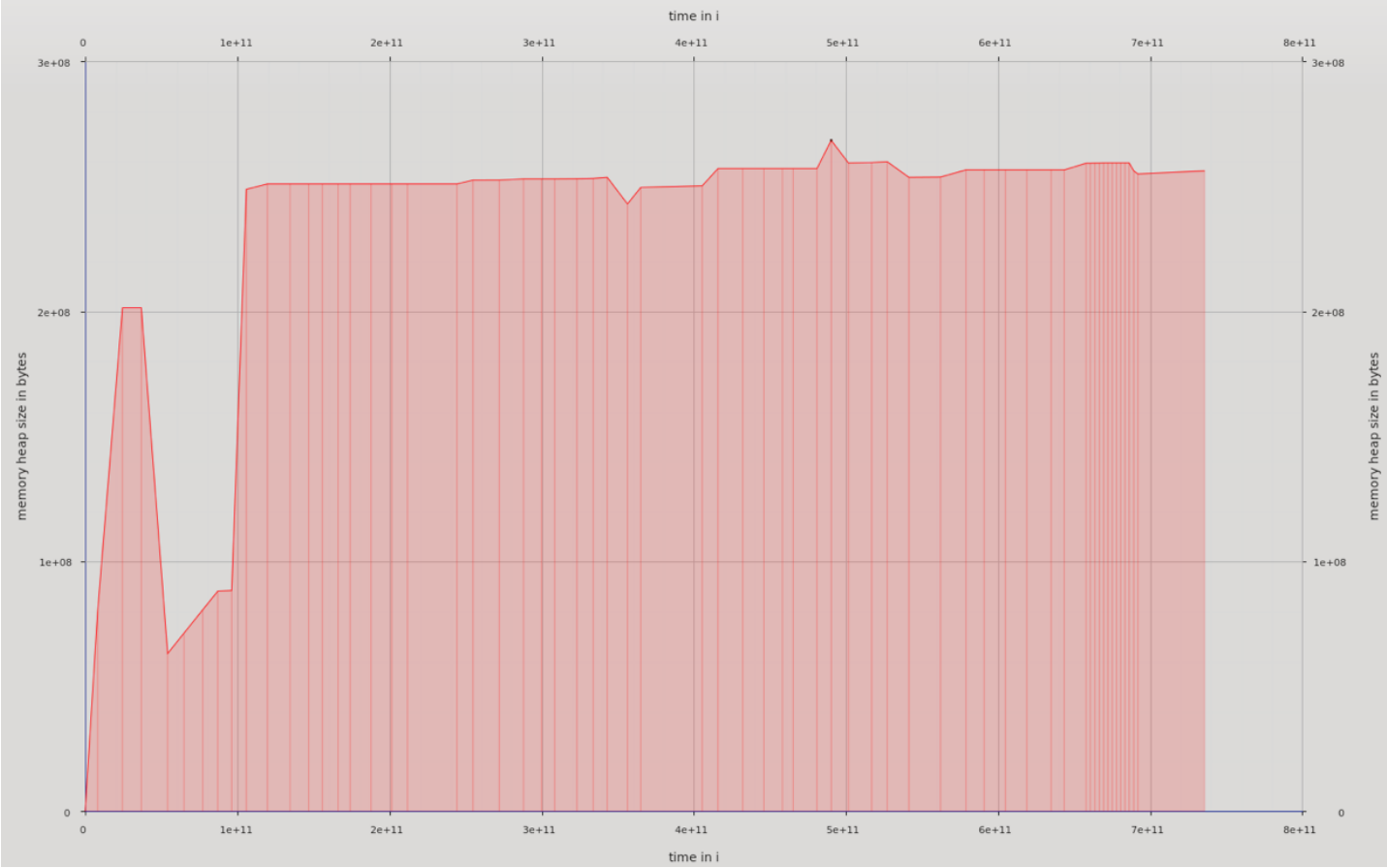


Figure 4.13.: Memory consumption of the HALO code at the startup of the calculation when restarting

See [18] for a discussion of more efficient methods to read the mesh in parallel. These do not need the previously described background grid.

4.3.5. Communication patterns

Communication and the accompanying effects can often be the root-cause for scalability issues of a code. Even more so if the communication involves not only a few but all processes. These so-called collective operations are both crucial and dangerous for efficient programming on supercomputing architectures. It is not unusual that such operations are implemented using a scheme built upon point-to-point messages resulting into an all-to-all communication. When implemented on their own these operations usually challenge the scalability of the code. The experience of the author is, that it is wise to avoid self-written collective operations and instead use an equivalent MPI_collective to ensure scalability of the written code.

Detecting self-implemented collectives and improving the communication patterns using VAMPIR

We used the tool VAMPIR to analyze the communication in the HALO code: Figure 4.14 and Figure 4.15 show visualizations of the communication over time from two complete HALO calculations. In the first figure we see the behavior of the previous implementation; the second figure illustrates the improvements due to our modifications of the HALO code. The communication matrices of these two calculations are presented in Figure 4.16.

Lets first have a closer look at the communication matrices:
There we can see that in the previous implementation every process communicates with every other process using point-to-point communication (collective MPI operations are not shown in the communication matrices). But each process exchanges only a few messages with most of the other processes. We can explain that by the fact the two processes only need to exchange data regularly during the calculation (in every timestep) when they share adjacent elements in the mesh. In other words: if we imagine that we assign a part of the computational domain to every process, only processes that work on neighboring parts should need to communicate at all. So the small number of messages that makes the communication matrix dense is created by self-implemented collective operations in the case of the HALO code. In order to identify the parts of the code that use such self-implemented collectives VAMPIR allows to adjust the time-range from which we want to extract the communication matrix.

In the following we describe the communication time-lines (Figure 4.14 and Figure 4.15):
In both pictures we see the timeline of the first 45 processes out of 128 processes. Red parts of the timeline indicate that a process is currently communicating or waiting for communication, e.g. waiting for an incoming point-to-point message or until all processes start a collective operation. Green (and white) parts represent local calculations on the specific processes. The flush-operation marked in blue should be ignored; it is merely a result of the VAMPIR profiling. In the second calculation the flush operation is not shown, because it only happened after the program terminated. Processes involved in a communication operation are connected with black lines; horizontal lines represent (unfinished) non-blocking MPI operations.

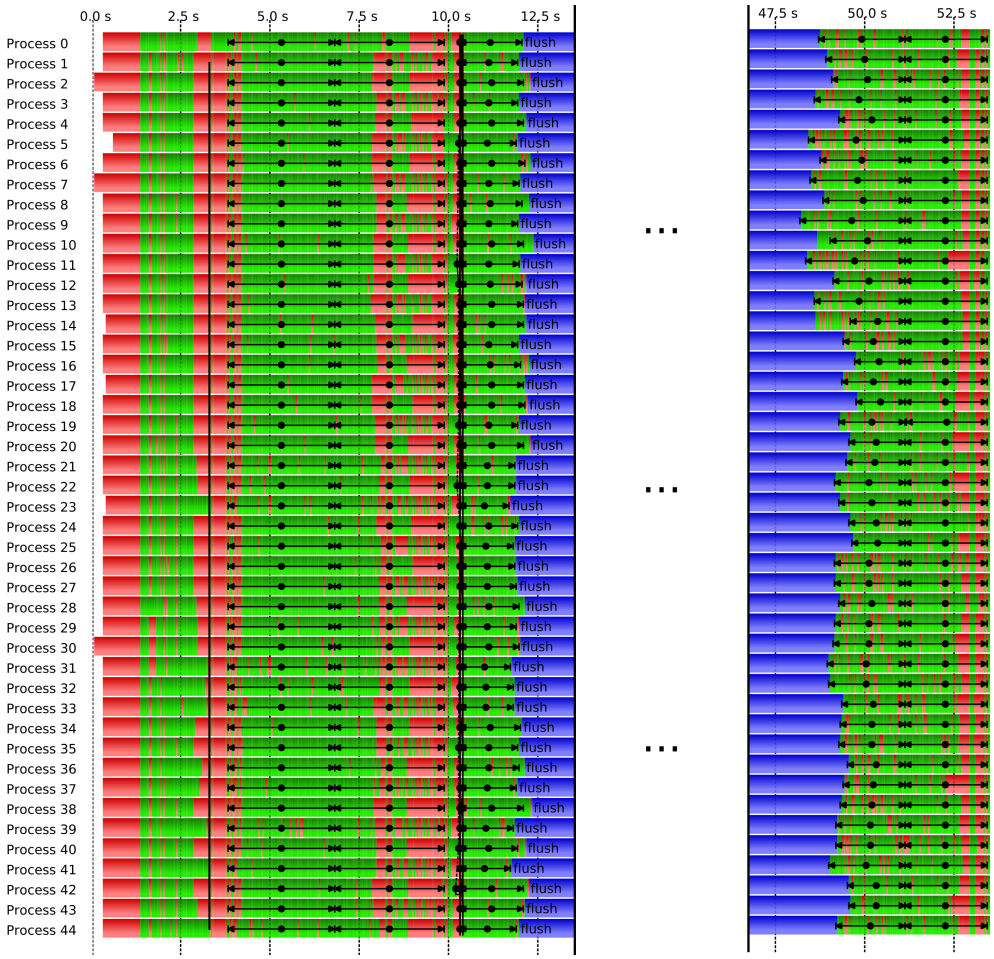
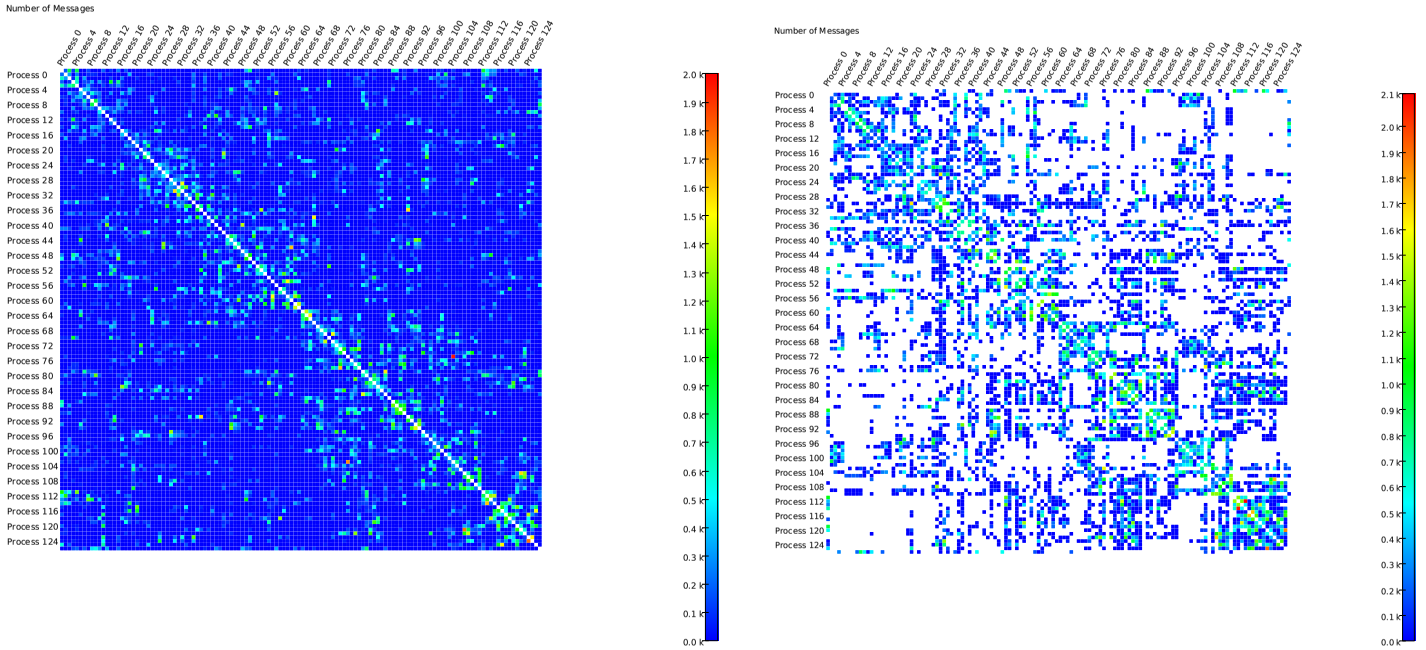


Figure 4.14.: VAMPIR communication timeline of a complete HALO calculation (without communication improvements) with 128 processes on 64 cores on the JUROPA

Figure 4.15.: VAMPIR communication timeline of a complete HALO calculation (with communication improvements) with 128 processes on 64 cores on the JUROPA



(a) unoptimized version with self-implemented collectives

(b) optimized implementation with only sparse point-to-point communication

Figure 4.16.: VAMPIR communication matrix of a complete HALO calculation with 128 processes on 64 cores on the JUROPA

To be able to compare the timeline of the first and the second calculations we can distinguish several phases:

- the startup phase, including output, (from the start to approximately $4s$, respectively $5s$)
- the first time-stepping phase ($4s \rightarrow 9s$, respectively $5s \rightarrow 9s$)
- the first synchronization, including output and load-balancing ($9s \rightarrow 10s$)
- the second time-stepping phase ($10.5s \rightarrow 52.5s$ (with the flush operation in between), respectively $10.5s \rightarrow 14.5s$)
- the second synchronization, including output and load-balancing (from $52.5s$, respectively $14.5s$, to the end)

The timeline is useful to visualize different communication patterns in different parts of the code (relating to the previously defined phases). It also hints at the overhead of the communication, as we can see when specific processes are waiting. Furthermore we can identify synchronization points, which are probably not intended.
In the case of the HALO code here we can draw the following conclusions: The overhead of the communication in the time-stepping phase seems to be quite small already; in fact the time-stepping algorithm is implemented using only sparse persistent communication (a special form of non-blocking communication defined by MPI to speed up similar operations that are used several times consecutively). This is important, because in a real-world calculation most of the time should be spent in this specific part of the code.
There are intended synchronization points between the time-stepping phases. These are necessary to write the state to disc in a given output interval. Additionally these synchronization points are used to redistribute the elements for dynamic load-balancing. As the timeline illustrates there is a considerable overhead in this specific test-case here due to the communication and synchronization. The question is if this overhead is still significant for bigger calculations when we decrease the output interval. But we can get some ideas where to look for potential problems; most of these have already been discussed in the previous sections, e.g. the record-point output and the element redistribution.

The others are described shortly in the following:
There was one case of a self-implemented MPI_EXSCAN operation using blocking synchronous point-to-point communication in order to sum up the number of elements on the processes.
In several parts of the code from the startup phase we could replace all-to-all communication by sparse communication patterns: in these cases it was necessary to exchange data about the mesh with neighboring processes, but the neighboring information was not fully setup yet or additional indirect neighbor processes were needed. There we could apply a bounding-box approach; in a first step all processes exchange the bounding box of their part of the mesh (which can be implemented using a collective MPI operation with a small message size). Afterwards only processes communicate with

each other whose bounding boxes overlap (with a given tolerance). Even if this approach may still be improvable, it effectively limits the number of different processes that need to communicate with each other.

As a result of the communication analysis and subsequently implemented optimizations there remain only sparse point-to-point communication, which we can nicely see in Figure 4.15.

MPI-Infiniband Buffer

Detailed memory analysis on distributed systems showed that the influence of communication buffers on the available memory becomes a significant factor. In particular the behavior of the MPI-buffers of socket based communication protocols as they appear in commodity clusters using Infiniband interconnect are investigated. Communication buffers are allocated at run-time for each process communicated with, and decrease the memory left for the application. The number of allocated buffers on each core depends strongly on the communication patterns and can vary from core to core. For example OpenMPI uses by default a communication buffer with the size of $512kb$ per connection for each process. These buffers are not deallocated after usage, thus a program with highly dynamical communication patterns (for example due to dynamic load-balancing) will suffer from a steady increase of buffers filling up the memory. Sophisticated MPI collectives have the potential to reduce the number of connections and with that the needed communication buffers. At the cost of run-time, memory could be reduced in the MPI internally, but the application cannot rely on it. Fig. 4.17 shows behavior of MPI_ALLTOALL with small message sizes (here: one double precision float) in comparison to large messages (one thousand double precision floats). Only for small messages a memory saving communication pattern is used by MPI internally.

The restrictions imposed by the memory requirements of the MPI library result in the need for strategies to overcome these problems. Algorithms with all-to-all or many-to-all communication patterns have to be avoided.

4.4. Performance Evaluation HALO

In this section the performance of the HALO code in its optimized form is assessed. The investigation is split into a machine-independent part in which the general performance of the code is evaluated and a machine-dependent part in which the performance of the code on several supercomputing architectures is evaluated both in serial and in parallel and are compared between those machines.

4.4.1. Algorithmic behavior

In this section the behavior of the HALO code is investigated in terms of memory and computational demand depending on the chosen order in the DG scheme. This investigation will be the ground work for discussing the performance maps of the HALO

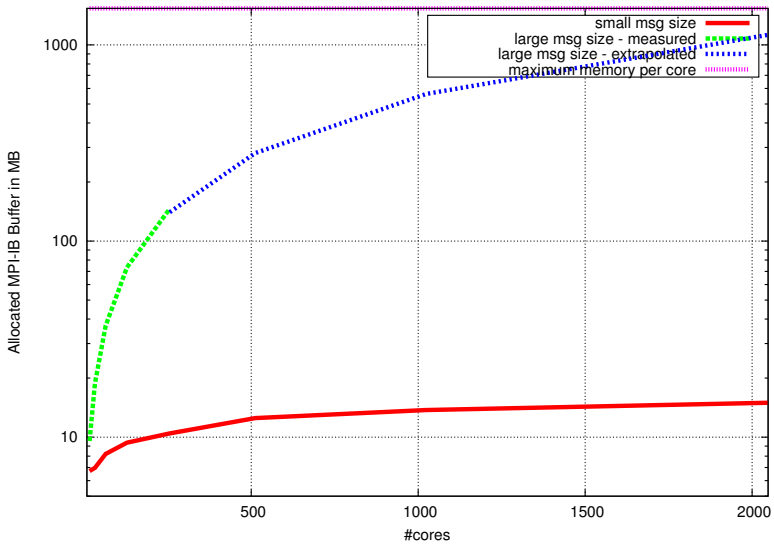


Figure 4.17.: Memory demand of MPI_ALLTOALL call depending on message and communicator size

code with respect to scalability and memory-boundness. Figure 4.18 shows both the memory demand and the computational demand depending on the spatial order of the scheme. In this case the space-time expansion is used to ensure the same accuracy in time as in space. One apparent drawback is the rising computational costs per element which is easily explained by the growing number of operations that need to be executed when using the STE in ever higher order. The number of operations per element behave like n^6, where n is the spatial scheme order. The memory per element behaves like n^3 which is to be expected for three dimensional Elements. To reduce computational costs the runge-kutta procedure is used in the ck-procedure instead of the STE, leading to a decoupling of spatial and temporal order in the scheme. In this case a 3rd order accurate Runge-Kutta scheme is used for all investigated spatial orders. It can be seen that the computational demand is much more moderate compared to the results obtained using STE. The number of operations per element behave like n^5, where n is the spatial scheme order. The memory per element behaves like n^3 and therefore the same as with STE. In general the number of operations per element are very high even in second order with STE, which resembles the least possible operations. This high computational costs per element will reflect in the parallel scaling investigations in the next section. Due to the higher coefficient in the computational effort with respect to the memory some estimations can be done on how the code will behave in general. Due to the high serial computational cost a good strong scaling is to be expected as with a high computation effort the increasing communication effort takes

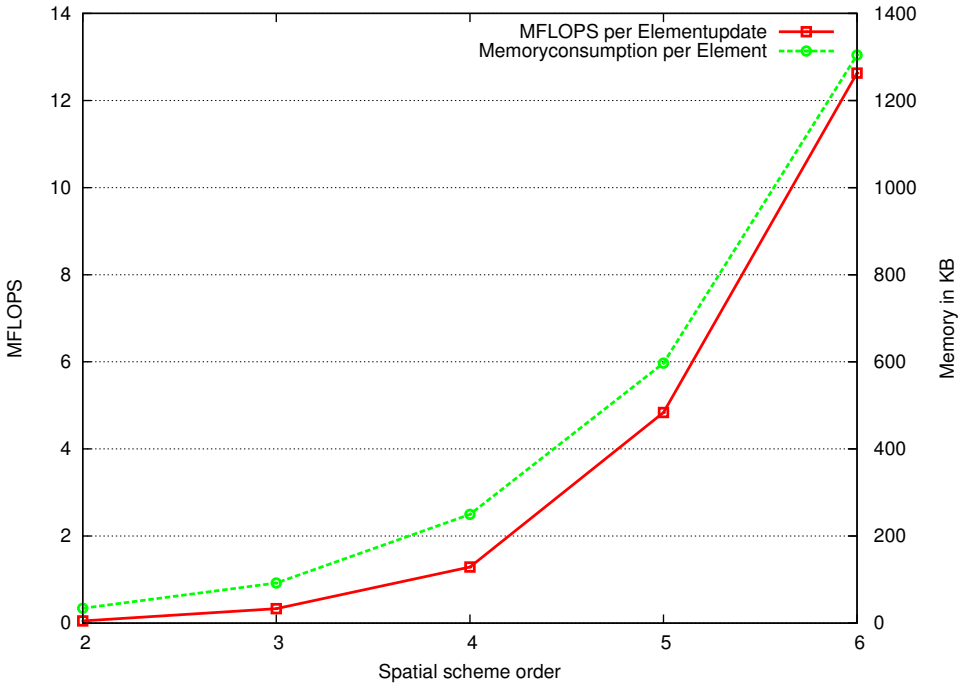


Figure 4.18.: Computational and memory demand of HALO code depending on the used spatial scheme order. Employing STE the order of accuracy in space and time are the same.

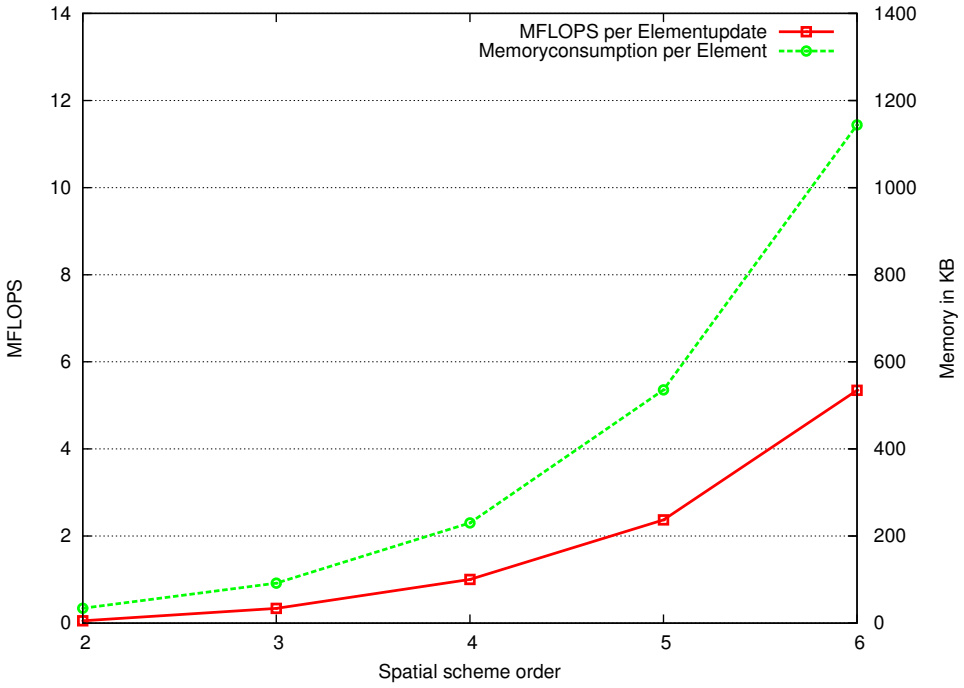


Figure 4.19.: Computational and memory demand of HALO code depending on the used spatial scheme order. Employing RKCK the order of accuracy in space and time are the decoupled. The time order is fixed to 3.

name	processor type	architecture	clock	memory
Laki	Intel X5560	Nehalem	2.8	1333 MHz
Juropa	Intel X5565	Nehalem	2.93	1066 MHz
RZ-Aachen	Intel X5675	Westmere	3.06	1333 MHz
Laki(SB)	Intel E5-2670	Sandy Bridge	2.4	1600 MHz
Hermit	AMD Opteron 6276	Interlagos	2.3	1333 MHz

Table 4.3.: Node description of the investigated architectures

longer to affect the scalability. Additionally with increasing spatial order the scheme should scale even better, in the sense that less elements per node are still efficient.

4.4.2. Serial Performance

In this section the serial performance on different architectures for the HALO code is discussed. Table 4.4.2 gives an overview over the architectures investigated.

Performance Influence of Compilers While investigating the available programming environments (Intel, GNU and PGI [1]) on hermit, large discrepancies where found regarding the run-time of the different executables produced by the different compilersFigure 4.20 shows the comparison of the serial performance for the different compilers. It can be seen that for the HALO code the Intel compiler produces the fastest executable. GNU and PGI compiled executables only reach 68.5% and 48.7% respectively of the performance of the Intel executable.

Comparison of Serial Performance between different Architectures Execution times of HALO on Laki and Hermit for Intel and PGI compilers are presented in Table 4.4. On Laki the single core run time is up to 42% less than on Hermit.

Examples	Cray XE6		NEC Nehalem		Run-time reduction	
	Intel -O2 -g	PGI -fast	Intel -O2 -g	PGI -fast	Intel -O2 -g	PGI -fast
LES disabled	104,35 s	256,74 s	68,17 s	160,41 s	34,7%	37,5%
LES enabled	274,99 s	749,60 s	191,26 s	429,89 s	30,4%	42,7%

Table 4.4.: Single core run-time on Cray XE6 and NEC Nehalem Cluster

4.4.3. Intra-node Performance

In this section intra-node performance of the HALO code are evaluated using the performance metrics introduced in section 4.1.

[1] The cray compiler could not produce an executable

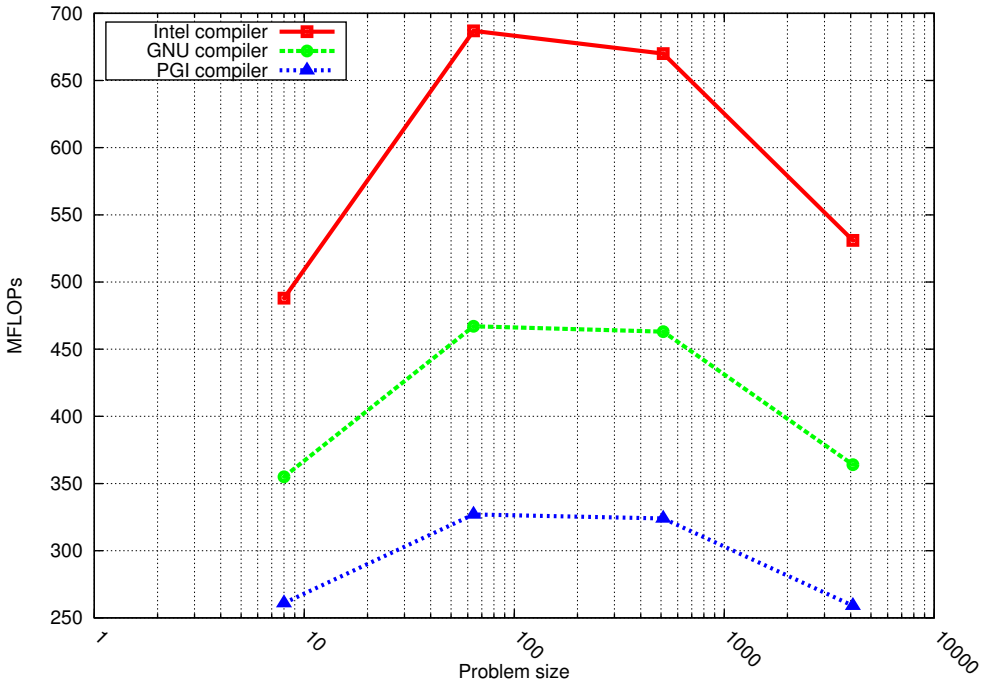


Figure 4.20.: Serial performance of HALO, compiled with different compilers (Intel, GNU, PGI)

name	processor type	architecture	sockets per node	cores per node	memory freq.
Laki	Intel X5560	Nehalem	2	8	1333 MHz
Juropa	Intel X5565	Nehalem	2	8	1066 MHz
RZ-Aachen	Intel X5675	Westmere	2	12	1333 MHz
Laki(SB)	Intel E5-2670	Sandy Bridge	2	16	1600 MHz
Hermit	AMD Opteron 6276	Interlagos	2.3	1333 MHz	

Table 4.5.: Node description of the investigated Architectures

Current Intel Architectures

The yearly release of new processor architectures leads to different node setups between the different supercomputers. With the processor development being a evolutionary process this section is supposed to give an impression on how the performance of the HALO code is influenced by the current Intel architectures. Table 4.4.3 gives an overview over the different architectures investigated. In figure 4.21 the intra-node performance map for the HALO code using a 2nd order DG scheme in space and a third order RK in time on the Laki system is depicted. A rather typical intra-node behavior can be seen, with the cache effect not being very distinctive but present. With increasing number of cores the cache effect moves to higher problem-sizes, displaying a expected behavior. Furthermore the performance increases with each additional core used up to the full node, although the Nehalem architecture provides only three memory lanes per CPU and therefore a drop in speedup was expected from 4 to 8 cores. This allows the conclusion that the HALO code is not memory bound. The best setup for large problem sizes consequently is the usage of all 8 cores of a node. In Figure 4.22 an 4.23 the intra-node performance maps for the same system are shown. With increasing scheme order the reached FLOPs per problem size rises (see also 4.4.1) as the processor can more efficiently work on one large chunk of data continuously (higher order leads to more FLOP per element) than on smaller chunks one after another (data needs to be rearranged more often - caching not as efficient). Higher scheme orders therefore enable a more efficient usage of the node in an environmental sense. With 6th order almost 15 GFLOPs are reached, which represents 16.7% of the theoretical peak-performance [2] for the investigated node. This is a respectable value on this kind of architectures. It also can be seen that the minimum number of elements needed to use a full node efficiently is independent from the chosen spatial order. This show that the overhead of the program does not increase with the order

[2] Ideal Floating-Point Throughput For the Xeon 5560 which operates at 2.8GHz, we can say that in the steady state and under ideal conditions each core can retire 4 double-precision or 8 single-precision floating-point operations each cycle. Therefore, the nominal, ideal throughput of a Nehalem core, a quad core and a 2-socket system are, respectively, 11.2 Giga FLOPs / sec / core= 2.8GHz X 4 FLOPs / Hz 44.8 Giga FLOPs / sec /socket= 11.2GigaFLOPs/sec / core X 4 cores 89.6 Giga FLOPs / sec / node= 44.8GigaFLOPs/sec / socket X 2 sockets

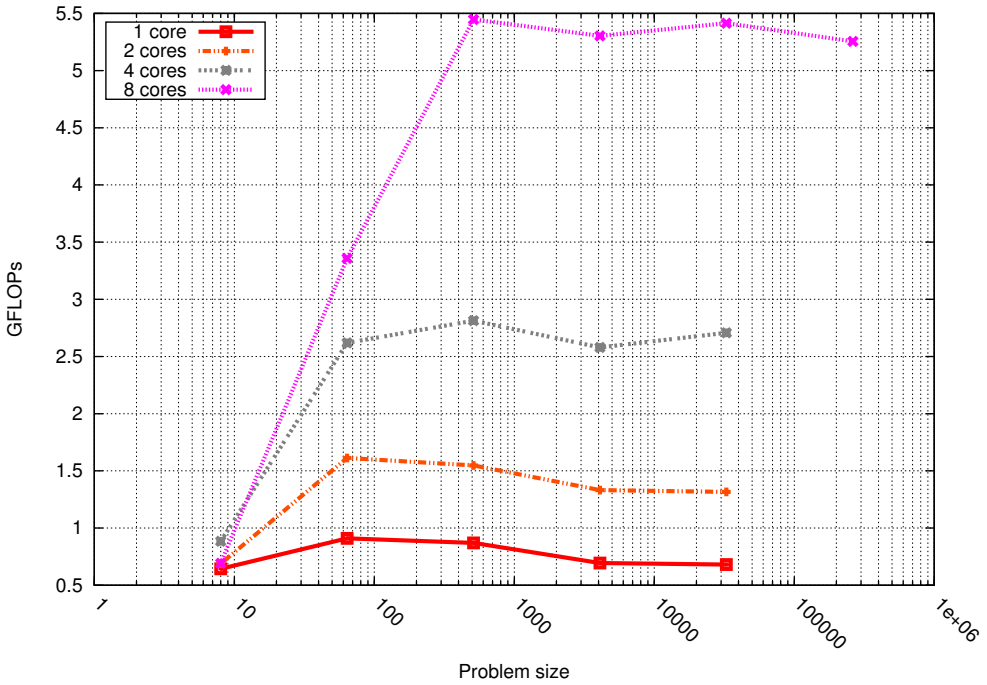


Figure 4.21.: Intra-node performance for the HALO code on a Laki node with two Intel Nehalem X5560 processors. HALO code was compiled using OpenMPI 1.5.4 and the Intel compiler 12.0.4. Spatial order:2, temporal order:3.

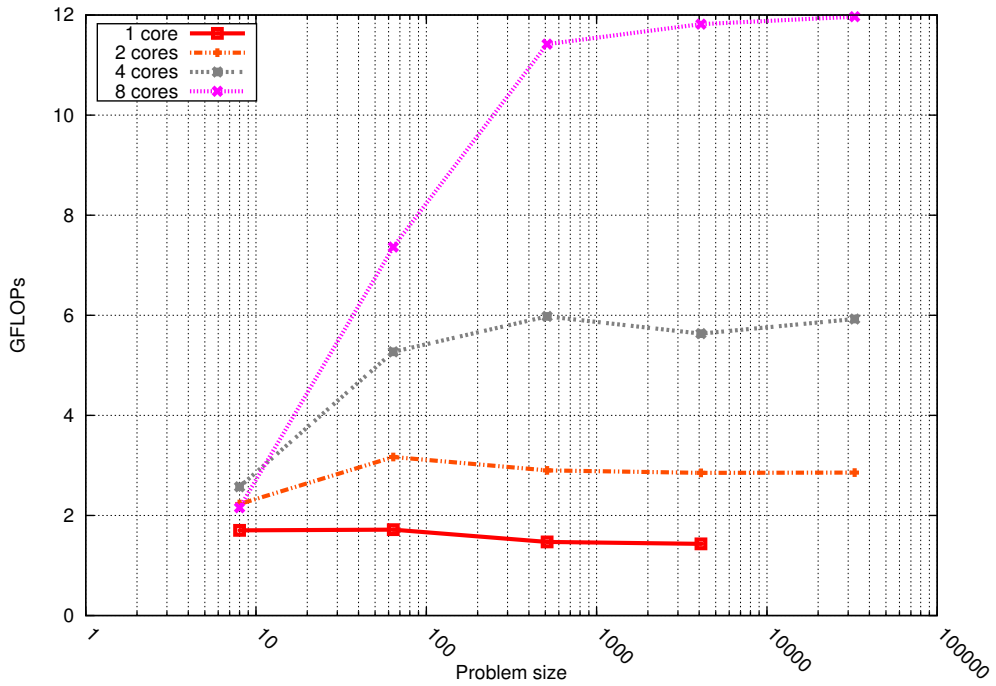


Figure 4.22.: Intra-node performance for the HALO code on a Laki node with two Intel Nehalem X5560 processors. HALO code was compiled using OpenMPI 1.5.4 and the Intel compiler 12.0.4. Spatial order:4, temporal order:3.

of the scheme. Due to the fact that the performance behavior across the investigated Intel based architectures are quite similar not every architecture is presented in full detail in this section. To be able to judge the differences between the different Intel based architectures figure 4.24 shows the 'per-core" performance for an efficiently used full node on the respective architecture. It can be seen that the performance per core does not increase significantly over the last iterations of Intels architectures.

Please refer to A.1 for detailed performance maps for all investigated architectures and orders.

AMD Interlagos Architecture

One significant feature of this processor-architecture is that always two cores share one floating point unit (FPU) potentially hindering the scaling to all 32 cores on a node. Figure 4.25 shows the intra-node performance for the HALO code on a hermit node. Despite the restrictions with the FPUs the code scales very well for large problem sizes on the node. The usage of all cores on a node gives the best performance.

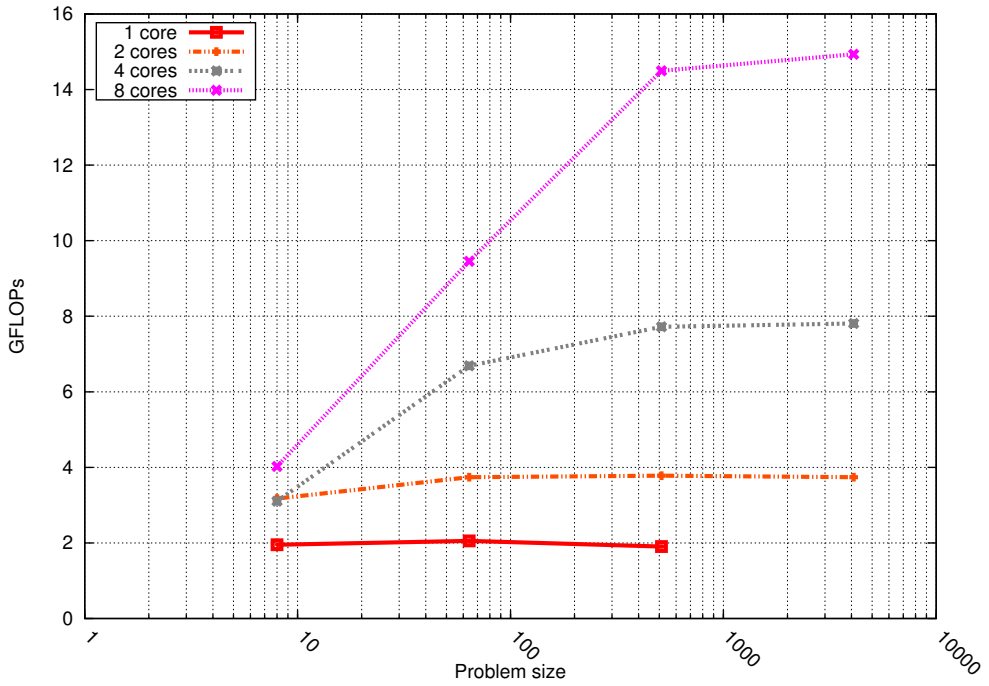


Figure 4.23.: Intra-node performance for the HALO code on a Laki node with two Intel Nehalem X5560 processors. HALO code was compiled using OpenMPI 1.5.4 and the Intel compiler 12.0.4. Spatial order:6, temporal order:3.

Figure 4.24.: Comparison between current Intel based architectures for the HALO code.

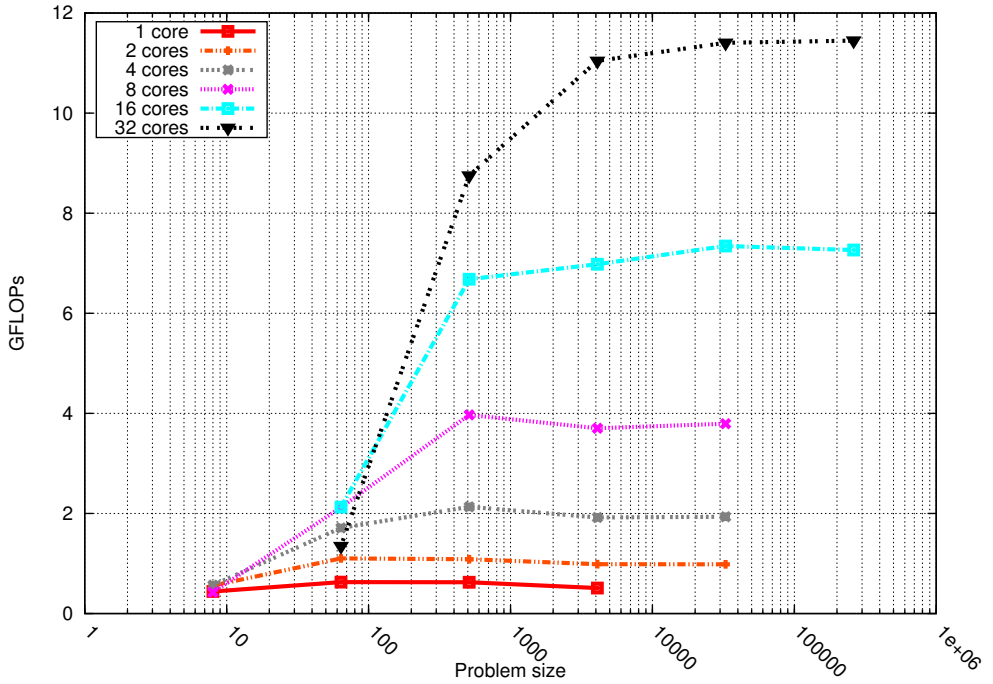


Figure 4.25.: Intra-node performance for the HALO code on a Hermit node with four AMD Opteron 6276 processors. HALO code was compiled using the Intel programming environment. Spatial order:2, temporal order:3.

With 12 GigaFlops the code achieves 4% of the theoretical peak performance of 294.4 GigaFlops. For smaller problems - especially for 64 elements - it can be seen that the overhead for spanning the simulation over 32 cores is reducing the overall performance. In the figures 4.26 and 4.26 the behavior for 4th and 6th spatial order is depicted. On the AMD architecture the behavior of the higher orders are very similar to the second order behavior.

Comparisons

Influence of memory bandwidth on total performance When comparing the performance results on the two Nehalem based machines Laki and Juropa (refer to figure A.2 and A.3) it can be seen that the Laki node offers higher performance on all three investigated orders despite being the node with the lower CPU clock frequency (refer to table 4.4.3). The explanation for this reverse behavior lies in the memory bandwidth. Whereas Juropa uses the 1066 MHz clocked memory Laki is using a 1333 MHz clock. Therefore the node can be used more efficient as the data gets faster from the memory

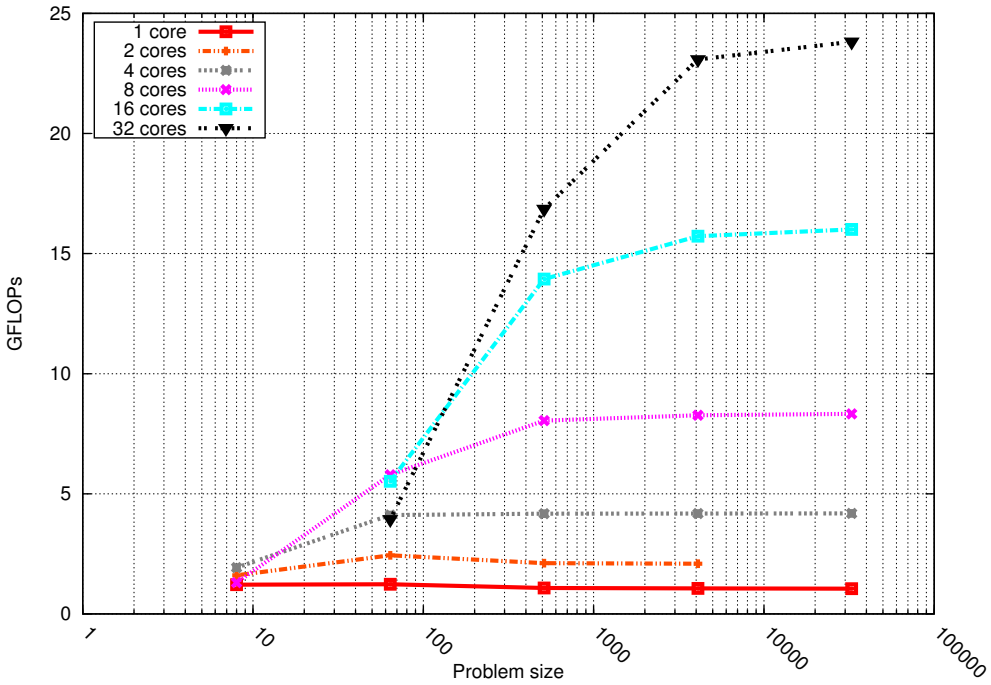


Figure 4.26.: Intra-node performance for the HALO code on a Hermit node with four AMD Opteron 6276 processors. HALO code was compiled using the Intel programming environment. Spatial order:4, temporal order:3.

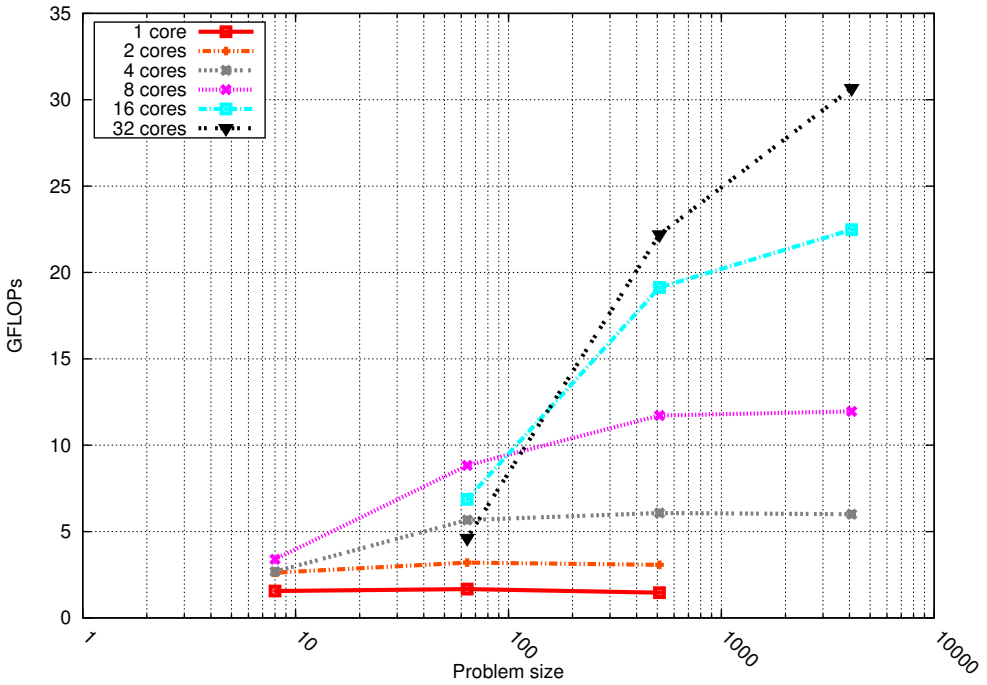


Figure 4.27.: Intra-node performance for the HALO code on a Hermit node with four AMD Opteron 6276 processors. HALO code was compiled using the Intel programming environment. Spatial order:6, temporal order:3.

to the core. With increasing order this effect should decrease as the scheme gets more and more memory unbound due to increasing work packages in form of operations per element. This can be seen when comparing the performance plus between the different orders.

Comparison between Intel and AMD architecture Figure 4.28 shows the comparison between a Hermit node using all available 32 cores and a Laki node using all available 8 cores. It can be seen, that a Hermit node achieves more than twice the performance of a Laki node - however the relation of the performance per core is exactly the reverse.

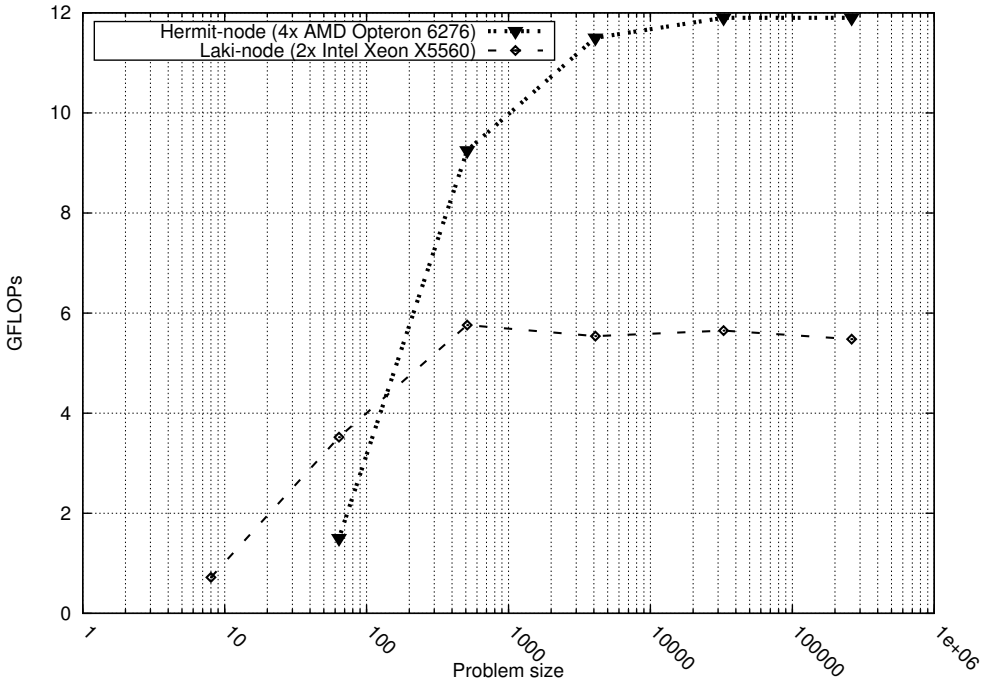


Figure 4.28.: Comparison of node performance of the HALO on Hermit and Laki

4.4.4. Inter-node Performance

The inter-node performance of a code can be strongly dependent on the network of the used architecture.

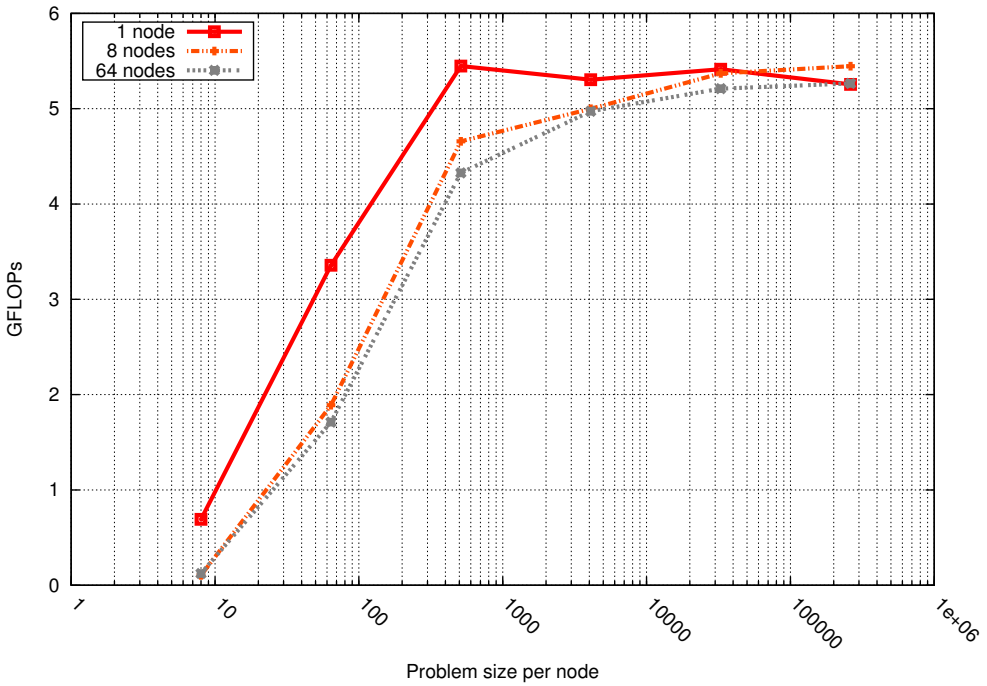


Figure 4.29.: Intra-node performance for the HALO code on a Laki node with two Intel Nehalem X5560 processors. HALO code was compiled using OpenMPI 1.5.4 and the Intel compiler 12.0.4. Spatial order:2, temporal order:3.

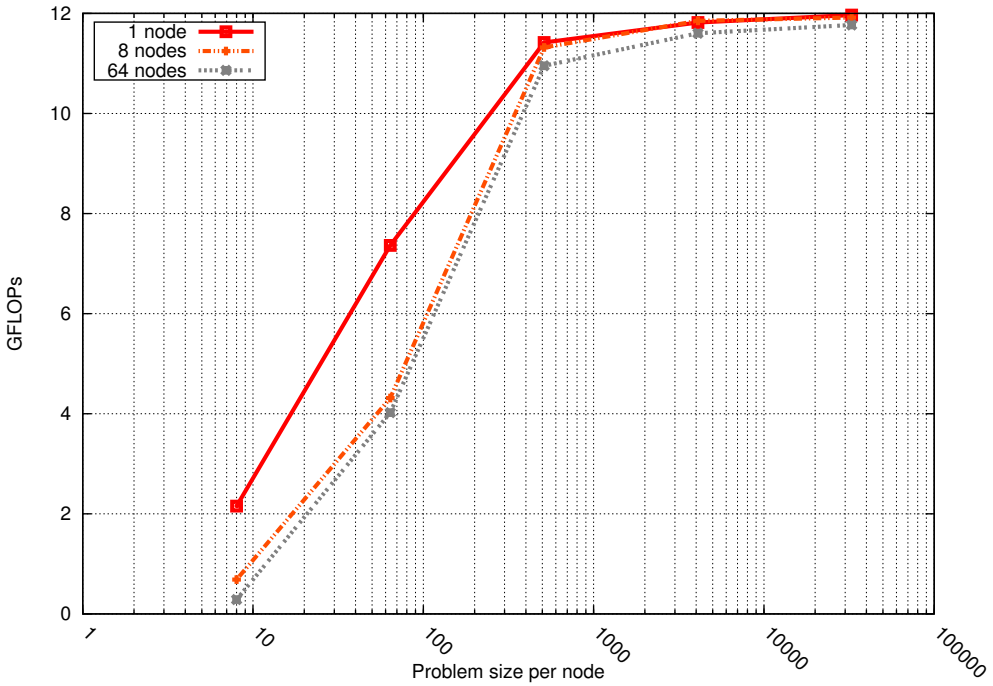


Figure 4.30.: Intra-node performance for the HALO code on a Laki node with two Intel Nehalem X5560 processors. HALO code was compiled using OpenMPI 1.5.4 and the Intel compiler 12.0.4. Spatial order:4, temporal order:3.

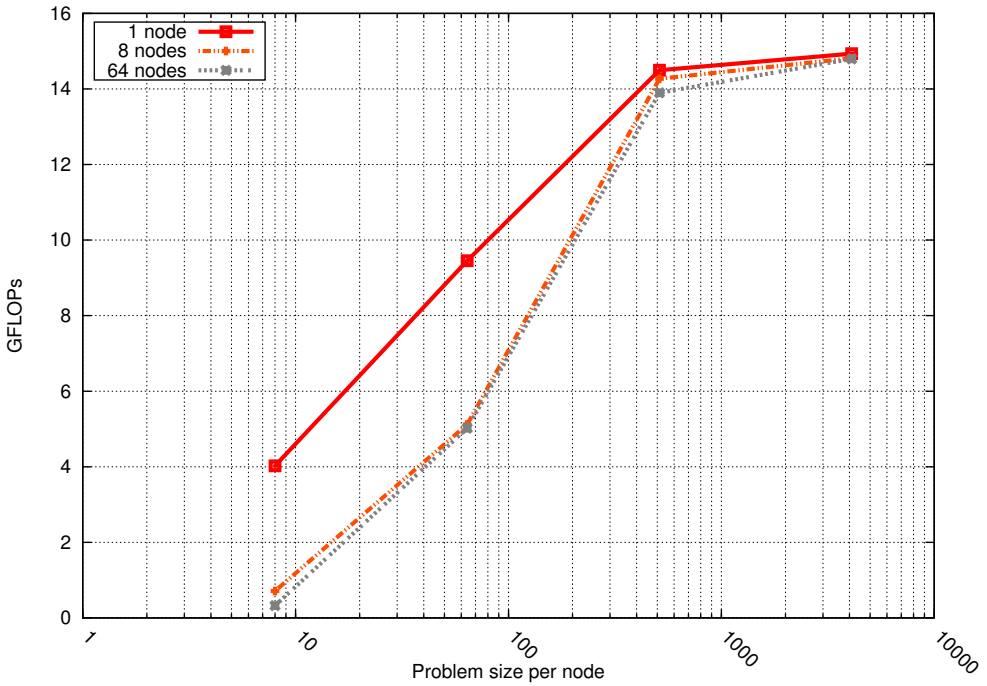


Figure 4.31.: Intra-node performance for the HALO code on a Laki node with two Intel Nehalem X5560 processors. HALO code was compiled using OpenMPI 1.5.4 and the Intel compiler 12.0.4. Spatial order:6, temporal order:3.

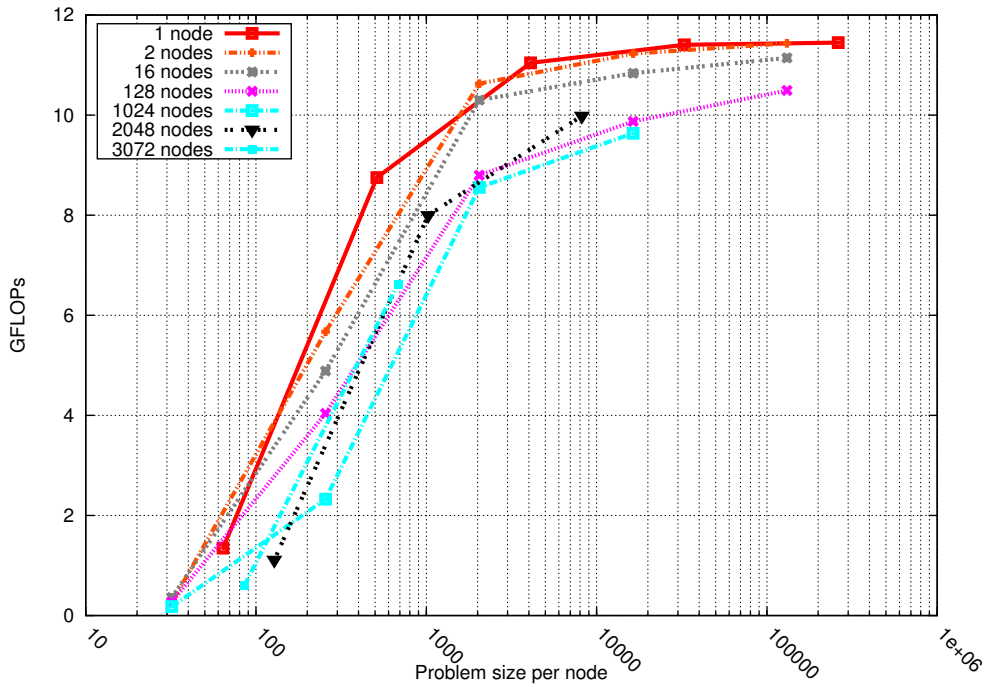


Figure 4.32.: Intra-node performance for the HALO code on a Hermit. HALO code was compiled using the Intel programming environment. Spatial order:2, temporal order:3.

Intel-Infiniband based Architecture

AMD-Gemini based Architecture

In this section we investigate the inter-node behavior of the HALO code on the Hermit system. In the figures 4.33 and 4.33 the behavior for 4th and 6th spatial order is depicted. On the AMD architecture the behavior of the higher orders are very similar to the second order behavior.

Comparisons

Figure 4.35 shows the scaling behavior for the largest mesh fitting onto a single Hermit node (262,144 elements) on Hermit and Laki. The results show that the scaling on Laki for up to 512 cores is identical to the behavior on Hermit, but resulting in higher overall performance due to the stronger per-core performance on Laki. To achieve the same performance and therefore similar run time for this test case on Hermit around twice as many cores are required. This corresponds well with the data from the serial

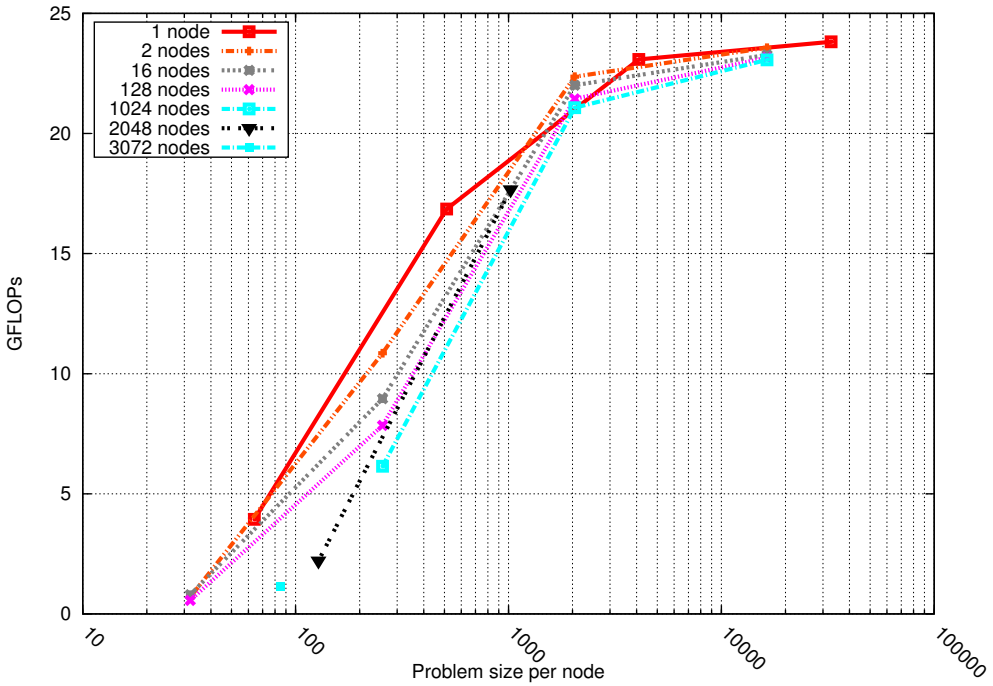


Figure 4.33.: Intra-node performance for the HALO code on a Hermit. HALO code was compiled using the Intel programming environment. Spatial order:4, temporal order:3.

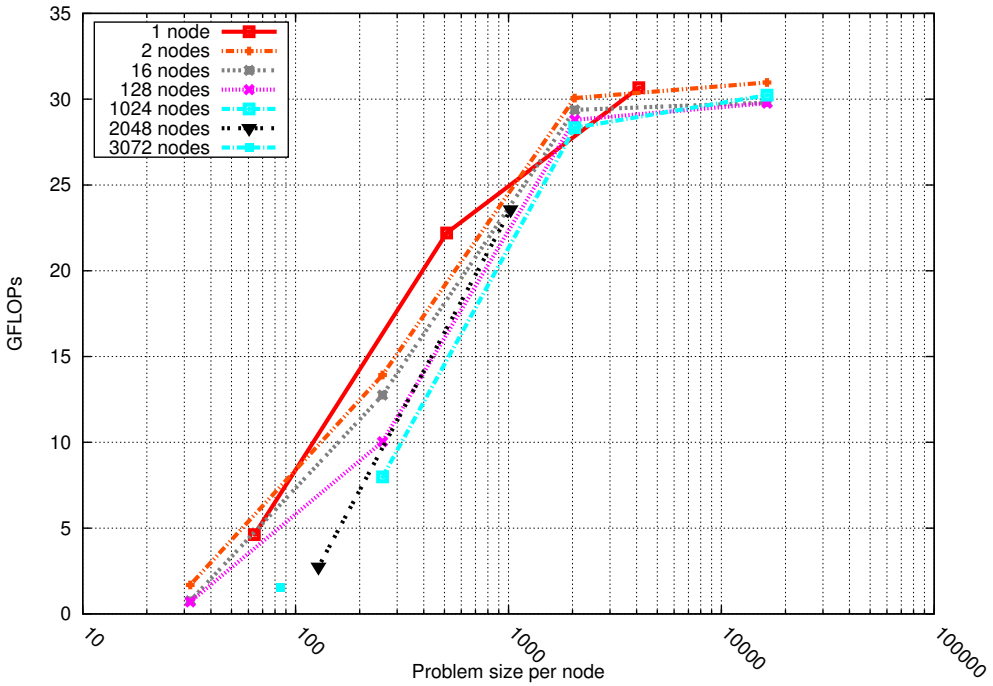


Figure 4.34.: Intra-node performance for the HALO code on a Hermit. HALO code was compiled using the Intel programming environment. Spatial order:6, temporal order:3.

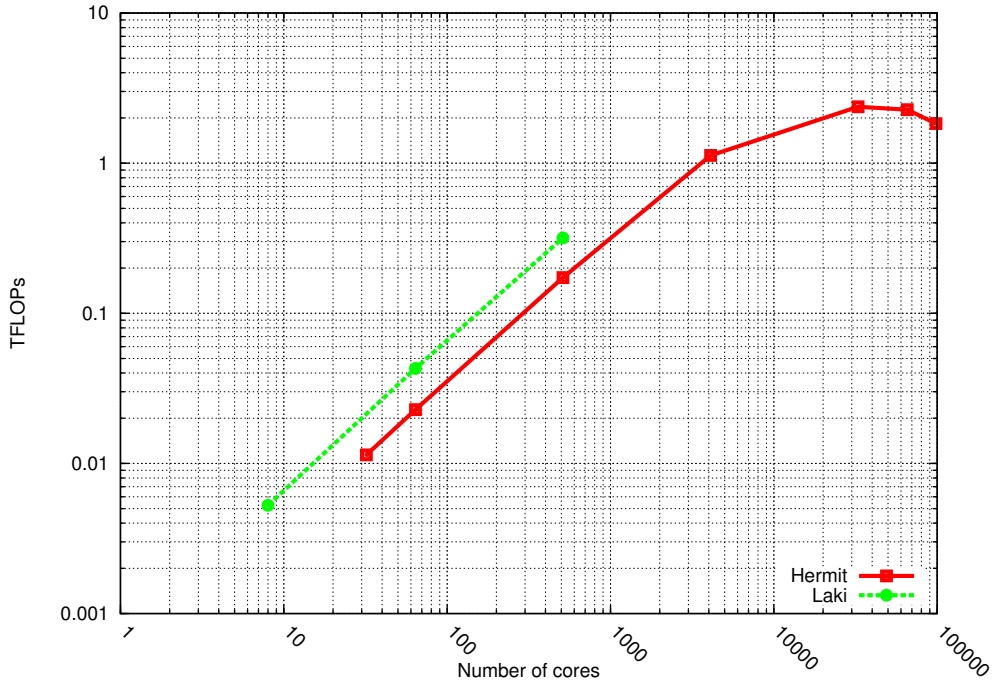


Figure 4.35.: Performance HALO for small problem sizes under strong scaling on Hermit and Laki.

investigations. Additionally the figure contains the information for the maximum speedup that is effectively possible. A total speedup of 100 can be achieved for a problem size, that just fits into main memory, before the scaling degrades.

4.5. Performance Analysis of AIA codes

For each cluster, the computation time of 3 equal computations was averaged to obtain the results. The basic cluster information is given in the table below:

Cluster	# Cores	Memory/ Core	CPU	Cores/ CPU	Speed
Nehalem Stuttgart	5600	1.5GB	Intel Xeon X5560	4	2.8 GHz
Juropa Jülich	17664	3.0GB	Intel Xeon X5570	4	2.93 GHz
Cray XE6 Stuttgart	1344	2.0GB	AMD Opteron	8	2.3 GHz
AIA Poweregde Cluster	136	2.0GB	Intel Xeon X5450	12	3.0 GHz
RWTH Bull Cluster	1350	2.0GB	Intel Xeon X5675	12	3.06 GHz

4.5.1. Code scalability

Speed-up measurements have been performed for different problem sizes on the BlueGene System JUGENE at the von Neumann Computing Center (NIC), Forschungszentrum Jülich, and on the CRAY HERMIT Cluster at HLRS Stuttgart with a 3D Lattice-Boltzmann flow solver with a D3Q19 LBGK solution method. The BlueGene System consists of 73728 nodes of type Power PC540 with 850MHz clocking with 2GB RAM each. The CRAY HERMIT Cluster has 3552 nodes, each with two sockets equipped with a 16 core AMD Interlago CPUs at a clocking of 2.3GHz and with a memory of 32GBs or 64GBs RAM per node. The scaling experiments were conducted on four different core numbers on the JUGENE, i.e. 1024, 2048, 4096, and 8192, and on five different core numbers on the HERMIT System, i.e. 512, 1024, 2048, 4096, and 8192. Domain decompositioning on a cubic domain has been performed with a Hilbert decompositioning method using space filling curves. This decompositioning method guarantees equal cell numbers per domain for the given geometry. The measurements cover the mean execution time of the inner iteration loop for three simulations of the same problem. Strong scaling experiments based on a 3D flow simulation in a cubic geometry with the in-house flow solver ZFS have been performed on both systems for a constant cell number of 0.403×10^9 cells and 0.537×10^9 cells on HERMIT and JUGENE, respectively.

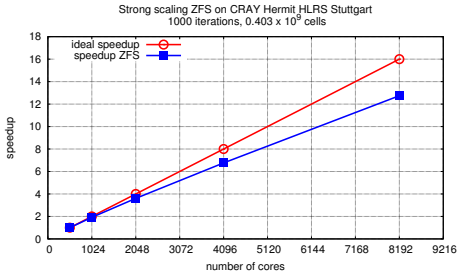


Figure 4.36.: Strong scaling on HERMIT

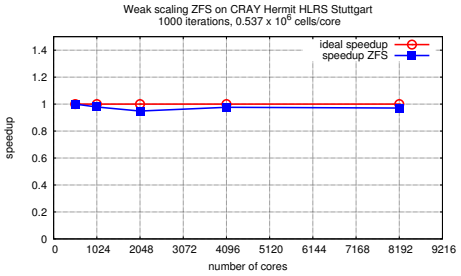


Figure 4.37.: Weak scaling on HERMIT

Fig. 4.36 shows the speedup on HERMIT and the according ideal speedup based on the reference time for 512 cores. The speedup for 1024 cores shows a perfect scaling while a continuous slight decrease appears for core numbers 2048 to 8192. This decrease is caused by the reduction of the number of cells per core and hence by the reduction of the computational time and a resulting higher weighting of the communication time. In contrast the speedup results obtained on the BluGene System depicted in Fig. 4.38 show a slightly higher drop of the scale-up from 4096 to 8192 cores. The weak scaling experiment uses a fixed local size of 0.537×10^6 cells per core on HERMIT and 0.262×10^9 cells per core on JUGENE. Fig.. 4.37 and Fig. 4.39 show the speedup for both systems based on the reference time for 512 and 1024 cores for HERMIT and JUGENE, respectively. In both cases the results show almost a perfect scaling for a

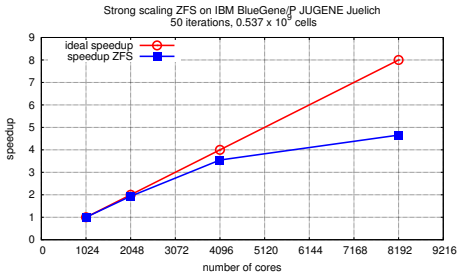


Figure 4.38.: Strong scaling on JUGENE

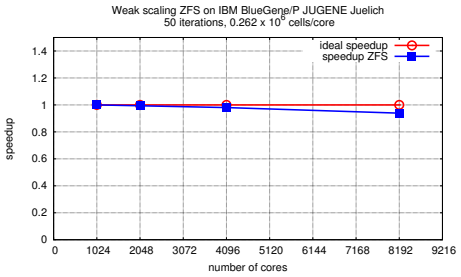


Figure 4.39.: Weak scaling on JUGENE

constant problem size per core under an increase of the number of cores. The total number of cells simulated on HERMIT for 8192 cores is 4.4×10^9 cells and 2.14×10^9 cells on JUGENE.

4.5.2. Benchmarking

Single core and single node performances measurements have been conducted on the AIA RWTH Dell Cluster, the RWTH Bull Cluster, HLRS NEC Nehalem Cluster and on the HLRS CRAY HERMIT System with different compilers and compiler options. The configuration of the different machines is listed in Tab. 4.5 For the single core performance measurement a total number of 8^5 cells was used for the simulation with a 3D Lattice-Boltzmann D3Q19 LBGK flow solver. As a reference, the simulation time on the NEC Nehalem System for ZFS compiled with the GNU compiler and the options "-O3 -fno-tree-vectorize -funroll-loops", which yielded the fastest execution time for 10^3 iteration steps was used. The testbed consisted of compiled versions of ZFS with the GNU compiler, the Intel compiler and additionally on the HERMIT System of the CRAY compiler. The compiler options given in Fig. 4.40 were used, each yielding the optimal results for the different compilers. Additionally, the standard compiler options of the CRAY compiler were used on HERMIT. The shortest execution time on all systems was obtained by using the GNU compiler with the given compiler options. For this compiler, almost similar execution times were obtained on the AIA RWTH Dell Cluster, the RWTH Bull Cluster and the HLRS NEC Nehalem Cluster. The speed on the CRAY system was about 75% the speed of the optimal speed on the HLRS NEC Nehalem Cluster. For all four systems the performance of ZFS compiled with the Intel compiler was in the range of 50-60% of the according execution times for the GNU compiler. The use of additional compiler options for the CRAY compiler resulted only in a slight decrease of the execution time. The single node performance measurements were performed with the GNU compiler with the optimal compiler options. A constant cell number of 8^5 cells per core was used on all systems. As shown in Fig. 4.41, measurements were performed for the computational execution time, the time for filling the communication buffers, and the time for the MPI-Communication

for 10^3 iterations. The fastest overall execution time was obtained on the the HLRS CRAY Cluster under a usage of every second core. Although the computational time for the calculation of the collision kernel for the LBGK is higher than the one obtained on the HLRS NEC Nehalem Cluster, the inner node communication is much faster. The overall execution time on 32 cores on HERMIT is due to the concurrent access demand on the FPUs, which are each shared by two cores, much higher than the execution time on only 16 cores. Interestingly, the computational time on the RWTH AIA Dell and the RWTH Bull Clusters are in contrast to the single core performance measurements lower than the execution time on the CRAY Cluster.

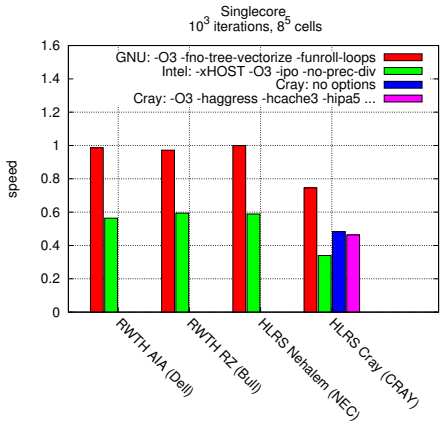


Figure 4.40.: Single core performance

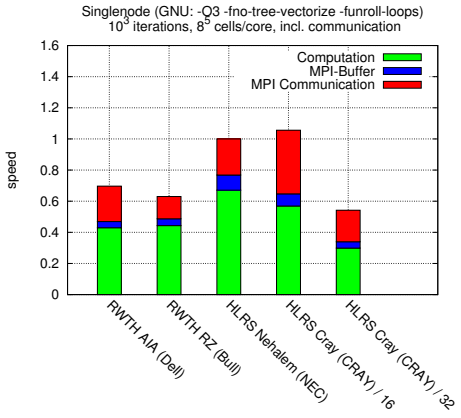


Figure 4.41.: Single node performance

A structured multi-block solver has been implemented to solve the acoustic perturbation equations (APE-4) [6] for aeroacoustic research. The equations are discretized using a sixth-order dispersion-relation preserving summation by parts scheme [17] along with a low-dissipation low-dispersion Runge-Kutta time-integration method [15]. All equations are expressed using very compact m4-macros that are preprocessed into a Fortran code that is highly optimisable, inlined, cache-optimized and parallelized hybridly. Thus only very few code lines are required to obtain a performant PDE solver. Fig 4.42 compares the code's performance on different architectures, including a Cray XE-6, a Bullx B500, and a IBM BlueGene/P system. Furthermore the performance of builds from several compilers has been benchmarked for the Cray system. For all test runs 100 time steps have been performed on a 256^3 grid with periodic boundary conditions using 64 MPI processes and 1 SMP thread per MPI process. Cray's Fortran compiler showed the best performance on XE-6. The idle times on this machine are moderate. Due to the lack of work that can be overlapped with the communication in a performant way, a small amount of idle time can hardly be avoided. Due to the von Neumann memory bottleneck the workload related to exchange is relatively large, which is caused by the large amount of data to be transferred via separate buffers. The computational performance on the Bullx B500 is much higher caused by the dedicated

floating point units available to each computational core. However, the high idle time indicates a insufficient network performance or inappropriate process mapping to the machine's layout. The computational performance on IBM BlueGene/P is more than six times worse when compared to the Cray XE-6, but the IBM system is intended to use a much larger number of cores for computations. The workload related to exchange is relatively low due to the good computational core to memory clock ratio. The network performance is superior to the other machines. The strong scaling for the Cray XE-6 and IBM BlueGene/P system in fig. 4.43 emphasizes the superiority of the IBM system for high core counts. Bad efficiencies for non-power of two core counts are caused by imperfect decomposition of the computational domain resulting in a worse load-balance and inefficient communication in between multiple MPI processes. Note that for case of 8192 cores in average only 2048 nodes are processed per MPI process, but the required number of halo nodes is approximately 4800, i.e., the necessary communication is too high to efficiently process grids of the investigated size on such high core counts.

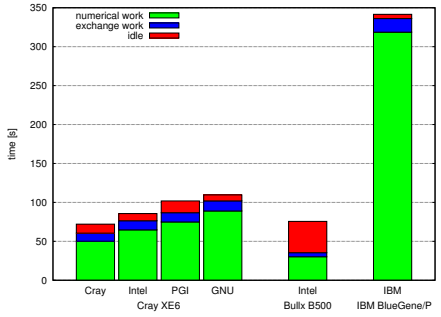


Figure 4.42.: APE-4 solver performance for several architectures and compilers

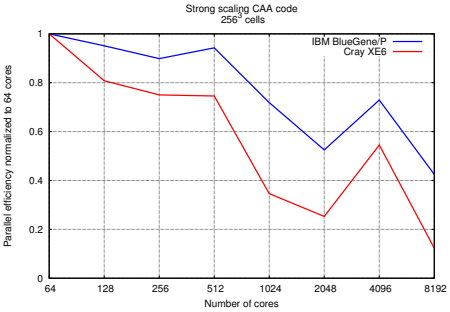


Figure 4.43.: APE-4 solver strong scaling efficiency

4.6. SPartA - Space-filling curve Partioning Algorithm

With the advent of highly distributed parallel systems in high performance computing, a major issue for algorithms is the memory consumption per process when deployed on many processes. Unfortunately, graph-based partitioning as used in ParMetis, does not cope well in this respect and does not work for applications on larger numbers of processes. In contrast to that, partitioning schemes based on a SFC ordering of elements yield the possibility to reduce the memory consumption to a constant size, independent of the number of processes. When a SFC is used to partition an unstructured mesh, the order of the distributed elements must be kept, to maintain the locality property of the resulting partitions, leading to a CCP problem for the load

balancing. Common to all CCP algorithms is the need to compute prefix sums of some weights [27]. In case of our application, these weights represent the workload of single elements in the unstructured mesh. Fortunately, the MPI standard already provides the functionality to compute these prefix sums in parallel. Although being quite powerful, these prefix sums can be implemented efficiently within MPI libraries as has been shown for example by Sanders and Träff in [29]. Though algorithms to find optimal solutions for the CCP are known, they are limited in their scalability. A very promising approximate solution to the CCP problem was suggested by Miguet and Pierson in [26]. It has the advantage, that splits between partitions can be determined completely local. The only global information required for this operation are the prefix sums and the total amount of work. These are attractive properties for a highly scalable parallel algorithm. Therefore, this heuristic is used in the implementations presented in this work. Various implementations of this SFC partitioning algorithm (SPartA) are described in the following section, afterwards the scalability of these implementations is investigated. Furthermore, we compare this method to ParMetis with respect to three criteria: i) the balancing quality, ii) the required memory, and iii) the running time.

Implementations

In the following subsection, the investigated implementations of SPartA are presented in detail and an example of the general algorithm is provided.

We consider an arbitrary mesh serialized into an one dimensional vector using a SFC. The vector has the length N which corresponds to the number of mesh cells. Weights are given as w_i for each element, where i corresponds to the global index of the element. These weights approximate the computational effort of each element, and can be derived from an on-line time measurement or a performance model. To achieve a balanced workload, the elements need to be moved between the partitions. It is the task of the balancing step to find which elements have to be moved to which process. In this paper, we use p to denote the total number processes, and assume that every such process can be uniquely identified by a process number called *rank* in the range $0 \leq rank < p$. For the proposed chains-on-chains approach on the serialized mesh, the partitions can be determined with the help of a prefix sum. In particular, we will use the exclusive prefix sum, which is defined as follows:

$$prefix(I) = \sum_{i=0}^{N-1} w_i \tag{4.1}$$

for $0 < I \leq N$ and with $prefix(0) = 0$. In order to calculate the distributed prefix sum over all processes, local prefix sums are computed, and the global offsets are adjusted afterwards using the MPI_Exscan() collective with MPI_SUM as reduction operation. After this step, each process has the global prefix sum for each of its local elements.

Under the assumption that the chosen weights correctly represent the workload of each element, the ideal work load per partition is given by $w_{opt} = \frac{w_{globsum}}{p}$, where $w_{globsum}$ is the global sum of all weights. Similar to the prefix sum, this global sum

can be obtained by using the MPI_Allreduce() collective again with MPI_SUM as reduction operation. As the last process inferred the information on the overall sum already through the computed prefix sum, an alternative uses MPI_Bcast() with the last process acting as root. Both options result in a similar running time and memory complexity. Afterwards, the splitting positions between balanced partitions can be computed locally for all processes on each process. That is no further communication is required for the decision on which elements should be moved to which processes. Thus the complete information necessary for the partitioning algorithm can be obtained with only two collective operations in MPI. Both collectives can be implemented efficiently using an asymptotic running time and memory complexity of $\mathcal{O}(\log p)$ (cf. [29, 28]).

The splitting positions for the new balanced partitions in the local elements can be found efficiently using binary search in the ordered list of prefix values. Assuming homogeneous processors, ideal splitters are multiples of w_{opt}, i.e., $r \cdot w_{opt}$ for all integers r with $1 \leq r < p$. The closest splitting positions between the actual elements to these ideal splitters can be found by comparison with the global prefix sums computed for all elements.

If the $prefix(I)$ of a local element is larger than $r \cdot w_{opt}$ and smaller than $(r + 1) \cdot w_{opt}$ then this element needs to be sent to the process with rank r. To obtain a partitioning closer to the optimal balancing, the final splitting position is decided by the minimal distance of the elements enclosing the optimal splitter: $min(|prefix(I) - r \cdot w_{opt}|, |prefix(I-1) - r \cdot w_{opt}|)$. Using this heuristic the load imbalance is limited by w_{max}/w_{opt}, where w_{max} is the maximum weight of a single element in the complete domain [26]. In general, the efficiency E of the distributed work load is limited by the slowest process, and thus cannot be better than:

$$E = \frac{w_{opt}}{max_{r=0}^{p-1}(w_{sum}(r))} \tag{4.2}$$

Where $w_{sum}(r)$ is the sum of all weights in partition r. This efficiency metric is used as a quality criterion for the resulting partitions.

Example To illustrate the algorithm, a small domain with $N = 25$ elements distributed across $p = 5$ processors is used. The initial distribution corresponds to equally-sized parts and is shown in Figure 4.44. The individual weights attached to each of the elements result in load imbalance. Such an initial load imbalance might for example arise from the input data, without any a-priori information on the computational costs for the individual elements. As the overall work with the amount of 53 should be distributed equally over 5 processes, the optimal work load for each partition in this example is $w_{opt} = 10.6$. The resulting ideal splitters $10.6, 21.2, 32.8$, and 43.4 are depicted in Figure 4.45. The thick red lines show the final splitting positions that are closest to these optimal splitting positions. The work-balanced distribution after the re-partitioning is shown in Figure 4.46. The efficiency E (cf. Equation 4.2) in this example improved from 66% in the initial distribution to 88% for the final distribution. As such, it resembles the optimal partitioning for the chains-on-chains problem in this case, besides using a heuristic to find the partition splitters. The missing gap

5	3	1	2	1	4	6	1	3	2	1	3	1	1	1	1	9	1	1	1	1	1	1	1	1
Σ=12					Σ=16					Σ=7					Σ=13					Σ=5				
Rank 0					Rank 1					Rank 2					Rank 3					Rank 4				

Figure 4.44.: Decomposed domain with different workloads per process.

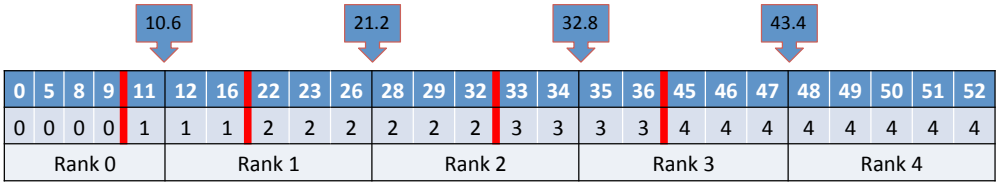


Figure 4.45.: Prefix sums of the weights and optimum prefix values between the processes marked above. The resulting splitters are marked in red and the destination process of each element is shown in the second row.

5	3	2	1	4	6	1	3	2	1	3	1	1	1	1	1	9	1	1	1	1	1	1	1	1
Σ=11				Σ=11			Σ=11						Σ=12				Σ=8								
Rank 0				Rank 1			Rank 2						Rank 3				Rank 4								

Figure 4.46.: Final distribution of the elements onto the processes.

to 100% efficiency results from two facts: i) the unfavorable ratio of the maximum individual weight and the optimal partition work load of 9/10.6, and ii) its unfortunate positioning. This is intentionally chosen to exhibit the major potential problem of this approach. However, such unlucky work load distributions are not expected in continuously load balanced flow simulations, where the optimal work load per process is normally much higher than the maximal weight of a single element. More realistic examples are analyzed with respect to their quality in Section 4.6.2.

Exchange of Elements After the information for a better balanced partitioning is known, elements actually need to be relocated. This relocation, or exchange of elements, is done via communication between processes. Unfortunately, so far only the senders know which elements need to be sent to which processes. The receivers do not know that they will eventually receive elements. When using message passing, the receivers need to be informed prior to the actual exchange of elements. Three different options to realize this exchange are pointed out here. A first method uses a regular all-to-all collective operation to inform all processes about their communication partners before doing the actual exchange of the elements with an irregular all-to-all collective operation (e.g., using MPI_Alltoallv). This method is straightforward to implement and also the method-of-choice used after a ParMetis partitioning. Since both all-to-all variants are an essential part of many applications, they have been optimized extensively for at least one of our target architectures [19]. As such, they can be expected to perform efficiently, especially in a dense case such as the initial partitioning (i.e., many exchanges between many processes). Alternatively, elements can be propagated only between neighboring processes in an iterative fashion. The elements are flagged with the destination process and forwarded in a virtual ring topology until they reach their destination. This approach can be benign when the re-partitioning modifies an existing distribution of elements only slightly. This could be expected if weights are changing slowly and re-partitionings are done frequently. In a beneficial case, only few exchanges, mainly between neighboring or at least "close" processes, are required. Unfortunately, worse cases can lead to $\mathcal{O}(p)$ forwarded messages, which becomes highly inefficient for larger number of processes. This scheme avoids the usage of $\mathcal{O}(p)$ memory needed for the input data of the all-to-all operation at the expense of a serialized communication pattern. It also reduces the required interconnect links to two for the direct neighbors in the linear list of partitions. Therefore, the iterative method offers a safe fallback if memory consumption is so important that it would otherwise inhibit the execution of the application. A third option fills the gap between these two extremes and involves a more sophisticated protocol involving a non-blocking barrier [14]. This approach for the dynamic sparse data exchange uses the fact that each sender has all the required information to start the communication. Therefore all processes begin to send their data to the appropriate processes, whereas the receiving parts just listen for messages from any source. However, this procedure results in a termination problem, as the receiving processes have no information, when to stop listening for new messages. With the help of a non-blocking barrier acting as a distributed marker, the authors solved this problem. This enables a very efficient implementation,

where each process just sends its information to the appropriate target processes, and thus minimizes memory requirements. It would therefore combine the strengths of the previously described options at the expense of implementation complexity.

Although the third option is most promising, the necessary non-blocking barrier is only proposed for the upcoming MPI version 3 and therefore not yet a standardized operation to rely on. The first option is selected for the implementation in this work, as the required memory for the all-to-all communication is not yet a limiting factor. This choice also allows a fair comparison with ParMetis, where this all-to-all operation has to be done. Even when using all 294,912 available processes on the largest Blue Gene/P installation "Jugene" at the Jülich Supercomputing Center (JSC), the memory consumption is still well below 10 MB.

4.6.1. Scaling Analysis

Two generic cases of load balancing are investigated. The first one is an extremely imbalanced mesh, for which a strong scaling analysis is performed. This mesh builds a torus consisting of 30 million elements, where the small weights are scattered across many small elements at the inner ring, whereas to the outer side fewer larger elements are found with large computational weights attached to them. It has been intentionally designed to be especially unsuited for space-filling curve approaches, though in real application cases heavy loads are normally confined in smaller local volumes. Therefore, the worst case scenario for actual applications should be covered by this example. The second test case is initialized with uniformly distributed random numbers as weights for the elements. This is used in a weak scaling analysis with $10,000$ random weights per process, and represents a more realistic simulation with smaller load imbalances. The algorithm is investigated on two very similar Intel Nehalem based cluster systems: one located at the High Performance Computing Center in Stuttgart (HLRS) and the other at the JSC. To evaluate the behavior of the presented methods on larger number of processes, the Blue Gene/P system Jugene at JSC is used. The proposed partitioning schemes are compared to ParMetis. Two different graphs are fed into ParMetis, one with the full graph of the real mesh (ParMetis on Graph), and one with a pseudo-graph resembling the linear space-filling curve only with links between immediate neighbors on the curve (ParMetis on SFC).

Memory Usage

A major concern, especially on distributed systems with limited main memory per core, is the memory required for the algorithms used. This section presents an analysis of the virtual memory usage per process. The necessary information is gathered from the status information in the pseudo file system *proc*, provided by Linux. A sleek memory footprint for the proposed algorithm is considered a key feature for the usage on future architectures on which the shrinking memory per core will become an increasing bottleneck for most applications. As the memory consumption for the partitioning algorithm is independent from the memory consumption of the application, the results can be directly used to judge the impact on the memory-footprint for any

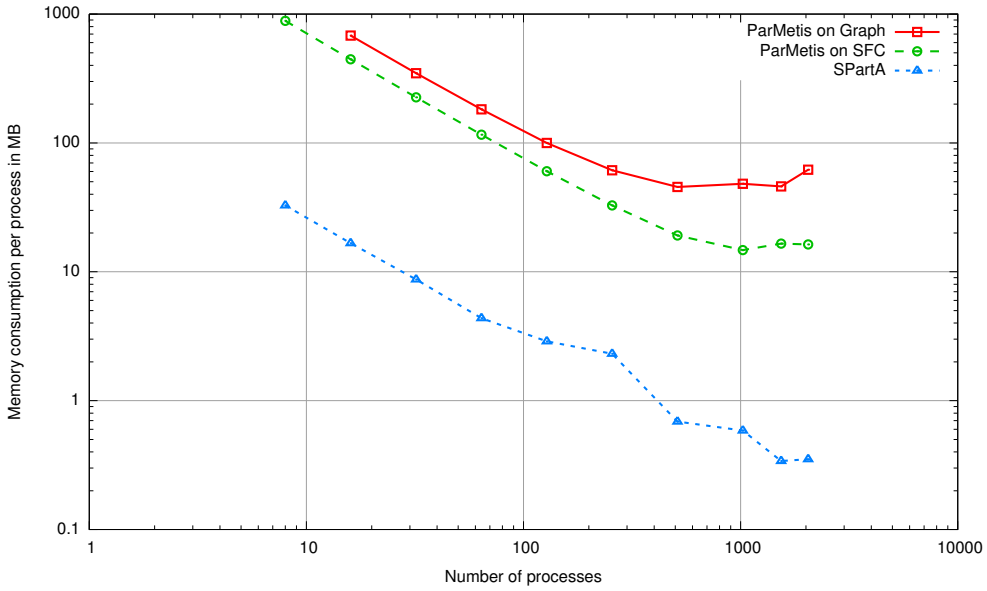


Figure 4.47.: Memory consumption under strong scaling on Laki (OpenMPI) and 30 million elements.

application. Thus the amount of memory measured in this analysis can be understood as an overhead cost attached to the chosen partitioning strategy. Figure 4.47 shows the memory usage of ParMetis, when it needs to handle the full graph for the mesh with 30 million elements, compared to the case where it has to handle only the simplified linear graph. The third series in this graph shows the memory consumption per core for the presented SPartA algorithm. This measurement was done on the HLRS Nehalem cluster Laki with OpenMPI 1.4.3 and an executable, compiled with the Intel Fortran 11.1 compiler. As can be seen, the usage of ParMetis with the simplified graph needs less memory than the partitioning with the full graph. Figure 4.48 compares the memory behavior of ParMetis to SPartA partitioning for the fixed mesh size of 30 million elements on the Nehalem cluster Juropa at JSC with ParaStation MPI 5.0. Unfortunately, the test with a complete graph of the mesh in ParMetis required a pre-allocation of all possible communication buffers beforehand on this machine, rendering any useful memory measurement for this case impossible. However, as already shown, the ParMetis partitioning with the simplified graph provides a lower bound for the full graph partitioning of ParMetis. It therefore can be used as an approximation for the comparison with SPartA. Figure 4.49 shows the memory consumption in the weak scaling experiment with $10,000$ elements per process. The strong scaling behavior shows a linear decline in memory consumption for both ParMetis and SPartA for smaller number of processes but a significant memory overhead is needed by ParMetis (up to a factor of 25). SPartA scales very well up to 4,096 cores before a memory

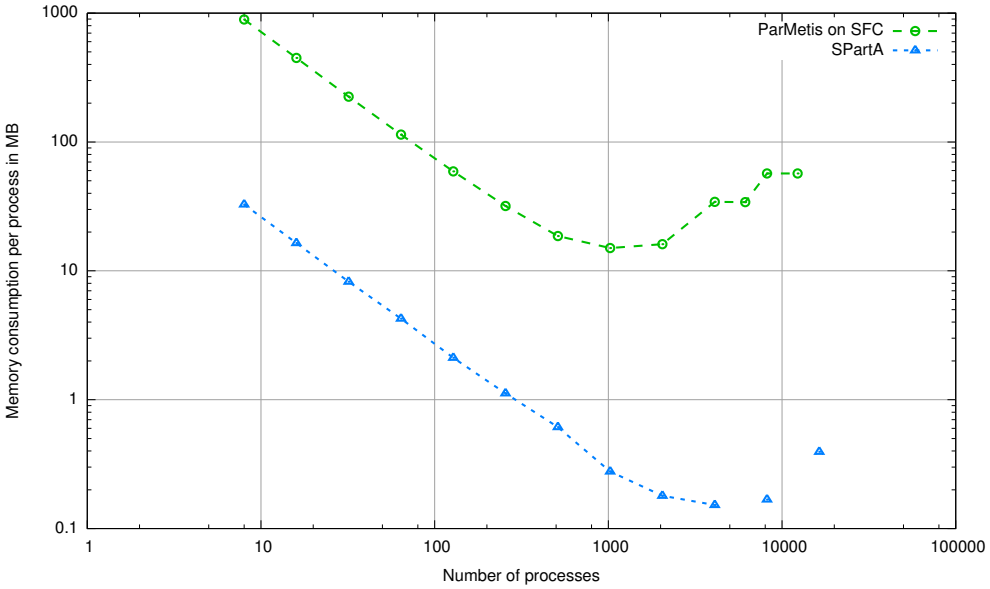


Figure 4.48.: Memory consumption under strong scaling on Juropa.

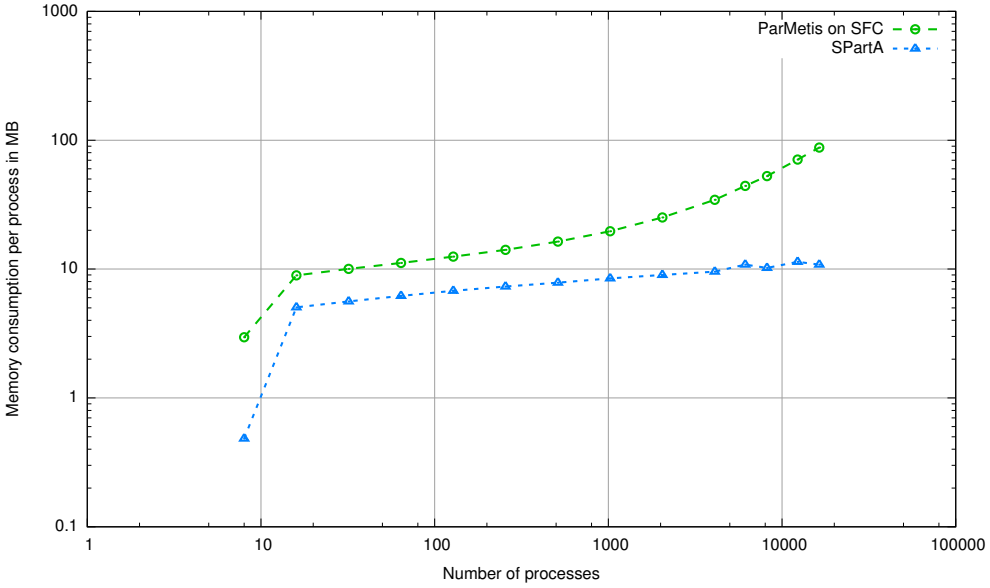


Figure 4.49.: Memory consumption under weak scaling on Juropa.

increment gets visible. These memory requirements arise from the necessary buffers for all processes in the subsequent all-to-all operation. Contrary, ParMetis scales only up to 1024 cores in memory, and then starts to demand significantly more memory. It can also be seen that the memory consumption of ParMetis depends on the next higher power of two in terms of the number of processes and therefore 6,144 cores already need as much memory as 8,192. The same behavior is observed for 12,288 cores which requires the same amount of memory as 16,384. The weak scaling of SPartA behaves very well with only a small slope and stagnation at around 10 MB per core for large numbers of cores. In contrast, a more than linear growth of memory consumption per core can be observed with ParMetis. Both ParMetis and SPartA show a jump in memory consumption from 8 to 16 cores where communication between computing nodes has to be done across the network. Both scaling results indicate excellent scaling of SPartA even on highly parallel systems with a small amount of memory per process.

Execution Time

Frequently repeated load balancing for highly dynamic simulations demand a short execution time of the balancing algorithm, especially on large numbers of cores. Figure 4.50 shows the execution times on Jugene for ParMetis both using the full graph and the simplified one, and SPartA for the fixed mesh of 30 million elements. Due to the limitation of 500 MB of main memory per core on this Blue Gene/P system, ParMetis simply fails at a certain number of cores as it requires too much memory. With the full graph it does not work with more than $16,384$ processes. Reducing the problem to the simplified graph, ParMetis succeeds up to $65,536$ cores. It should be noted, that these experiments are done without any real application data, which would reduce the memory available to the balancing algorithm even further. These memory issues are completely avoided by the simpler algorithm based on the space-filling curve. It can be seen that the execution time of SPartA is dominated by a behavior according to $\mathcal{O}(p)$ starting from $16,384$ processes. This is due to the allocation and initialization of arrays of the size p for the subsequent all-to-all operations. The weak scaling using $10,000$ elements on each process is shown in Figure 4.51, and confirms the already described trends.

Overall, it can be observed in these measurements on the different machines, that the simpler SPartA method compared to the more complicated graph-based alternative scales much better with respect to memory and time. For the extreme scaling beyond 10 thousands of cores on supercomputing machines in the foreseeable future, this is an important property. The memory restriction is a hard constraint that decides if an simulation can be done at all or not. The time consumption of the balancing influences the overall application efficiency, especially when it needs to be executed repeatedly such as in increasingly important dynamic numerical simulations.

4.6.2. Load-Balancing Quality

Figure 4.52 and 4.53 show the resulting sum of workloads $w_{sum}(r)$ for each process r after balancing the mesh with 30 million elements on $8,192$ processes using ParMetis

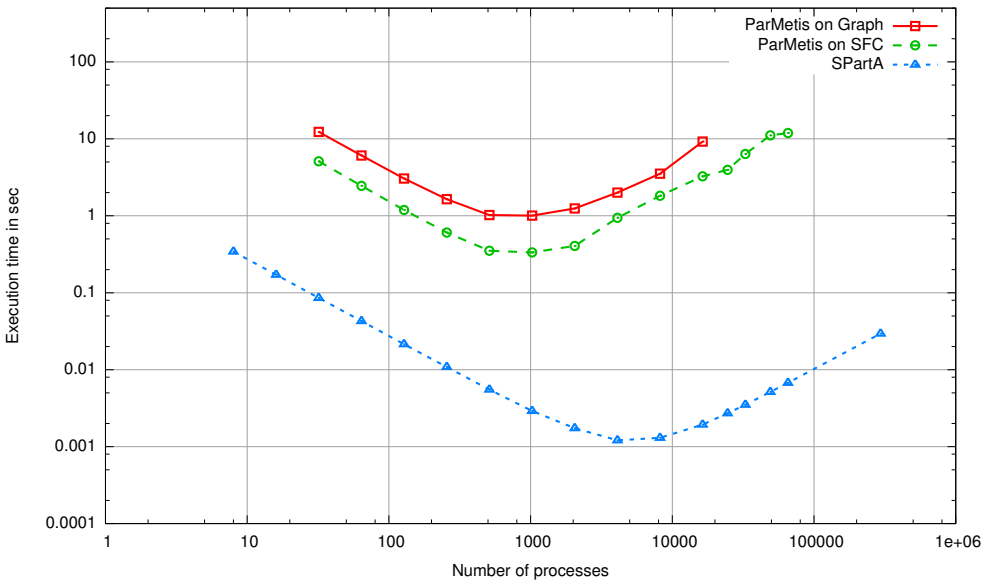


Figure 4.50.: Execution time under strong scaling on Jugene.

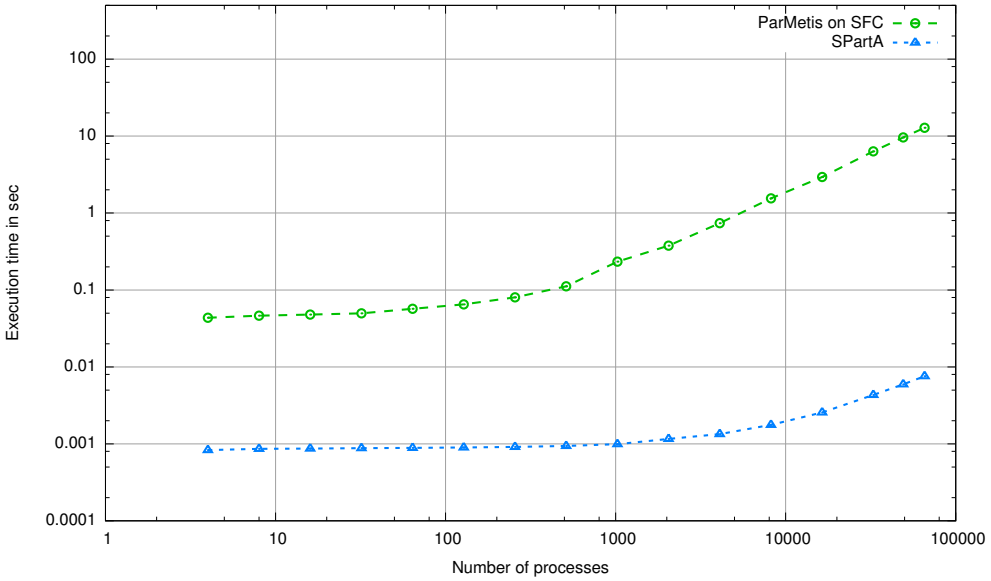


Figure 4.51.: Execution time under weak scaling on Jugene.

and SPartA, respectively. As this comparison focuses on quality and neither on memory consumption nor on running time, ParMetis was given the full graph information. Nevertheless, the remaining load imbalance after re-partitioning induces an efficiency E of only 95.8% with ParMetis, compared to 99.9% for SPartA. As can be seen in the two figures, the resulting workload distribution is much more regular when SPartA is used instead of ParMetis. That is true for overloading as well as underloading, while ParMetis has some processes, which are significantly underloaded. This is due to the fact, that the important factor for the running time is the bottleneck, that is the relation of largest load share to the average. However, it is important to note, that with such a strategy the achieved balancing is worse then it could be. Also this difference in the achievable overall efficiency has to be recovered in the graph-based approach by an accordingly reduced communication effort. However, with a highly local application as the fluid dynamic solver considered in the next section, the computational load is usually higher than the communication. Thus, potential running time advantages by the graph-based approach are diminished, and can be neglected in most scenarios.

4.6.3. Deployment in Application

SPartA and ParMetis are deployed within a compressible Navier-Stokes solver to balance the load dynamically during the simulation of a supersonic turbulent free stream. Both methods are accessible from the same interface within the application. The simulated problem is highly volatile and propagates shocks through the computational domain, resulting in drastic changes of the computational effort between time steps. We will analyze the dynamic behavior of this specific application and apply the two different partitioning algorithms to it. As already mentioned, the application is capable of adapting the time-step in each cell such, that it is optimal in the sense of the stability criterium for explicit time integration. This results in individual time-steps for each element rather than a single global time step for all. These time-steps are strongly dependent on the size and form of the cell as well as the physical phenomena within the cell. Besides this factor, there are several other influences on the computational cost of a single cell, like the adaptable order of the polynomial representation, usage of geometrical conversions for curved walls and special treatments of shocks within an element. To cover all the various aspects accurately, a time measurement is performed for the individual elements. These timings are then used as the weights in the balancing algorithm.

MPI persistent communication is used during the simulation to exchange data between adjacent elements on different processes. The mesh handling is based on GEUM [18] which employs a morton-curve to linearize the mesh and provides the solver with an initial mesh distribution. This initial distribution divides the number of elements equally on each process, regardless of their respective loads.

The investigated flow simulation is a highly turbulent super-sonic free-stream configuration using a hybrid mesh with 4 million elements. This mesh has extreme differences in the spatial resolution, and the ratio of the largest to the smallest volume is around 100. Using just a second order representation of the solution in the entire domain the simulation will contain 16 million degrees of freedom.

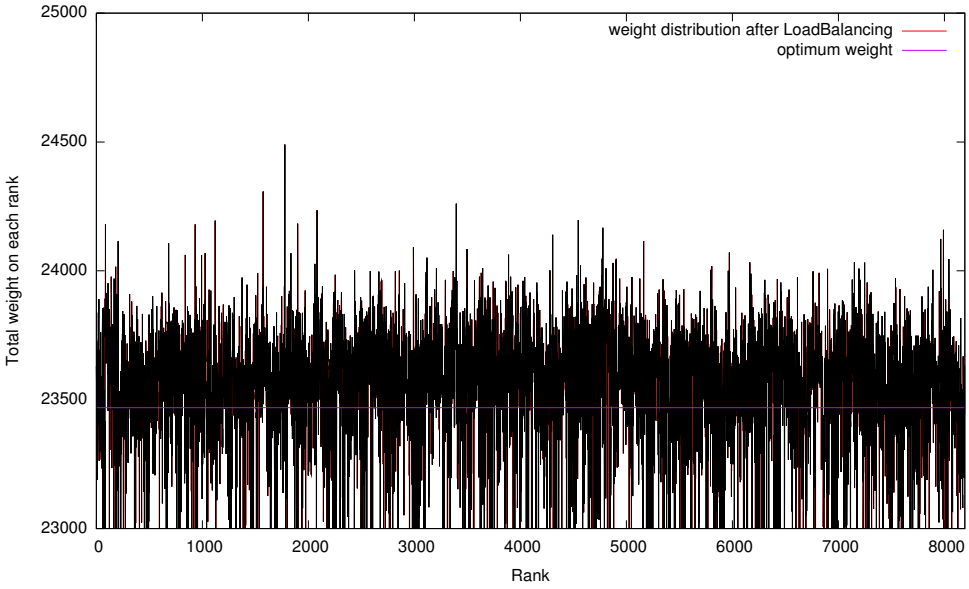


Figure 4.52.: Workload distribution on 8,192 processes after ParMetis.

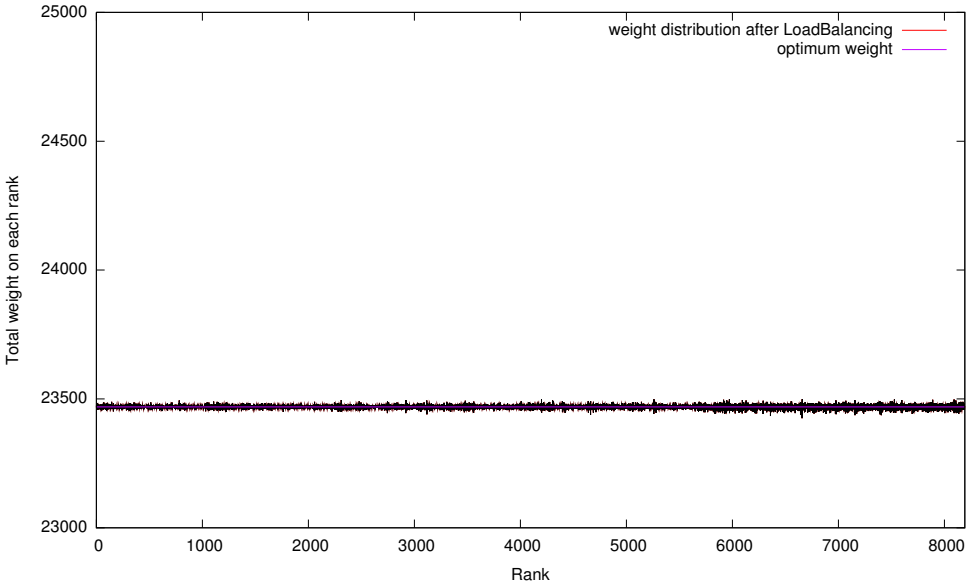


Figure 4.53.: Workload distribution on 8,192 processes after SPartA.

As already pointed out, the best possible estimation of the workload is important to obtain a balanced computation. However, this load depends on many different factors, which can also change at run time. Therefore, additional instrumentations were introduced into the code to measure the amount of time required to actually compute each element. These measurements also enable an evaluation of the current partitioning efficiency during the simulation. To avoid potentially needless re-partitioning at every step, this efficiency indicator can be used to decide if a re-partitioning is worthwhile or not. The measurement is done locally for several iterations of each element, and does not require any communication. However, there are some user-defined intervals, at which all processes have to synchronize. At these points output can be written to disk and additional administrative tasks might be executed. These points in time are natural choices to determine the current load distribution across the complete domain, as well as to perform the actual re-balancing if needed. The decision to perform a new partitioning of the mesh is based on the bottleneck factor, limiting the maximum parallel efficiency. A re-balancing is only done if the ratio of maximal load to average load falls below a user-defined threshold. While ParMetis gets the full graph information to find a better partitioning, SPartA only uses the measured weights. A fair comparison of the following simulations is achieved by using identical input parameters and the same test case on the Nehalem cluster at JSC Jülich using 1024 cores. Non-deterministic behavior of time-based re-partioning and scheduler-dependent mapping of the processes onto the actual network layout leads to different distributions of elements onto the processes. Therefore, all comparisons show some unavoidable inaccuracies. However, this error is negligible compared to the differences between the two partitioning approaches.

Scaling of the Memory

The application itself is nicely scaling in memory, as the required memory is directly attached to the elements in the mesh. Thus, a reduced partition size leads to a reduced memory footprint. Therefore, the main effects that can be observed are those that are related to the balancing mechanism. However, the application requires some more memory for the data to be transferred for the moved elements. In total, the quality of the already measured results for the balancing method still holds for the overall application. For a simulation with 1024 processes, we observed a peak memory consumption of 1.79 GB with SPartA, while ParMetis leads to a peak of 2.85 GB.

Communication overhead

Given the simple nature of the proposed partitioning algorithm, the communication surface of each computational domain is not actively optimized. It is rather based on the inherent locality given by the SFC. Therefore, an additional overhead in the communication time during the simulation is to be expected. A comparison of the measured overall times needed to simulate a given Δt between two load balancing steps shows however, that the overhead has little impact on the consumed running time, when compared to the graph-based partitioning from ParMetis. For the first five

intervals between load balancings, the ParMetis partitioning yields a total running time of 2664 seconds without the balancing itself. In contrast, the much simpler SPartA method yields 2700 seconds. Thus, the difference in running time between the two approaches is less than 2%.

Dynamic behavior of the application

Due to the dynamic nature of the flow phenomena which occurs in this particular simulation, dynamic load balancing is a key feature to ensure a high parallel efficiency during the complete simulation. Figure 4.54 illustrates the achieved parallel efficiency over the simulation time induced by the different load balancing methods. The same simulation is done three times: once without any load balancing, and two times with the different partitioning approaches. Partitioning is applied dynamically when the efficiency falls below a threshold of 80%. In the case of applied load balancing, an a-priori estimation of the workload per element is done, and the mesh is distributed accordingly before the simulation starts. Figure 4.54 does not depict the theoretical efficiency E directly after the redistribution of the elements as the weights w_i on which the distribution is based on may no longer be valid for the new distribution. The change of characteristics of elements, e.g. communication cells can become inner cells or vice versa, re-introduces new load imbalances. Therefore, the resulting efficiency E for this application scenario is lower than in the previous benchmark scenarios. Without any load balancing, the initial equal distribution of elements to all processes is used during the complete simulation. As can be seen, the efficiency of the simulation is quite similar for both partitioning methods, while only one third of the optimum is achieved without any balancing.

4.6.4. Conclusion

Memory consumption was identified as the most critical advantage of our method over ParMetis. While the graph-based approach in ParMetis was not capable to run our test case on more than $65,536$ processors, even with a drastically reduced graph, our own approach was shown to scale well up to $294,912$ processors. Potential for further memory reductions was outlined, which would satisfy the needs of future large-scale systems with lower memory-per-core ratio. Disadvantages in comparison to ParMetis in terms of suboptimal communication surface and locality during the actual simulation were found to be rather small in the order of 1-2%. On the other hand, the curve-based partitioning operation itself is orders of magnitude faster, which alleviates the above penalty to some degree. We expect to benefit from this compensatory effect especially in highly dynamic simulations where re-partitioning occurs frequently. Finally, our implementation based on MPI collective operations is supposed to deliver portable performance across a broad range of architectures.

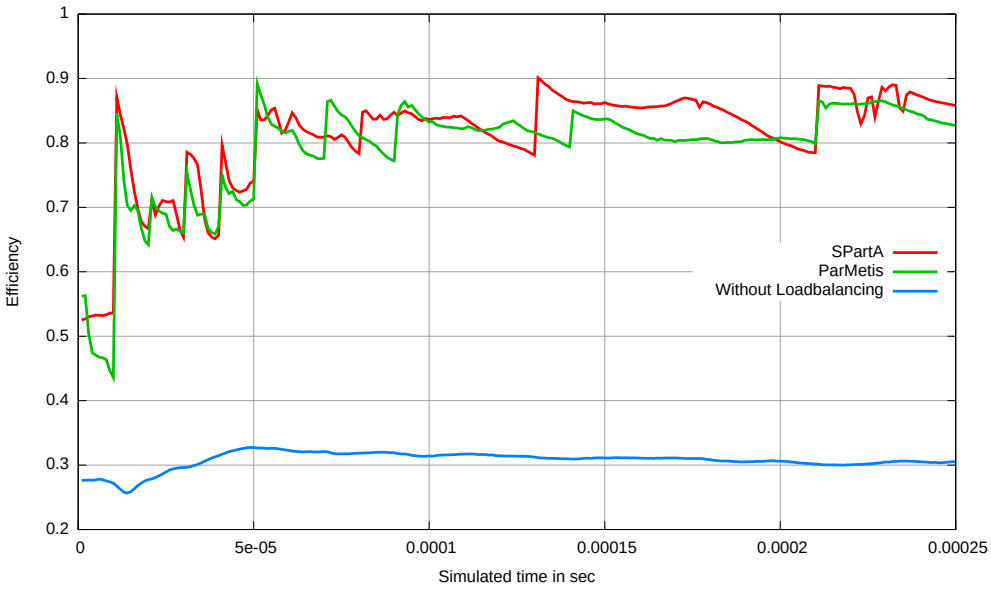


Figure 4.54.: Efficiency of the simulation over the simulated time.

4.7. Mesh Preprocessing for Highly Parallel Treatment

In this section we describe the code development done for preparing the unstructured meshes used in HALO for parallel simulations. During the project, the meshes were generated using commercial software packages Gambit, ANSA and ICEM-CFD, which were all able to write the standardized CGNS mesh format. To run a parallel simulation with the HALO code, several preprocessing steps need to be performed.

1. read in a linear mesh (CGNS)
2. find the inter-element connections
3. establish a high order mesh with curved boundaries
4. partition the mesh and assign a domain to each core of the parallel simulation

In the beginning of the project, all these steps were done by HALO at the start of each simulation, and ParMetis was used as an in-build partitioner. With an increasing number of cores used for the simulation, we found out that nearly all preprocessing steps are not scalable and the parallel implementation was very cumbersome and in some cases even unstable. In addition, for a large number of cores the Parmetis partitioner took very long computation time to partition the mesh. It is clear that for a given mesh, all preprocessing steps are always the same, thus being repeated at the start of each simulation. This is why instead of optimizing the parallel implementation, the

preprocessing steps were encapsuled in a simple single core tool, called the preproctool in the following. It is used after the mesh generation and before the simulation start. The ParMetis partitioning was replaced by the space-filling curve approach, which is much more stable and flexible. Examples of space-filling curves are shown in Figure 4.55, and a comparison of the partitioning is done in section 4.7.1.

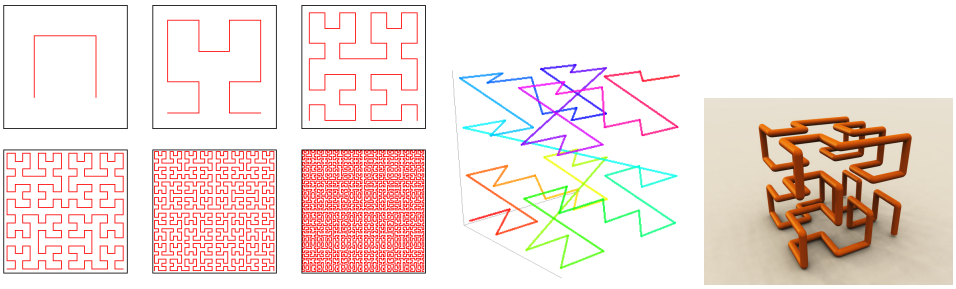


Figure 4.55.: Examples of space-filling curves, a 2D Hilbert curve (left) a 3D Morton and 3D Hilbert curve (middle and right), source: Wikipedia.

The preproctool is designed to provide a extended and *parallel readable* mesh file (using the binary HDF5 format), including element connectivity lists and high order curved element information. The elements are sorted on a space-filling curve, which leads to a one-dimensional element list. The information is stored in an array containing blocks for each element, which enables fast parallel I-O. The element list is independent of the mesh topology and the mesh partitioning becomes very simple, since a 1D list can always be distributed on an *arbitrary number* of computation cores, see section 4.6 for details. Note that the preproctool only needs to be run *once* for a given mesh, providing a single mesh file. The HALO code now only reads the HDF5 mesh file at each start of a simulation, and the number of cores can be arbitrarily chosen. The preproctool allowed to greatly reduce the overhead during the initialization of HALO and to guarantee scalability of the initialization, as well. The restart implementation in HALO was redesigned adopting the same element list fashion, writing in parallel a single HDF5 file with the restart data stored as blocks for each element. This makes it possible to easily restart a simulation on a different number of cores.

4.7.1. Partitioning with the Space-Filling Curve

In this section we compare the partitioning of the ParMetis partitioner with the space-filling Curve (SFC) approach on the same mesh, with an increasing number of domains. We use the ratio of the mean number of MPI faces per core to the mean number of elements per core as a quality measure. It can be interpreted as the ratio between communication data and local data operations. The lower this ratio, the better for the parallel computation. We compare both approaches on an unstructured hexahedral mesh of the TRUMPF nozzle in free-stream configuration with $110,000$ elements. An

estimate of the surface/volume ratio is found on a periodic cartesian hexahedral mesh, where one element block has n_e elements and the smallest possible ratio is

$$R_{cart} = \frac{6(n_e^{1/3})^2)}{n_e} = \frac{6}{n_e^{1/3}}\,. \tag{4.3}$$

#Cores	8	16	40	80	160
#Elems/Domain	13760	6880	2752	1376	688
R_{cart}	0.250	0.315	0.428	0.539	0.680
R (ParMetis)	0.094	0.170	0.317	0.467	0.652
R (SFC)	0.204	0.349	0.562	0.729	0.948
R (SFC) / R (Parmetis)	2.17	2.05	1.77	1.56	1.45

Table 4.6.: Comparison of the ParMetis and space-filling curve (SFC) mesh partitioning.

This ratio must be different on an unstructured mesh with boundary conditions. It is clear that for increasing number of domains, the ratio increases until the limit of $R_{cart} = 6$ for 1 element per core. In Table 4.6 the ratios for ParMetis and space-filling curve are summarized. ParMetis is a graph-based partitioner, whereas the space-filling curve does not account for the mesh topology. Especially for a low number of cores, ParMetis will give better results, since it accounts for non-communication faces on the boundaries of the domain. Due to the boundary conditions, ParMetis reaches lower ratios than for the SFC.

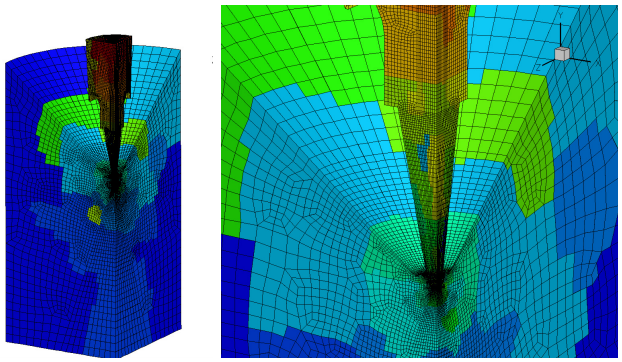


Figure 4.56.: Domain decomposition on 160 cores of the TRUMPF nozzle mesh using the space-filling curve.

However, the comparison shows that difference between both approaches diminishes for increasing number of cores. Thus, for the preproctool, the SFC approach is used,

since data structures and the domain decomposition process are greatly simplified and domain decomposition results are similar on large number of cores. The domain decomposition of the nozzle mesh on 160 cores using the SFC is shown in Figure 4.56.

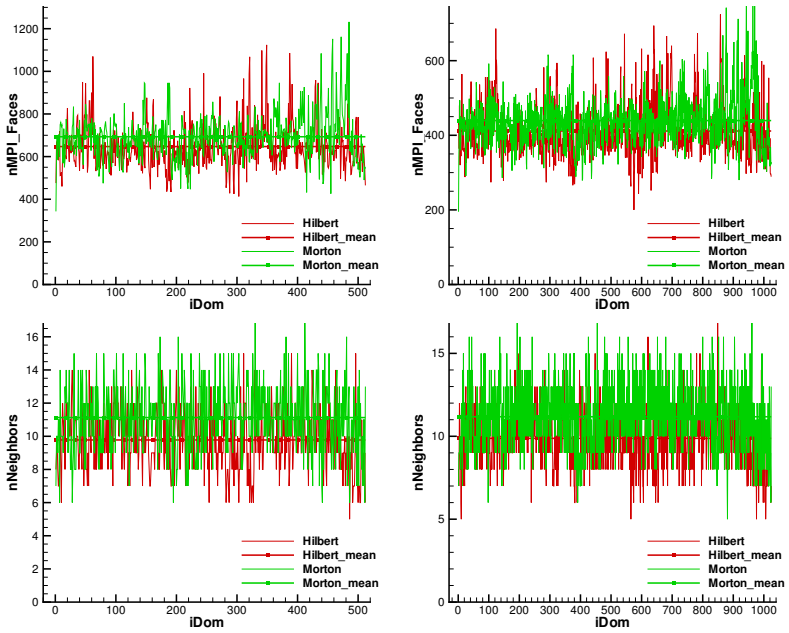


Figure 4.57.: Comparison of the Hilbert and Morton curve, on 512 cores (left) and 1024 domains (right).

The space-filling curve is not unique, the Morton curve and the Hilbert curve mentioned above were implemented. Both implementations were compared on the subsonic jet mesh, which is a cartesian stretched hexahedral mesh 357216 elements. The comparison in Figure 4.57 shows that the number of MPI neighbor cores and the number of MPI faces is lower for the Hilbert curve on both domain decompositions of 512 and 1024 cores. It shows that the domains are more compact for the Hilbert curve, leading to better results than the Morton curve.

4.7.2. Curved Mesh Generation

In complex geometries like the ones defined in this project, wall boundaries are normally curved. The accuracy of a high order method can be deteriorated by the use of meshes with linear edges on the curved wall boundaries. Typically, the commercial mesh generators provide only linear meshes. The elements adjacent to the curved boundaries must be curved too, then the high order accuracy can be maintained.

Three approaches to provide the extra information of the curved geometry were

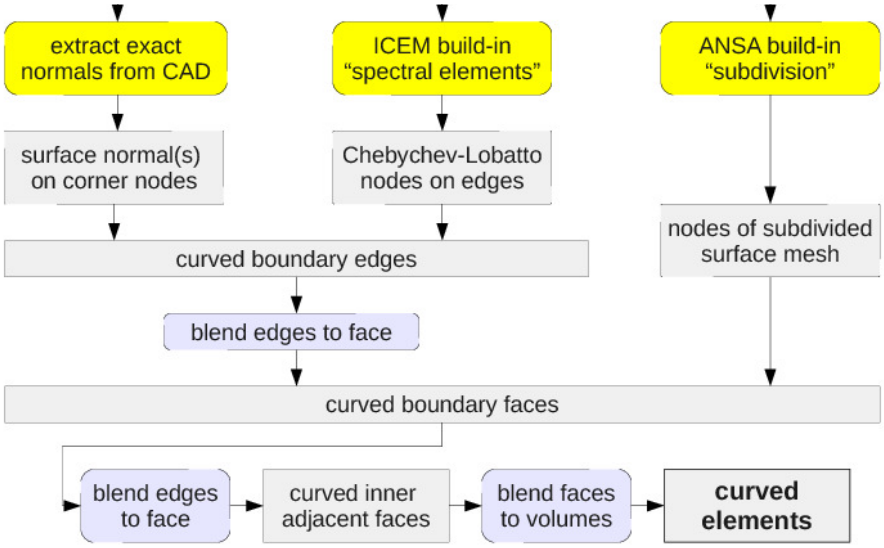


Figure 4.58.: Flowchart of the curved mesh generation process in the preproctool.

developed in the course of the project. The first is to provide normal vectors on surface element corner nodes. This can be done for simple geometries by exact functions, or using a CAD tool to extract the normal vectors directly from CAD. The second approach uses a refined surface mesh by subdivision of the original surface mesh, which can be provided for example by the commercial mesh generator ANSA. The third approach is an in-build function of the mesh generator ICEM, where high order Chebychev-Lobatto nodes are assigned to element edges lying on curved boundaries. The curved mesh generation is incorporated into the preproctool. A flowchart of the curved mesh generation process is shown in Figure 4.58. More details to the different approaches can be found in [13]. An example of the subdivision approach for the TRUMPF nozzle mesh is shown in Figure 4.59.

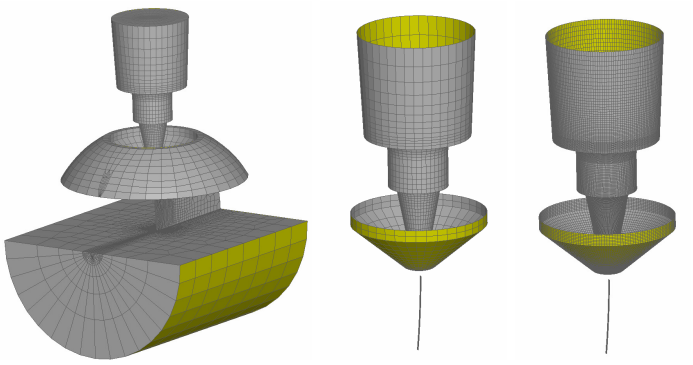


Figure 4.59.: Subdivision approach for curved mesh generation of a TRUMPF nozzle mesh (left), and two steps of subdivision of the curved surfaces (middle and right).

5. Academic Test-cases

This section is dedicated to academic test-cases regarding turbulence modeling and Large Eddy Simulation (LES). Since turbulent behavior is very difficult to compare, we present in this section simulations of test cases which are well documented in literature. The simulations are not only for validation purpose, but furthermore we want to assess how implicit and explicit modeling can be done for a high order method.

5.1. Turbulence Modeling: Taylor-Green Vortex

The simulation of turbulent flows with DG methods is a relatively new topic and the behavior of turbulence models like LES have to be investigated thoroughly. Here we will focus on the analysis of the Taylor-Green vortex [5], where a laminar-turbulent transition produces isotropic homogeneous turbulence by consecutive break up of large vortices into smaller ones, as shown in Fig. 5.1. We implemented a sub-grid-scale

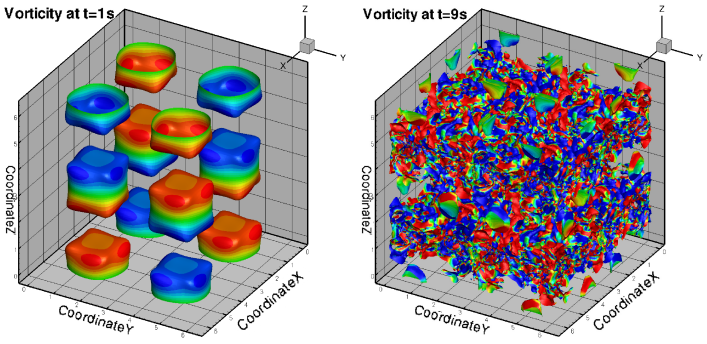


Figure 5.1.: Taylor-Green vortex ($Re = 5000$, $Ma = 0.1$), time evolution of the vortical structures (iso-surfaces of vorticity, left $t = 1$, right $t = 9$)

model, namely the standard Smagorinsky model. The sub-grid-scale viscosity is defined as $\mu_{\text{sgs}}^{\text{SM}} = (C_S\Delta)^2 \left|\tilde{\underline{\underline{S}}}\right|$, with the Smagorinsky constant C_S, the filter width Δ and the filtered stress tensor $\tilde{\underline{\underline{S}}}$. The resolution of a DG based approximation is determined by the size of the element Δx and furthermore by the number of internal DOF, i.e. by the polynomial degree N. Typically, the resolution is proportional to $\sim \Delta x/N$, which is used to determine the filter width of our LES discretization.

The LES results of coarse Taylor-Green vortex simulations are shown in Fig. 5.2. We choose a Mach number number $Ma = 0.1$ to compare our results with incompressible

DNS calculations [12]. For the discretization, we choose $N = 8$ with 4^3 elements. The filter is defined via $\mathbb{L}_2$-projection onto polynomial degree $\tilde{N} = 4$. In a preliminary step, the effect of different Smagorinsky constants for different Reynolds numbers is shown. We kept the overall resolution constant, to solely investigate the different model effects. The standard Smagorinsky constant $C_S = 0.18$ results in too much dissipation. Tuning the constant reveals that it is possible to find a suitable sub-grid-scale viscosity. However, the results demonstrate that this optimal constant depends on the simulated problem, as we get $C_S = 0.13$ for $Re = 200$ and $C_S = 0.09$ for $Re = 400$. More investigations of the different sub-grid-scale model aspects (Smagorinsky constant, filter, definition of filter width, higher Re number) are necessary.

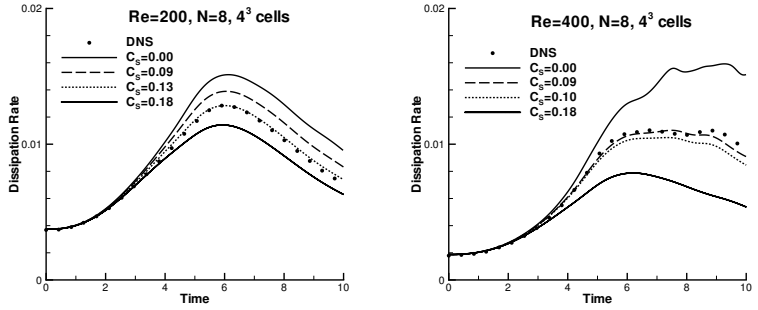


Figure 5.2.: LES of Taylor-Green vortex for different Smagorinsky constants C_S

In Fig. 5.3 we plot the dissipation rate for a low order and high order discretization without modeling and the result of the LES from Fig. 5.2. The low order simulation is over-dissipative, even for a very high resolution of $2.1mio$ DOF, whereas the high order simulation perfectly reproduces the DNS results. We listed the computational cost of each simulation to show that a successful high order LES is very promising.

5.2. Turbulent Subsonic Round-jet

The subsonic turbulent round-jet is a prominent candidate to investigate free-stream turbulence and is well studied in literature. The paper by Bogey and Bailly [3] presents well documented Large Eddy Simulations of a compressible subsonic jet at Mach number $Ma = 0.9$ and different Reynolds numbers. The numerical method employed by Bogey and Bailly is a 13 point Finite Difference scheme of 4th order and a classical Smagorinsky model is used as LES model. They discretize the jet with 12.4 million grid points on a stretched structured mesh. The jet is imposed by a laminar inflow profile, which is disturbed inside of the developing shear layer. Thus no geometry is involved and a cartesian mesh is used.

We choose a moderate Reynolds number of $Re = 5000$ for comparison, the vorticity contours of the reference computation is shown in Figure 5.4. We adapt the same mesh stretching of the paper, but, of course, use a coarser mesh because of the sub-grid

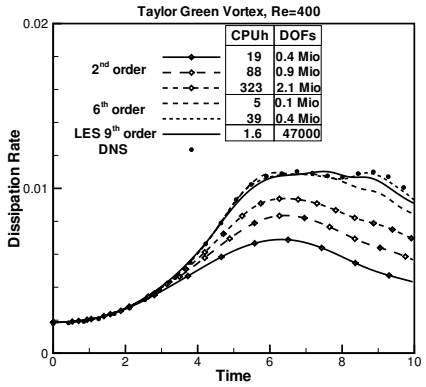


Figure 5.3.: Taylor-Green vortex ($Re = 400$, $Ma = 0.1$), dissipation rate for varying discretizations, with computational costs

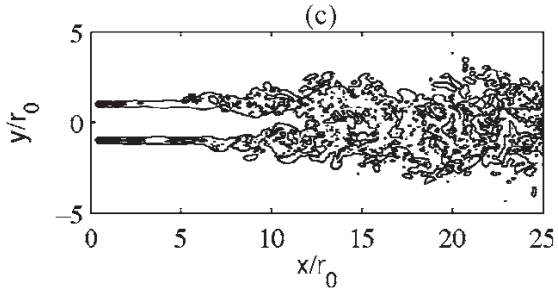


Figure 5.4.: Vorticity plot at $Re = 5000, Ma = 0.9$ of the reference solution from Bogey and Bailly[3]

resolution of the DG scheme. Three mesh densities are studied, with the nomenclature E15/E20/E25. The number is derived from the resolution of the innermost zone of a size $[4 \times 4 \times 25]r_0$, with an element distribution of $[15 \times 15 \times 46]$, $[20 \times 20 \times 62]$ and$[25 \times 25 \times 78]$. The total domain size including damping zones around the jet and in its wake was $[16x16x40]r_0$. Adding the elements of the damping zones results in a total number of elements of 78033/192500/357216 . A fourth order DG scheme is used, leading to 1.56/3.85/7.14 million degrees of freedom. The Smagorinsky model with a $C_s = 0.1$ was used. All simulations were run on the RZ cluster in Aachen on 1024 cores, with a wall-clock time of $2.82h/5.46h/7.78h$ for 400 time units.

In Figure 5.5, the instantaneous and mean Mach number and the mesh is of the three computations is shown. The averaging interval is $t = 200 - 400$. The higher the resolution, the smaller length of the potential core of the jet. In comparison to the reference data with a potential core length of $10r_0$, the potential core is too long for all three simulations.

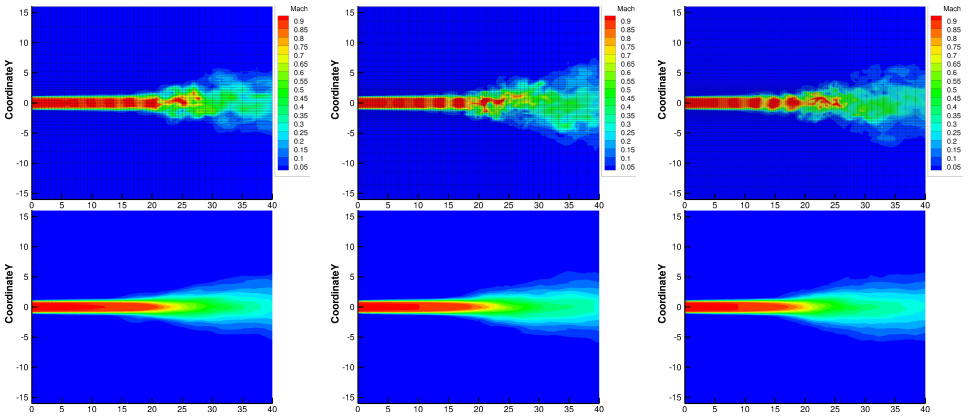


Figure 5.5.: Instantaneous (upper row) and mean Mach number at simulation time $t = 400$ for resolutions E15 / E20 / E25 (from left to right), in the slice plane $z = 0$.

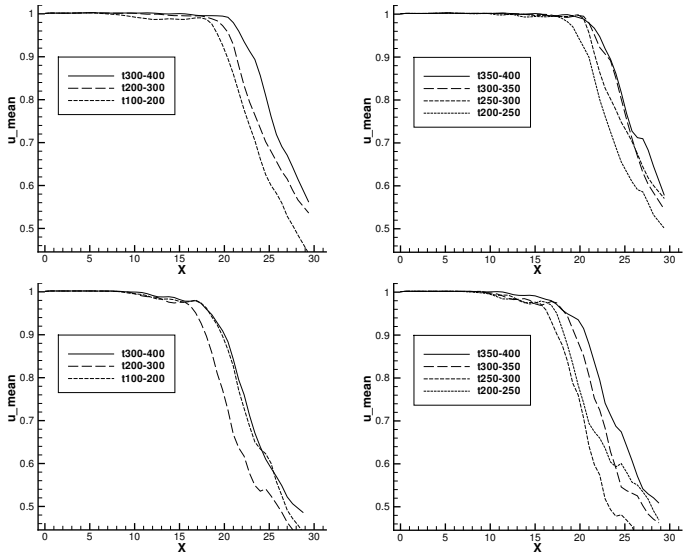


Figure 5.6.: Statistical convergence of the mean center-line velocity for resolution E15 (upper row) and E25 for an averaging period of 100 and 50 time units.

In Figure 5.6 the statistical convergence of the mean center-line velocity is shown for different averaging intervals. The jet is fully developed after $t = 200$, whereas Bogey and Bailly claim that at $t = 100$, the jet is fully developed.

The length of the potential core depends on the growth of the shear layer instability. To trigger growth the instability, the forcing is crucial. The forcing term was not

exactly adapted from the reference paper, and only inflow forcing with amplitudes of 5% of the jet center-line velocity were used. The presented simulations show that the instability is damped too much, and the potential core is longer, around $20r_0$ instead of $10r_0$. Since the high resolution remains only until $x = 25$, the turbulent jet is damped by the coarser mesh. A simulation with a higher forcing of 20% did not make any difference, and reducing the Smagorinsky constant to 0.05 led to an instable computation. Unfortunately, no more simulations were performed to further investigate the subsonic jet. It seems that the a simple Smagorinsky model may be too dissipative in the case of a high order under-resolved simulation. At least a combination of stabilization techniques for the high order polynomial to be able to decrease the Smagorinsky constant, and thus decrease overall dissipation, could be helpful.

6. Bosch Gas Injection Nozzle

6.1. Test case description

6.1.1. Compressed Natural Gas Injection

The ongoing development of efficient natural gas powered engines aimed at reducing carbon dioxide emissions impels the development of improved storage and transport components. Especially in automotive applications, high standards concerning safety, quality and convenience are set. One important component of the engine management system in gas powered vehicles is the gas injection valve which manages the fuel mass flow. To produce an optimal air/fuel mixture the gas is expanded through the valve from its operation pressure into the intake manifold. During the injection process, a supersonic highly turbulent jet is established in the duct. The sound waves produced by the aeroacoustical effects interact with the duct walls, the jet flow and themselves (see Figure 6.1). The resulting noise is audible in the direct environment of the car and is subject of ongoing investigations. Concerning simulation techniques this application is very challenging and was first studied by Schönrock [30] using a commercial CFD software (Ansys CFX). Extensive design studies become unfeasible due to high computational effort and long turn around times shown in Figure 6.2. Hence, the application of novel specialized software using a parallel and scalable Discontinuous Galerkin scheme is of great interest.

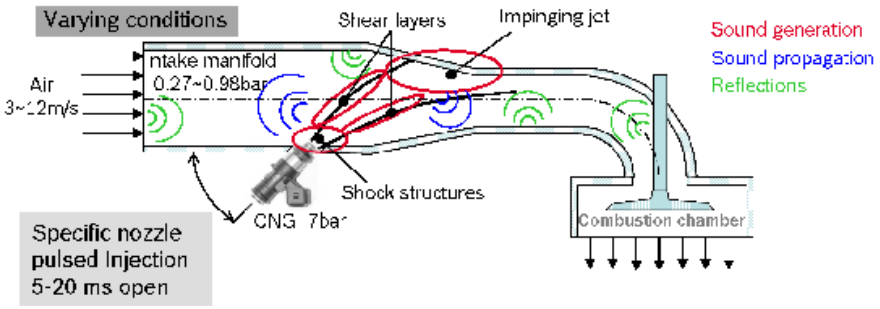


Figure 6.1.: Aeroacoustics during gas injection process

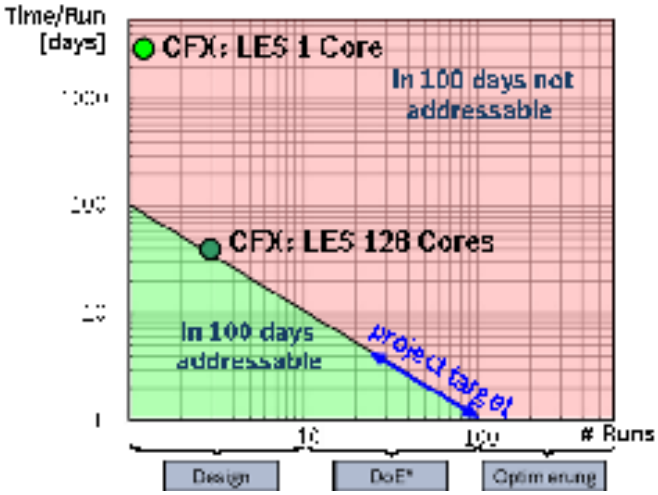


Figure 6.2.: Simulation time with Ansys CFX

6.1.2. Free-stream Configuration

The complex system with intake manifold is simplified to a free-stream configuration which facilitates the accessibility for experiments and the modeling in numerical approaches. In this setup, air is expanded through the valve from its operating pressure to ambient conditions ($\rho_\infty = 1.2\frac{kg}{m^3}$ $u_\infty = v_\infty = w_\infty = 0.0\frac{m}{s}$ and $p_\infty = 100000Pa$). The nozzle exit geometry of the valve, as depicted in Figure 6.3, ends in a silencer duct with a main-diameter of $6.11 \times 10^{-3}m$. In total, the length of the geometry is $3.74 \times 10^{-3}m$. At the outlet of the silencer, a notch reduces the inner diameter to $5.95 \times 10^{-3}m$. At the base of the duct a step narrows the flow domain to a diameter of $2.26 \times 10^{-3}m$. In the bottom of the duct, four kidney shaped orifices are symmetrically positioned around a truncated cone located at radius $r_0 = 1.71 \times 10^{-3}m$ (measured from middle-axis of the duct to the averaged free-stream center). The Reynolds number based on the width of an orifice is $Re = 52000$ ($\mu = 3.7 \times 10^{-5}\frac{m^2}{s}$, $Pr = 0.72$).

To capture the acoustic wave generation as well as the flow and wave interaction a simulation of the entire three-dimensional geometry is performed. Due to high flow speed and the small size of the geometry, a very high resolution is required within the silencer geometry. Therefore, this domain demands the largest part of computational efforts in the simulation. With a calculated Kolmogorov length of $\sim 10^{-7}m$ a full direct numerical simulation of all turbulence scales within the silencer is out of reach. Thus, all simulations are under-resolved, whereas the influence of resolution on aerodynamic and aeroacoustic results is subject of ongoing investigations.

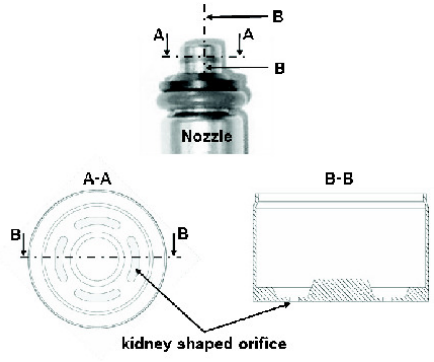


Figure 6.3.: Specification of nozzle exit geometry

6.2. Aim of the project activity

The goal in this project is to show the feasibility of a direct noise computation using the STEDG code HALO developed at the Institute of Aerodynamics and Gas dynamics, University of Stuttgart. The respecting simulation model of the free-stream configuration shall be able to compute both, the sound producing turbulent, supersonic jet as well as the sound propagation in the near and far field. Experimental data to validate numerical results will be provided and finally the required solution quality and turn around times will be compared with solutions provided by Ansys CFX. A distinct reduction of the required simulation time by a factor of about 100 is aimed (see Figure 6.2) and would allow further numerical studies to predict sensitivities towards parameter and geometry variations. In a final step, a simulation model of the more complex intake manifold configuration to show the applicability is aimed.

6.3. Experiments

In order to evaluate numerical results concerning their solution quality, experimental data is needed. Thus, the free-stream configuration, in which air is expanded through the injection nozzle to ambient conditions, was investigated with Schlieren optics, Particle Image Velocimetry (PIV) and microphone sensors. In the following section, the used test rigs as well as results are shown and discussed.

6.3.1. Shadowgraph measurements

Experimental setup

Shadowgraph photography is a method to visualize steady and unsteady flow structures of density varying flows. The objective of this measurements is to evaluate the flow features like jet development and presence and shape of shock structures at different

operation pressures. To capture all relevant features, the silencer duct prescribed in section 6.1.2 was removed.

The so called Schlieren method visualizes spacial gradients of the refraction index due to density gradients in a translucent fluid. In the experimental setup shown in Figure 6.4 bright light is emitted by an LED flash lamp and focused by a focal lens. Subsequently, an aperture limits the used light to guarantee monochromatic and uniform properties. A second lens parallelizes the light in the test region where the injection nozzle is positioned and the density gradients of the gas jet refract the incoming light. Finally all light is focused again by a third lens and redirected by two mirrors. In the focal point of this lens, a sharp Schlieren blade is applied to remove light that is bent towards the blade. So finally the arising bright and dark patterns in the photo are related to regions where positive or negative density gradients in stream-wise direction arise in the flow field.

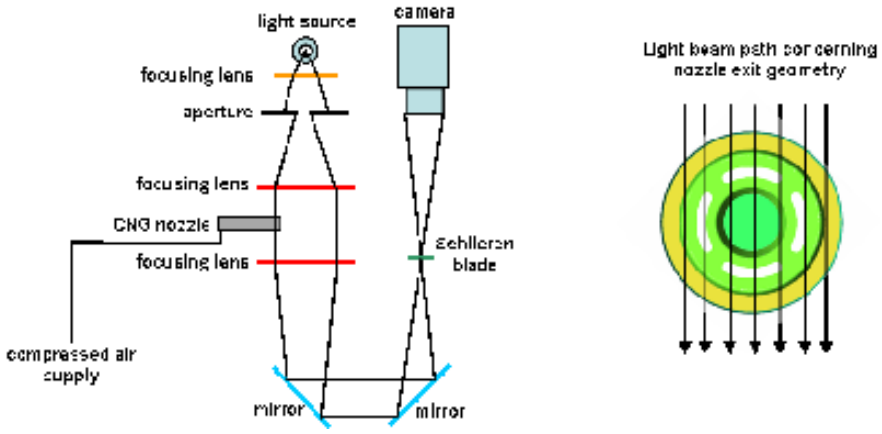
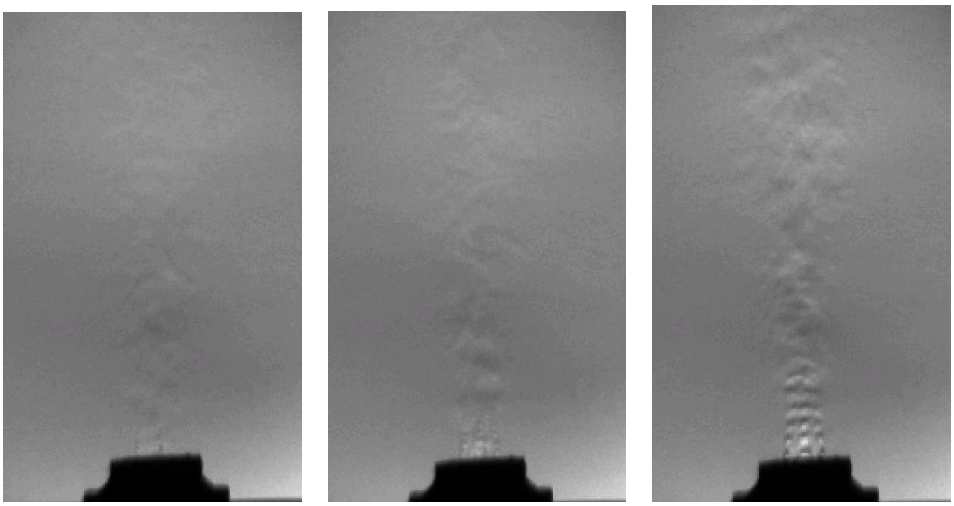


Figure 6.4.: Sketch of shadowgraph measurement setup

Results

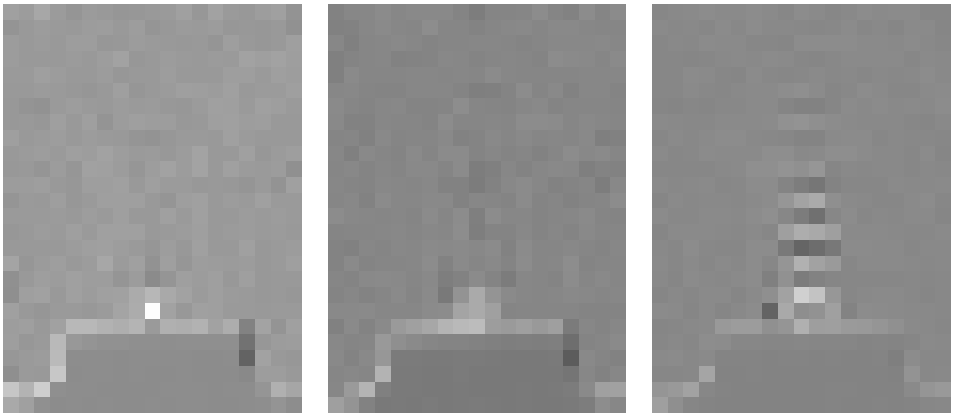
Figure 6.5 represent shadowgraph pictures of the instantaneous density gradient in stream-wise direction for different operation pressures. By averaging over a total of 50 images the instantaneous density fluctuations vanish and the quasi stationary shock structures are highlighted (see Figure 6.6). To visualize all relevant flow characteristics, the silencer duct shown in Figure 6.3 was removed. As the experimental setup provides information of the integral value of the density gradient along the whole light path through the jet flow, the pictures can be used for comparisons regarding quality.

The flow conditions at the outlet of the injection nozzle result in four separate jets downstream the kidney shaped orifices combining to a turbulent flow (see Figure 6.5). In dependence of the operation pressure a system of shock structures establish over



(a) 2 bar (b) 4 bar (c) 7 bar

Figure 6.5.: Instantaneous shadowgraphs for different inlet pressures.



(a) 2 bar (b) 4 bar (c) 7 bar

Figure 6.6.: Averaged shadowgraphs for different inlet pressures.

each nozzle exit. One can see in the averaged shadowgraph images in Figure 6.6 that no obvious shocks are detected operating the injector with 2 bar. By increasing the operation pressure up to 4 bar, 2 shock cells are observed until their breakup 40 mm downstream the nozzle exit. With an operation pressure of 7 bar the number of distinct shock cells increases to 5 and an enlargement of their spacial dimension is observed. The final break up and combination into a turbulent jet is observed 80mm downstream the nozzle exits. Additionally, the intensity fluctuations observed in the images further downstream the nozzle exits indicating density gradients due to turbulent flow motion are more intense with increasing operation pressure.

So in conclusion, the shock and turbulence structures of the fluid flow through the injection valve are highly dependent on the operation pressure. Thus, it is also expected that the resulting aeroacoustics will be louder with increasing operation pressure.

6.3.2. PIV measurements

Experimental setup

2D-2C Particle Image Velocimetry (PIV) is a contact-free, optical measurement technique used to detect the 2 dimensional velocity components in fluid flows. This method is based on the detection of the movement of added seeding particles within a known time period. Thus, a PIV system consists following steps which have to be considered in the test setup.

- seeding with appropriate particles
- illumination of the seeding particles
- recording of the light scattered by the particles
- evaluation of the spacial movement of the particles
- post processing of results

Figure 6.7 shows a picture and a respecting sketch of the PIV test rig with the used components. The CNG injection nozzle (1) is installed in a Plexiglas capsule (4) and is operated with pressurized air available in the test laboratory (3). Before the air is expands through the nozzle it is seeded with small silicon oil (DEHS) drops with a mean diameter of 1 μm generated by an special aerosol generator (2). These drops, following the flow without significant slip, are illuminated by a pulsed Nd:YAG laser system (5)producing 532 nm light pulses of 8 ns with a maximum puls energy of 50 mJ. The minimum time between two pulses is 400 ns obtained by a parallel operation of two lasers. The round laser profiles are transformed into a laser sheet by a system of optical lenses (6). Finally, the scattered light by the seeding particles is detected with a CCD camera (LAaVision Imager Intense) with a spacial resolution of 1376x1040 pixel.

The camera is operated in double frame mode taking 2 pictures in a very short time (min. 400 ns). The movement of the seeding particles is evaluated out of these

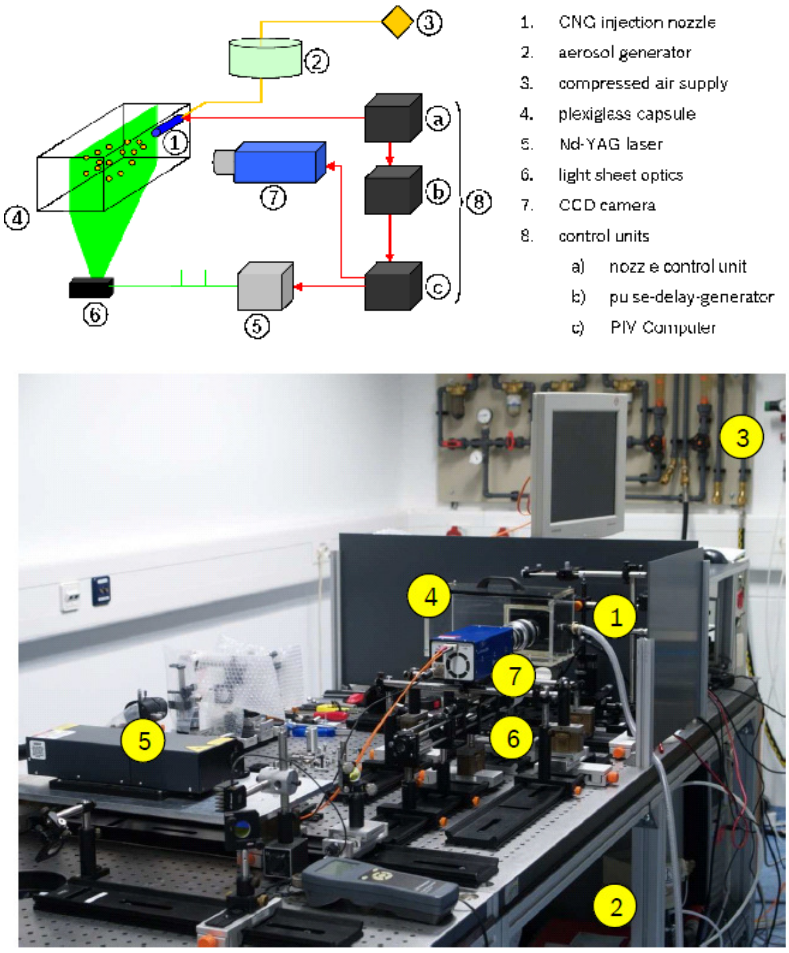


Figure 6.7.: Sketch and picture of the PIV test rig.

two pictures by a cross correlation algorithm and the velocity is determined under the knowledge of the time between the two pictures. The Plexiglas capsule prevents the camera, the laser and the optical components to be polluted by the ejected oil particles. In order to reduce refraction errors two areas, one for the laser light sheet and one for the detection camera, are replaced by optical glass with a refraction index of 1.15 (see Figure 6.8).

In order to manage these processes a trigger chain is build based on the master signal opening the injection nozzle (see Figure 6.10. The signal is delayed by a Puls-Delay-Generator and forwarded to to the PIV module in a computer. With this unit the camera and laser are synchronized to obtain proper images.

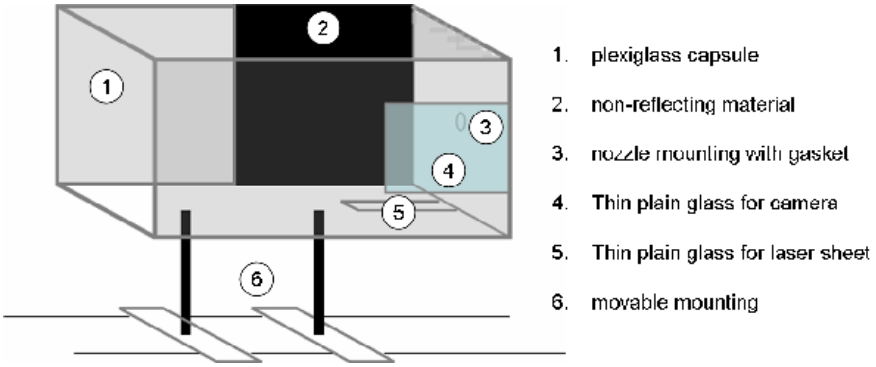


Figure 6.8.: Sketch of the glass capsule.

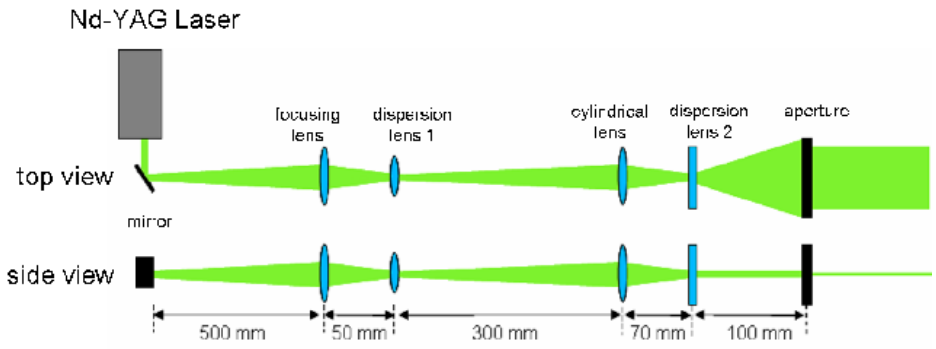


Figure 6.9.: Optical components to produce a laser sheet

The exposure time of the two frames last 10 μs and 1000μs, respectively, with a dead time of 400 ns. The illumination by the laser pulses only last 8ns and are positioned and the end of frame 1 and at the beginning of frame 2 enabling a very short time of

0.5 μs between two pictures. In order to obtain a comparable intensity distribution in both frames, the two lasers are controlled using a specific percentage of their maximal power (see Figure 6.11).

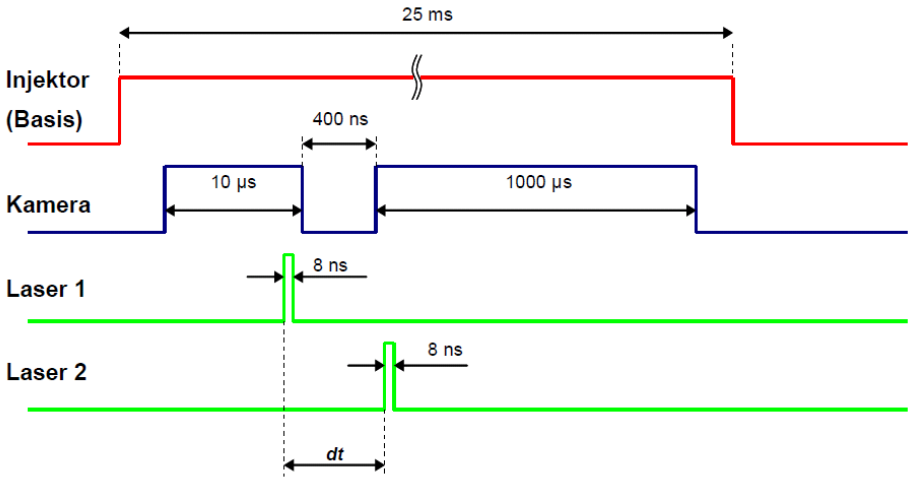


Figure 6.10.: Sketch of the trigger chain

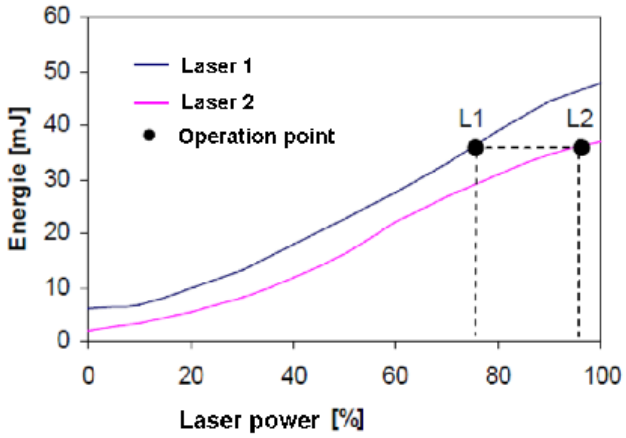


Figure 6.11.: Operation points for laser

Results

The Particle Image Velocimetry calculating the velocity vectors out of the source images using a cross correlation technique results in a instantaneous vector map. By averaging over 100 samples one can detect the averaged velocity field components and extract velocity profiles over specific positions (e.g. centerline, radial profiles at different positions in stream-wise direction). This post-processing procedure is depicted in Figure 6.12 showing the respecting images of the results operating the valve with 2 bar.

In order to find the optimal settings for each operation pressure measurements were performed with variation of the time between two pictures δt, resolution in space, window size in PIV setup and operation pressure of the injection valve. In Tabular 6.1 all performed measurements are listed.

operation pressure [bar]	$\delta t[\mu s]$	resolution in space	window size
2	5	40x50mm	32x32
2	5	40x50mm	64x64
2	3	40x50mm	32x32
2	1	40x50mm	32x32
2	0.5	40x50mm	32x32
2	0.5	40x50mm	64x64
4	5	40x50mm	32x32
4	5	40x50mm	64x64
4	3	40x50mm	32x32
4	1	40x50mm	32x32
4	0.5	40x50mm	32x32
4	0.5	40x50mm	64x64
4	0.5	16x20mm	32x32
7	3	40x50mm	32x32
7	3	40x50mm	64x64
7	1	40x50mm	32x32
7	0.5	40x50mm	32x32
7	0.5	40x50mm	64x64
7	0.5	16x20mm	32x32

Table 6.1.: Performed PIV measurements

Based on the results, which will not be explicitly discussed in this report, the optimal settings were identified and are highlighted green in 6.1. Additionally, the region in which confidential results are obtained is confined from $x = 4 \cdot r_0$ until $x = 28 \cdot r_0$. Due to the high seeding particle density near the nozzle exit from $x = 0 \cdot r_0$ until $x = 4 \cdot r_0$, overexposure effects prevent a proper distinction between single particles in the source images which is crucial to obtain proper PIV results. Figure 6.13(a) and Figure 6.13(b)

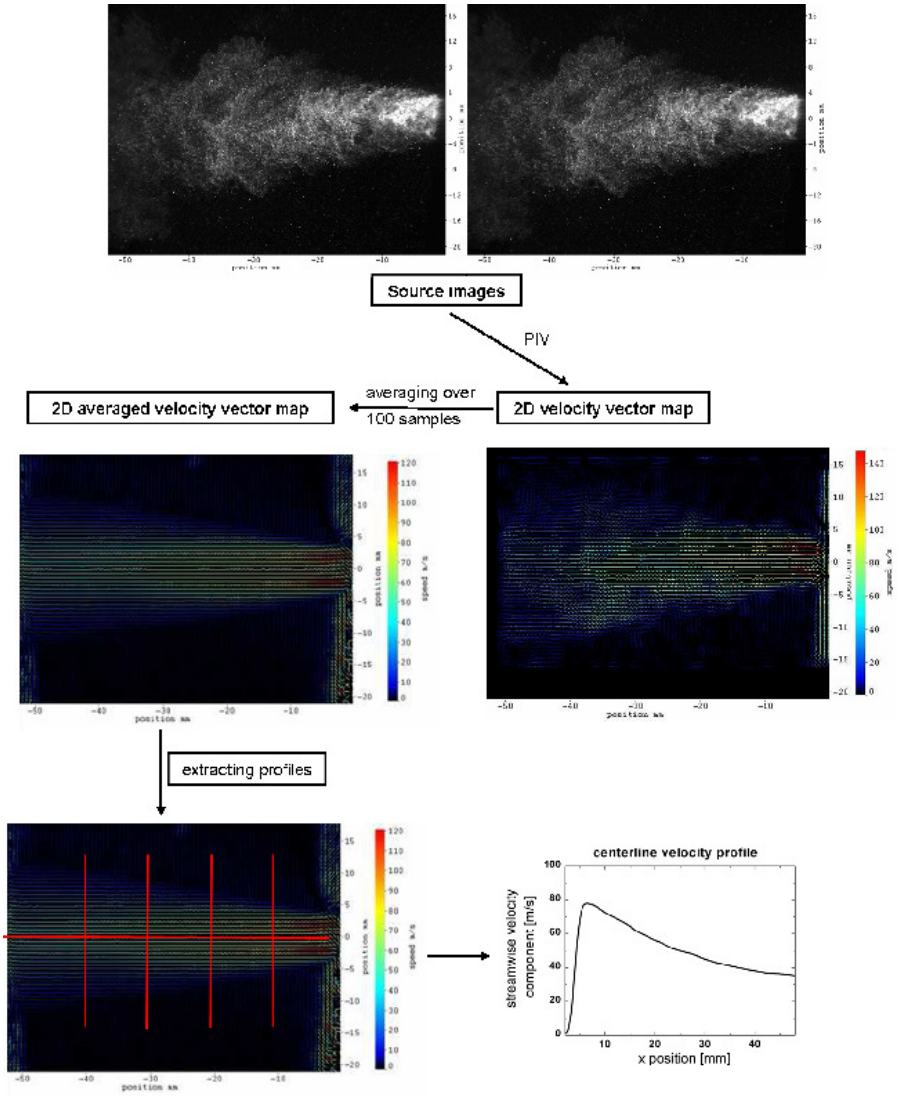
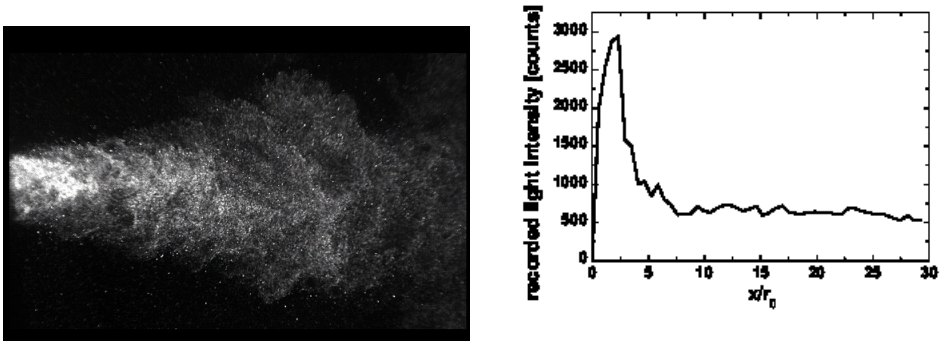


Figure 6.12.: Analysis of Results

show a source image and the respecting intensity distribution of the scattered light on the center-line profile. The intensity distribution is about four times higher near the nozzle exit indicating the overexposure effects. Thus, all velocity profiles are analyzed only downstream $x = 4 \cdot r_0$.

The final center-line velocity profiles of operation pressures 2, 4 and 7 bar are depicted in Figure 6.14. In all profiles a velocity decay is observed from $x = 4 \cdot r_0$ until



(a) (b)

Figure 6.13.: Source Image and Scattered light intensity over center-line

$x = 28 \cdot r_0$. By operating the valve with 2 bar, the velocity at $x = 4 \cdot r_0$ reaches up to 80 m/s and decays exponentially down to 30 m/s at $x = 28 \cdot r_0$. In the respecting radial profiles (see Figure 6.15) one can also detect the velocity decay at the center-line $y = 0 \cdot r_0$ with increasing x-coordinate. Additionally, the profiles at $x = 5 and 7.5 \cdot r_0$ are characterized by two peaks with increased velocities next to the center-line. This characteristic can be traced back to the specific outlet geometry (see Figure 6.3). As the laser sheet plane was positioned over 2 of the 4 kidney shaped orifices an increased velocity over the nozzle exits was expected. Additionally, one can observe a spreading of the jet, indicated by the radial extension of the profiles in stream-wise direction. The jet spreading characteristic can be prescribed using the so called jet half width $\delta_{0.5}$ (see Figure 6.18) which is defined by the radial position at which the velocity is dropped to half of the center-line velocity at the respecting stream-wise position. The half width starts at $x = 4 \cdot r_0$ with a value of $\delta_{0.5} = 3.5$ and ends at $x = 28 \cdot r_0$ with $\delta_{0.5} = 6.8$.

By increasing the operation pressure up to 4 bar, the maximum observed center-line velocity increases up to 210 m/s at $x = 4 \cdot r_0$ with an ongoing decay to 75 m/s at $x = 28 \cdot r_0$. In the respecting radial profiles depicted in Figure 6.16, the velocity peaks over the nozzle exits at $x = 5 and 7.5 \cdot r_0$ vanish, which can be traced back to the formation and break up of shock structures at the nozzle exits observed in the shadowgraph experiments. The shock as well as the higher velocity gradients in the shear layer lead to a slimmer jet spreading further upstream from $\delta_{0.5} = 2.2$ at $x = 4 \cdot r_0$ to $\delta_{0.5} = 6.3$ at $x = 28 \cdot r_0$ (see Figure 6.18).

The increasing jet velocities with higher operation pressures and the associated jet spreading further upstream are confirmed by the results provided by the measurements with 7 bar. The respecting center-line velocity decays from 310 to 150 m/s and the jet spreads from $\delta_{0.5} = 2.1$ at $x = 4 \cdot r_0$ with an exponential characteristic to $\delta_{0.5} = 5.2$ at $x = 28 \cdot r_0$.

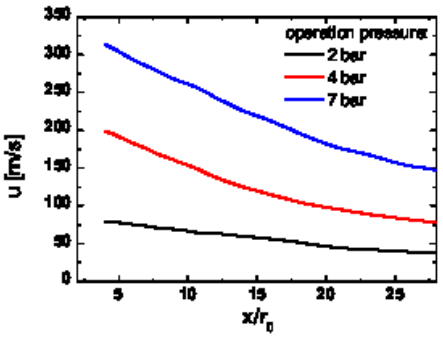


Figure 6.14.: Velocity profiles on jet center-line

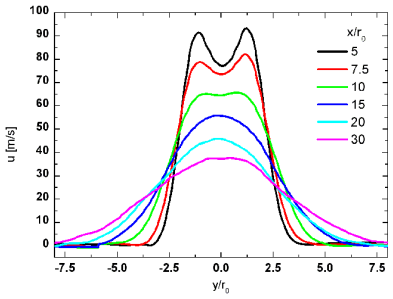


Figure 6.15.: Radial velocity profiles at 2bar

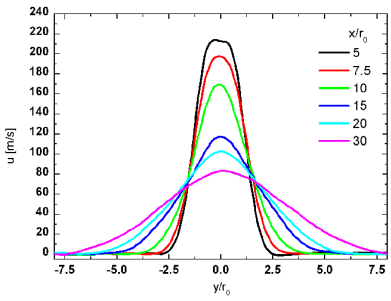


Figure 6.16.: Radial velocity profiles at 4bar

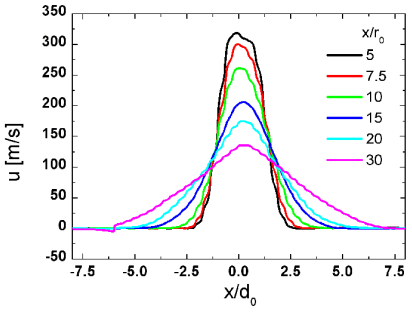


Figure 6.17.: Radial velocity profiles at 7bar

6.3.3. Acoustic measurements

Experimental setup

The acoustic measurements were performed with calibrated microphones at a sampling rate of 96000 Hz and a sensitivity range from 0-16kHz. For each measurement 5 microphones were positioned in circumferential direction around the nozzle exit at a radius of 20mm. In order to get validation data in a broader frequency range, at one position in circumferential direction an ultrasound microphone using a sampling rate of 192000 Hz in a sensitivity range from 0-60kHz was used.

Measurements are performed at propagation angles in between 20 to 90 degrees with respect to the jet axis (see Figure 6.19) by operating the nozzle with 2, 4 and 7 bar. At each configuration a total of 10 samples with recorded data over 5ms is used to perform a spectral analysis in the frequency range from 100Hz to 80kHz.

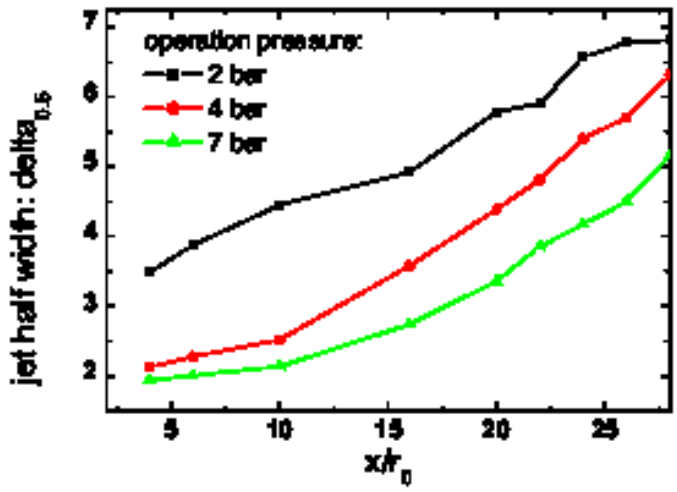


Figure 6.18.: Jet Half Width

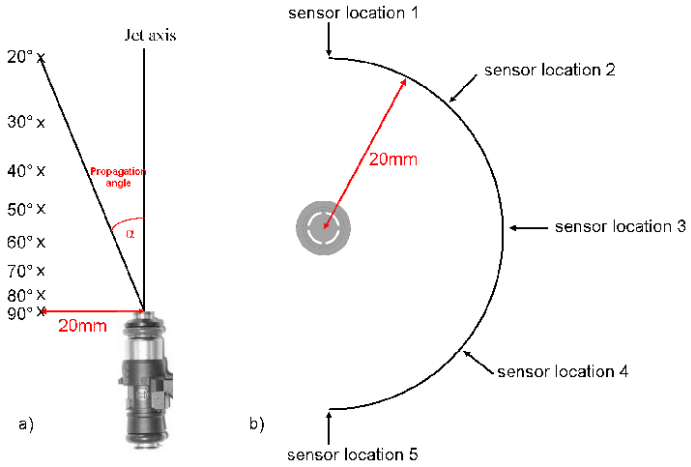


Figure 6.19.: Test setup for acoustic measurements: a)side view b)top view

Results

Figure 6.20 and Figure 6.21 show the narrow band- and third octave sound pressure level spectra for the observation points P1-4 at 20, 40, 60 and 80 degrees propagation

angle with an operation pressure of 7 bar.

In the spectrum of observation point P1 three major broadband peaks are noticeable. One occurs in the range from 1-4 kHz, the second between 10 and 30 kHz and the third between 50 and 70 kHz. Further upstream at observation point P2 the amplitude of the first broadband peak decreases and is shifted to higher frequencies. While the broadband peaks at higher frequencies disappear, the spectrum is dominated by high frequency noise where levels increase constantly from 10-80 kHz to a maximum of 112 dB. At the observation angle of 60 degrees in P3, the first broadband peak completely vanishes and sound pressure levels increase constantly from 93 dB at 1000 Hz to 97 dB at 10 kHz. Subsequently, a broadband peak with a maximum of 103 dB between 10.5 and 11.5 kHz appears and, finally, high frequency noise causes increasing levels up to 112 dB at 75 kHz. Radially from the nozzle, at observation point P4, the sound pressure level hardly changes above 9 kHz compared to the levels observed in P3. Up to this frequency, sound pressure levels are constantly 3dB lower.

As the main spectral characteristics are broadband, the respective third octave spectra provide all important information. Due to the averaging of levels over the third octave bands, the amplitudes have slightly increased but are not very sensitive to the frequency resolution of the used Fourier analysis. Additionally, the overall sound pressure level (OASPL) considering the whole frequency spectrum to is used to evaluate the noise intensity at different propagation angles resulting in the so called directivity plot depicted in Figure 6.22. Thus, all following acoustic analysis of experimental and simulation data consider the third octave spectra and overall sound pressure levels.

By reducing the operation pressure down to 2 and 4 bar, the OASPL at propagation angle of 20 degrees decreases from 128 dB to 120dB and 115dB, respectively. Accordingly, a reduction of sound pressure levels is also observed at all frequencies in the third octave spectra (see Figure 6.24). While the spectral characteristics hardly change, levels slightly decrease in the low frequency range in between 100 to 500Hz and a major reduction of levels over 10 dB at 4 bar and up to 20dB at 2 bar are observed at higher frequencies.

At propagation angles of 40 and 60 degrees the reduction in OASPL with decreasing operation pressure is much lower with 2-4 and 1-2dB respectively. Primarily, the reduction in sound pressure levels at both propagation angles is observed in the frequency range from 1kHz - 80kHz (see Figure 6.25 and Figure 6.26).

Finally, in lateral direction at propagation angle 80 degrees, a nearly constant reduction of 2dB and 3 dB can be detected over the whole frequency range by reducing the operation pressure to 4 and 2 bar. The final effect on the OASPL is lower with 1 and 1.5dB.

6.4. Simulations of Free-stream Configuration

6.4.1. Simulation Setup

The computational domain of the direct noise computation is shown in Figure 6.28. The nozzle exit geometry is centered in a cylindrical domain with radius $18r_o$ and

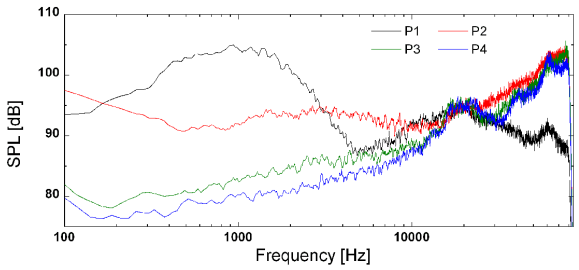


Figure 6.20.: SPL Spectra at 7bar

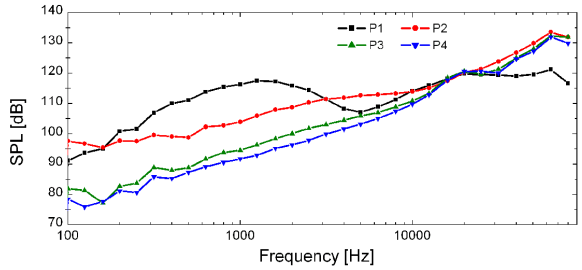


Figure 6.21.: Third octave spectra at 7bar

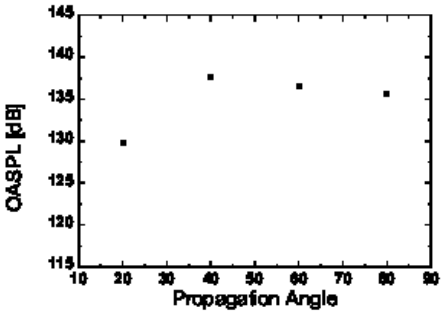


Figure 6.22.: OASPL directivity 7bar

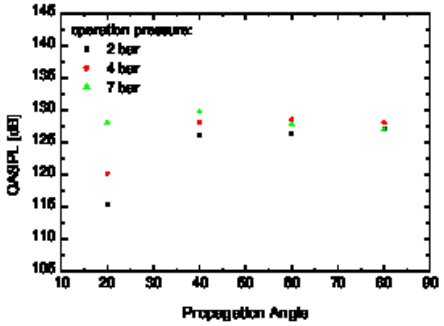


Figure 6.23.: OASPL for different operation pressures

spans from 0 to $35r_0$ in stream-wise direction. The fluid enters the domain at the inlet boundary with non-uniform velocity, pressure and density profiles obtained by a preceding steady state simulation of the whole valve geometry. The pressure history

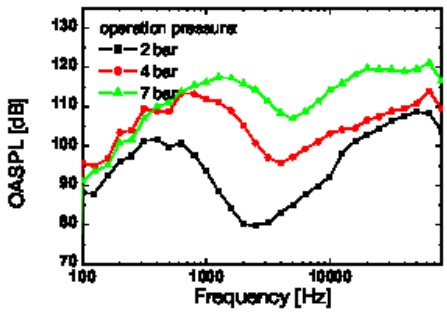


Figure 6.24.: SPL Spectra at 20 degrees

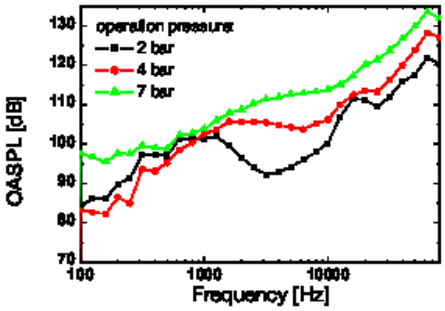


Figure 6.25.: Third octave spectra at 40 degrees

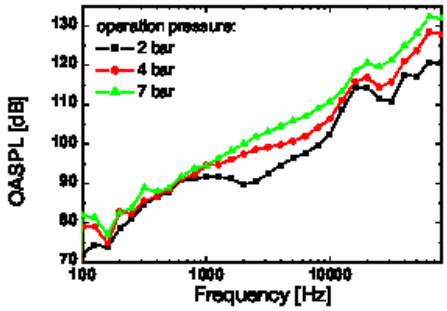


Figure 6.26.: Third octave spectra at 60 degrees

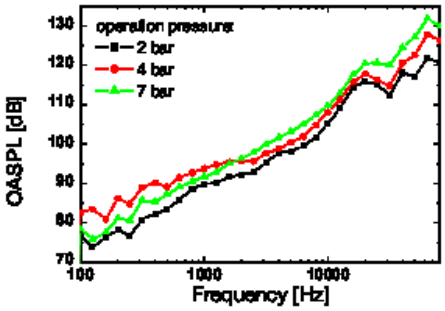


Figure 6.27.: Third octave spectra at 80 degrees

is recorded in a total of 40 observation points at radius $r = 12r_o$ from the jet axis at different propagation angles. Corresponding to P1-P4 shown in Figure 6.28, the points are located at angles of 20, 40, 60 and 80 degrees with respect to the jet axis. In order to accelerate statistical convergence, the data of each propagation angle are averaged over 10 positions distributed in azimuthal direction around the jet axis.

Numerical specifications

HALO:
The jet flow through the valve is computed by solving the compressible three dimensional Navier-Stokes equations with a Discontinuous Galerkin scheme of 3^{rd}, 4^{th} and 5^{th} order in space and a 3^{rd} order time integration. A fully explicit local time stepping algorithm proposed by Gassner et al.[9] is used. Advection fluxes are computed by the HLLC Riemann solver and viscous fluxes by a generalized Riemann solver [7, 23] . The

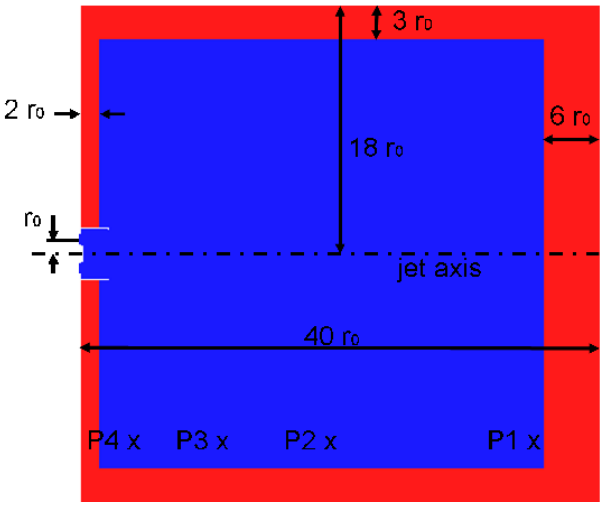


Figure 6.28.: Computational domain

fluid medium air is considered as ideal gas and the relationship between its dynamic viscosity and temperature is described by Sutherlands law.

The used grid consists 342,217 unstructured hexahedra, yielding a total of 3,422,170 degrees of freedom at 3$^{\text{rd}}$ order, 6,844,340 degrees of freedom at 4$^{\text{th}}$ order and 11,977,595 degrees of freedom at 5$^{\text{th}}$ order, respectively. As represented in Figure 6.29, the jet region is strongly refined to resolve the turbulent structures and shocks. The smallest cells have dimensions of $\Delta x = 0.01 r_0$. In radial as well as stream-wise direction the mesh spacing increases exponentially to a maximum of $\Delta x = r_0$ at the boundary starting at $r = 3r_0$ and $z = 8r_0$, respectively. For a proper representation of the geometry, all wall boundary elements are curved based on a reconstruction scheme proposed by Hindenlang [13] . In Figure 6.30 the influence of the reconstruction on the boundary elements is demonstrated using the geometry of a kidney shaped orifice at the inlet. The maximum frequency of sound waves resolved with the mesh was estimated by $f_{max} \simeq \frac{c_o}{10\text{ppw} \cdot \frac{\Delta x_{max}}{p}}$, where c_0 is the speed of sound, p the polynomial degree of the DG scheme and ppw stands for points per wavelength. This yields $f_{max} \approx 80kHz$ and $f_{max} \approx 100kHz$ for the 4$^{\text{th}}$ and 5$^{\text{th}}$ order simulation, respectively. The presented simulations run for $T_{sim} = 5.5ms$ where the last $5ms$ were used for averaging and data monitoring, allowing a reliable spectral analysis to a minimum frequency of $1kHz$. They were performed on 2048 cores on a Cray XE6 system at the High Performance Computing Center Stuttgart. The 4$^{\text{th}}$ and 5$^{\text{th}}$ order simulation required 38 and 81 hours, respectively.

Ansys CFX:

To compare results and simulation times with a commercial solver, a simulation model is also developed in Ansys CFX. Based on the work by Schoenrock [30] a direct aeroa-

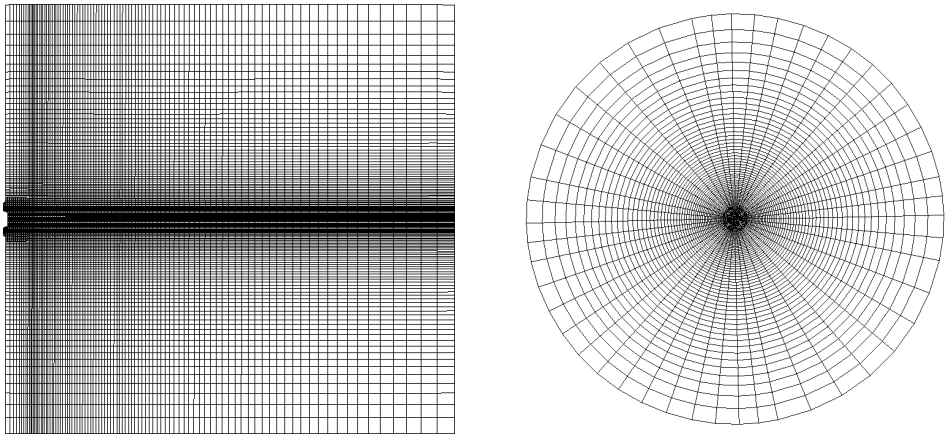


Figure 6.29.: Numerical grid: cut through A-A and B-B plane with respect to Figure 6.3

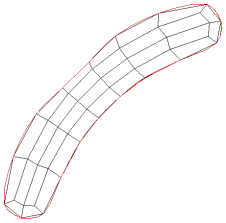


Figure 6.30.: Curved boundary elements at a kidney shaped orifice with (red) and without (black) reconstruction

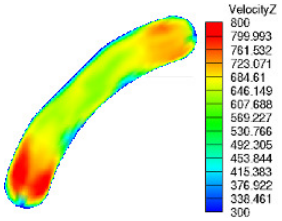


Figure 6.31.: Normal velocity profile at inlet

coustic simulation is performed using a static LES Smagorinsky model. The governing equations are solved with a Finite Volume method using the high resolution advection scheme and an implicit second order backward euler transient scheme with a fixed global timestep of $5 \cdot 10^{-8}$. The number of implicit subiterations for each timestep is limited to a maximum of 3. The computational domain is discretized by the grid prescribed above refined by a factor of 3 in all space dimensions yielding a total of 9,239,859 hexahedra (degrees of freedom).

Boundary Conditions

Nonreflecting boundary conditions:
In aeroacoutic simulations, non-reflecting boundary conditions are necessary since acoustic noise and aerodynamic fluctuations can produce spurious reflecting waves while leaving the computational domain [32, 2] . Several techniques were developed to

avoid this phenomenon, as reviewed by Tam [33] .

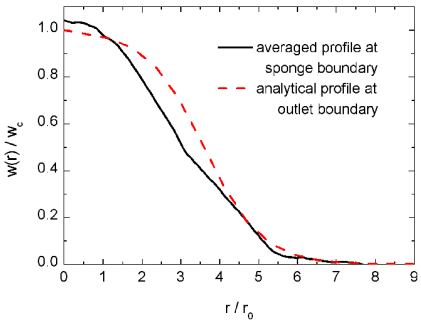
In terms of Ansys CFX the specialized non-reflecting boundaries for this testcase developed by Schoenrock [30] were used.

In the present simulations using the HALO code, an absorbing boundary condition based on explicit relaxation is chosen to minimize acoustic reflections. In this approach a relaxation term is substracted from the vector of conservative variables $\vec{U}$ in a sponge zone to damp reflections to a specific boundary state $\vec{U_0}$ (see e.g in Bodony [1]).

$$\vec{U}_{n+1} = \vec{U}_{n+1} - \underbrace{C\sigma(d) \cdot (\vec{U}_{n+1} - \vec{U_0})}_{\text{relaxation}} \qquad \sigma = (1 - \frac{d}{D})^{\exp} \tag{6.1}$$

The amplitude of relaxation σ within this sponge layer increases exponentially with decreasing distance d to the boundary, where sponge thickness D and exponential decay are given parameters. According to Mani [24] , an optimal sponge strength leading to the lowest reflectivity exists for any given problem. Thus, the sponge strength can be controlled by a constant C. For this application it was found to be optimal for $C = 0.01$.

At all free field boundaries, radiation boundary conditions are applied to damp the incident waves to ambient conditions with a sponge thickness of $D = 3r_0$. The outflow boundary condition state is defined by an analytical function providing a mean flow field of the jet (see Figure 6.32). All incoming fluctuations are damped in the outflow sponge zone with thickness $D = 6r_0$ to this specific state $\vec{U}_{0,Outlet}$.



$$\vec{U}_{0,Outlet} = \begin{pmatrix} \rho \\ u \\ v \\ w \\ p \end{pmatrix} = \begin{pmatrix} \rho(r) \\ 0 \\ 0 \\ w(r) \\ p(r) \end{pmatrix} \tag{6.2}$$

Figure 6.32.: Velocity profile at outlet boundary

Inlet boundary conditions:

The inlet boundary condition in the nozzle exit geometry is generated once by a steady state solution of the inner injector geometry. The primitive flow field variables are extracted on a plane within the kidney shaped orifices and interpolated on the grid of the aeroacoustical setup. In both codes (HALO and Ansys CFX) the interpolation method proposed by Schoenrock [30] is used. According to Figure 6.31, air ($\gamma = 1.4$,

$R = 287.058$) enters the domain with a normal velocity profile (with mean value $w = 505\frac{m}{s}$). The respecting pressure and density distributions over the offices with mean values of $244252Pa$ and $\rho = 1.97\frac{kg}{m^3}$ yield a mean Mach number of $Ma = 1.21$ and a mean temperature of $423K$.

Wall boundary conditions:
At the walls of the inlet duct slip wall boundaries are used. All other wall boundaries of the nozzle geometry are defined with no slip conditions.

6.4.2. Results and Validation

In order to validate the computational approach, experimental data from Schlieren optics, Particle Image Velocimetry and acoustic experiments were used (see 6.3). We analyzed the mean flow field characteristics, the capturing and formation of shock structures and the prediction of near field acoustics.

Flow Field Features

Flow development

Figure 6.33 displays the instantaneous mach number contour plots provided by all performed simulation in the B-B cut plane. The resulting flow fields show all a reasonable flow development with shock structures and the observed combination of the single jets after the break up of shocks. The growing instabilities in the shear layers on the inner radius produce a cluster of small scale turbulences in between the single jets. The rolling up of the respecting outer shear layers generates larger eddies resulting in a typical three dimensional turbulent mixing.

Nevertheless, the resulting flow fields differ concerning the resolution of turbulent structures and shock structures. As expected, the simulation with a 3^{rd} order DG scheme is only able to resolve the biggest turbulent structures. By increasing the polynomial order up to 4^{th} and 5^{th} order using the same grid, evermore small turbulent structures are captured by the numerical approach, which can be seen especially in the turbulence cluster between the single jets and in the shear layer development. The result provided by the simulation with Ansys CFX resolves turbulent structures which are smaller than observed with 3^{rd} order DG simulation but do not reach the resolution provided by 4^{th} and 5^{th} order. Based on these observations, the acoustics produced by turbulent motion are expected to be affected by the provided spatial resolution.

The positions of the shock structures in stream-wise direction can be compared with averaged Schlieren images (see Figure 6.34) where the density gradients in stream-wise direction are found in intensity minima and maxima. In the simulation results the density gradients were computed using the time averaged density distribution and visualized on a plane cutting the regions with the maximum shock cells.

One can see that all solutions provide a system of shock structures. While the downstream positions of the first four changes of the density gradient show good agreement with experiments using the 4^{th} and 5^{th} order DG scheme, the shock structures pro-

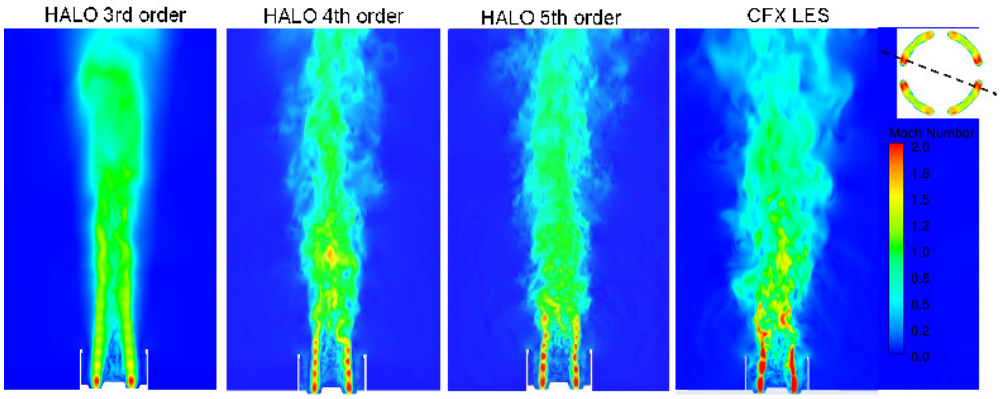


Figure 6.33.: Instantaneous mach number contours on A-A cut plane provided by simulations

vided by the 3rd order simulation are smaller and weaker and those provided by Ansys CFX are bigger.

From the Schlieren image, it can be estimated that the fifth shock lies within the region where the shear layer instabilities initiate the combination of the single jets. This process seems to be predicted well with the 5th order DG approach showing only weak stationary gradients in this region. The 3rd order and 4th order simulation result in the appearance of additional quasi stationary density gradients in the single jets indicating that the shear layer instabilities are not correctly resolved. Ansys CFX provides a break up of shocks at the adequate position, but caused by the prediction of bigger structures, the total number of shock cells is under-predicted.

In conclusion, the spatial resolution in this region strongly influence the prediction of present shock structures. As shock associated noise is a major sound source in supersonic jets, a correct prediction of these structures seems to be crucial to obtain reasonable acoustic results. In addition, due to the different positions where the shock structures break up, major deviations of flow field variables in this region are expected.

Mean Flow Field

The time averaged mach number contours provided by simulations are shown in Figure 6.35.

The velocity profile in the jet center-line provided by PIV decays nearly linearly in downstream direction from 320 m/s at $z = 5.88r_0$ to 150 m/s at $z = 30.0r_0$ (see Figure 6.36). Within this range the jet velocity decay is well predicted by the 5th order simulation. Further upstream, where no experimental data is available, the velocity increases to a maximum of 350 m/s at $z = 5r_0$ and then drops rapidly to negative values in the turbulence cluster indicating a back stream in this region. This development of the jet center-line velocity is also predicted by the 3rd and 4th order simulation, as

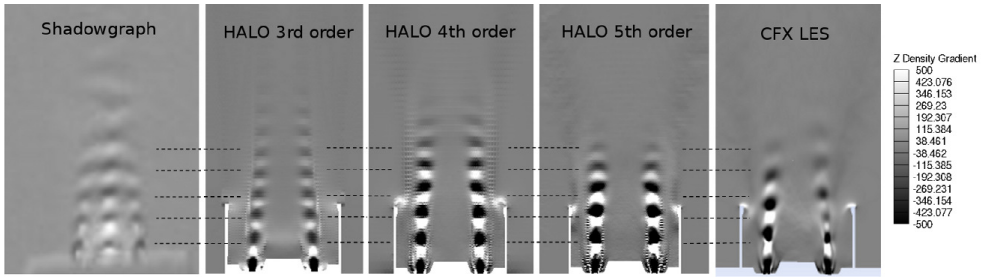


Figure 6.34.: averaged Shadowgraph image and contours of averaged density gradient provided by simulations

well as Ansys CFX. However, some fundamental deviations are observed. Firstly, the increasing velocity in the turbulence cluster is observed further downstream and ends up in a maximum of 350 m/s at $z = 10r_0$, 420 m/s at $z = 7.5r_0$ and 390 m/s at $z = 6r_0$, respectively. Secondly, the following velocity decay is not linear until $z/r_0 = 20$ with a 4^{th} order DG schme and $z/r_0 = 13$ with Ansys CFX, where the profiles approach experimental data and finally match further downstream. The velocity profile provided by a 3^{rd} shows additionally an over-prediction of the velocity within the decay region.

The respecting jet half width profiles are depicted in Figure 6.37. While the Jet spreading observed with the 5^{th} order DG simulation matches pretty well with experimental data, the simulated jet with Ansys CFX shows a stronger spreading characteristic, especially near the nozzle exit. The half widths provided by the 3^{rd} and 4^{th} order DG simulation increase further downstream, showing that the jets are slimmer over the whole computational domain.

The observed deviations are likely to be attributable to non-resolved small turbulence structures in the whole jet and the deviations in shock prediction near the nozzle exit. Due to the added numerical viscosity used to stabilize the numerics, the break up of shock structures and the development of turbulent structures is not represented correctly. Thus, the jet spreading is observed further downstream and the instabilities in the shear layers result in too large vortices with the 3^{rd} and 4^{th} order DG simulation leading to the over-prediction of the jet velocity in the particular regions. In Ansys CFX, the bigger shock structures breaking up in the right position lead to the slight over-prediction near the nozzle exits. The ongoing resolution of only large turbulent structures leads to the increased jet spreading with respect to experiments.

Acoustics

In Figure 6.38 one can see snapshots of instantaneous density contours provided by Ansys CFX and the 5^{th} order DG simulation. Additionally, the observation points P1 - P4 are depicted. The acoustic waves produced in the turbulent jet are clearly visible while spurious reflections at free field boundaries are not detected. The wave fronts originate mainly from the region where the shock structures over the four ori-

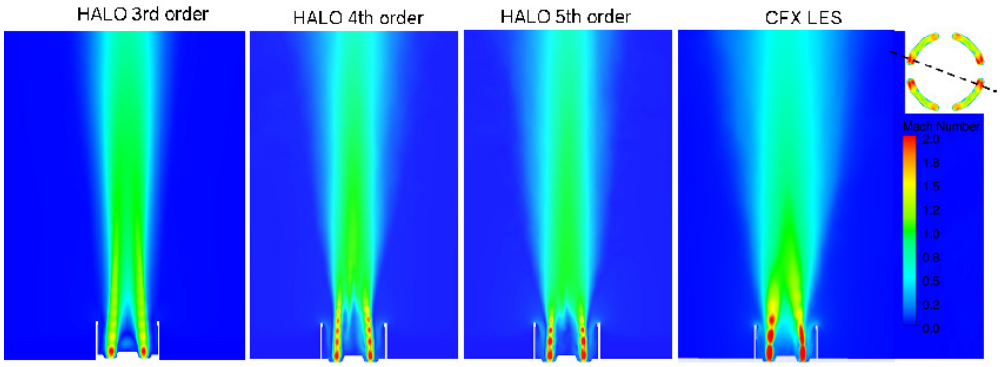


Figure 6.35.: Averaged velocity contours simulations

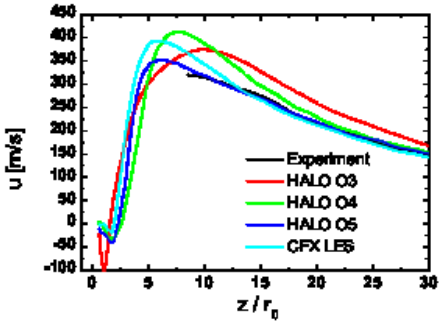


Figure 6.36.: Velocity profiles on jet center-line

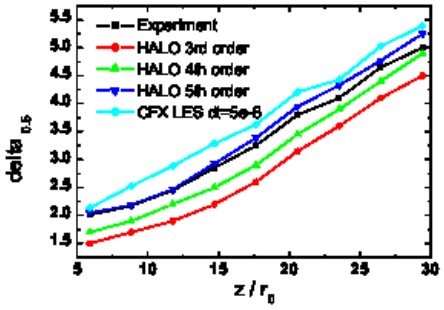


Figure 6.37.: Jet spreading characteristic

fices collapse into the turbulent jet flow at around $z = 5r_o$ indicating predominant sound sources. Additionally, a directivity pattern is visible showing a more intense propagation in downstream direction.

Figure 6.39 and Figure 6.40 compare the resulting OASPL at P1 - P4 with experimental data (see 6.3.3) taking into account the whole frequency range provided by the ultrasonic microphone up to 70kHz ($\rightarrow$ OASPL_{70}) and the audible frequency range (up to 20kHz $\rightarrow$ OASPL_{20}), respectively. In order to get a better understanding the respecting third octave spectra are depicted in Figure 6.41. As the simulation using Ansys CFX comprises data over only 1 ms, the spectra are cut beneath 5 kHz.

In observation point 1, the OASPL_{70} as well as OASPL_{20} is over-predicted by all simulations. While the minimal deviation is observed in the results of the 5^{th} order DG simulation with 3dB and 5dB, the OASPL provided by Ansys CFX and the 3^{rd}

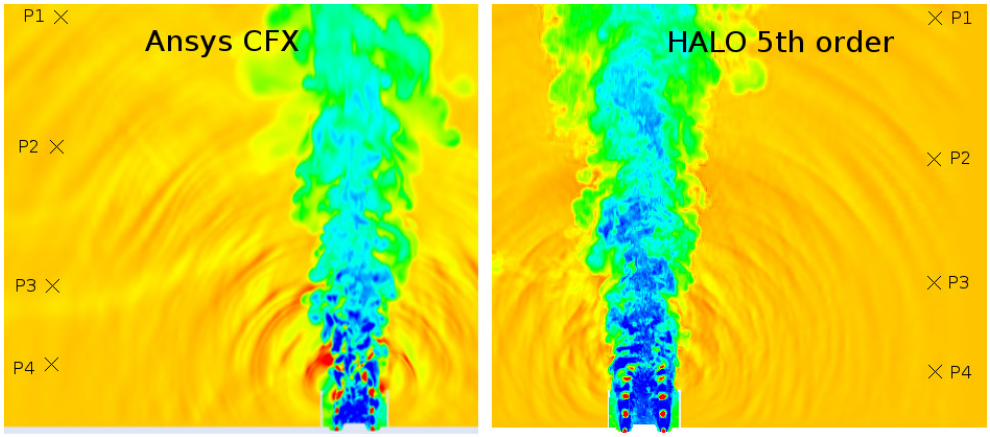


Figure 6.38.: Instantaneous snapshot of density contours (colormap: 1.1 - 1.25 kg/m^3)

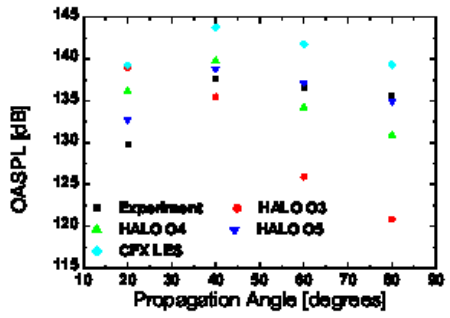


Figure 6.39.: Comparison of OASPL frequencies 1-60 kHz

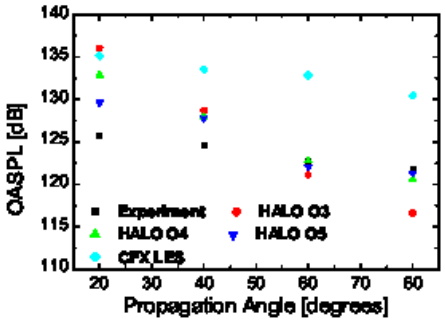


Figure 6.40.: Comparison of OASPL frequencies 1-20 kHz

order DG simulation over-predict experimental data up to 10dB. In the respecting third octave spectra, one can analyze in addition the frequency ranges in which the deviations take place. One can see that levels up to 2kHz are predicted only with slight deviations. The ongoing characteristical drop of sound pressure levels in between 2 and 20kHz observed in experiments is not well predicted by numerical approaches. Within this frequency range the levels observed in the 3^{rd} order DG solution are about 15dB higher. Only in the high frequency range levels slightly approach experimental data and end in an over-prediction of 4 dB at 70 kHz. The higher resolution of physics with 4^{th} and 5^{th} polynomial order in space leads to improved results with maximal deviation of 10 and 8dB at 5 kHz. Especially results of the 5^{th} order approach match

quite well with experiments in the high frequency range over 20kHz, leading to the reduction of OASPL taking into account the frequencies up to 70kHz. Ansys CFX provides also over-predicted sound pressure levels, which are nearly constantly 10dB higher with respect to experiments over the whole frequency range, leading to 10dB higher OASPL_{70} and OASPL_{20} in both evaluations.

In observation point 2, the $OASPL_{20}$ in the audible range observed with 125 dB in experiments is as well over-predicted with all simulations. While Ansys CFX over-predicts the level by 10 dB, all HALO results provide a better value of about 128dB. In the respecting third octave spectra, one can see that the increase of sound pressure level with frequency is captured by simulations and the slight over-prediction of levels starts at about 5kHz and stays 3dB higher than experimental data up to 12 kHz. At higher frequencies up to 70kHz Ansys CFX over-predicts levels by 10dB. While the results provided by the 4^{th} and 5^{th} order DG simulation keep the 3dB distance to experimental data the spectrum calculated with 3^{rd} order DG simulation data starts to drop under experimental data starting at 40 kHz. The latter observations lead to an over-prediction of the $OASPL_{70}$ by Ansys CFX, HALO 4^{th} order and HALO 5^{th} order simulation and finally a 2dB smaller value with a 3^{rd} order DG approach.

In observation point P3 the OASPL_{70} and OASPL_{20} provided by the 5^{th} order DG simulation match well with experimental data showing a deviation smaller than 1dB. While these levels are under-predicted with 10dB and 5dB using a 3^{rd} order DG approach, the results provided by the 4^{th} order DG simulation match in OASPL_{20} and show only a slight under-prediction of OASPL_{70} by 2dB. Finally, Ansys CFX shows increased levels by 4dB in OASPL_{20} and 8dB in OASPL_{70}. In the respecting third octave spectra, one can see that levels provided with the HALO solver also match well in the frequency range up to 12kHz. Then, the spectra of the 3^{rd} order simulation and further starting at 40 kHz the spectra of the 4^{th} order simulation start to drop beneath the levels observed in experiments. One again, the spectra obtained by the pressure history recorded in the Ansys CFX simulation result in sound pressure levels 10dB higher with respect to experiments nearly over the whole frequency spectrum from 5-30kHz. Only over 40kHz the deviations slowly decrease.

Finally, in observation point P4 the deviations in OASPL_{70}, OASPL_{20} and the third octave spectra of numerical and experimental results are similar than those observed in P3. It is apparent, that the spectra of the 3^{rd} and 4^{th} order DG simulation start to drop earlier beneath the levels observed in experiments starting at 8 and 12kHz, respectively. This results in a stronger under-prediction of OASPL_{70} and OASPL_{20} than observed in P3.

Based on these observations, two effects are likely to be responsible for the deviations in predicted acoustics by the HALO code to those provided by experiments. First, the solutions provided by the 3^{rd} and 4^{th} order DG approach do not capture all noise generating effects associated with turbulent mixing. Due to the high numerical viscosity which has to be added to stabilize the numerics, the small scale turbulences can not be resolved and too large turbulence structures are predicted. Thus, lower sound pressure levels are observed in the high frequency range. As turbulent mixing noise is primarily propagated in radial directions, this effects is more intense in P3 and especially in P4. Secondly, the observed prediction of too high jet velocities and deviations of shock

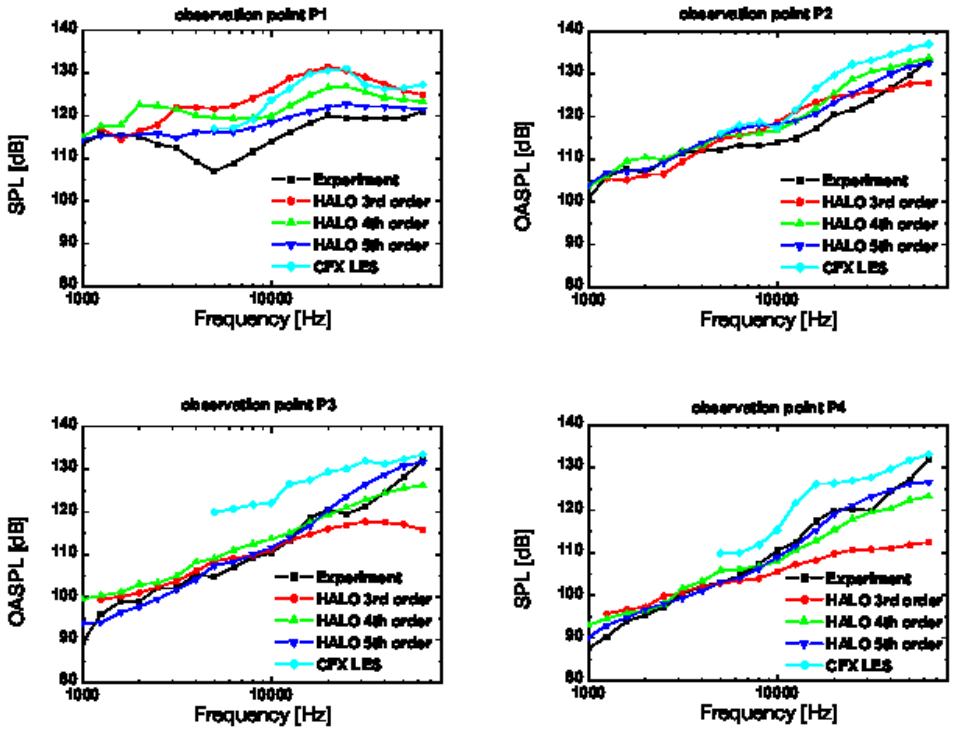


Figure 6.41.: Comparison of third octave spectra at observation points P1-P4

cell structures cause additional sound sources producing low frequency acoustics which are mainly propagated in downstream direction. In addition, the model of the inlet boundary does not cover any fluctuations of the operation pressure, which are likely to be present in experiments, affecting the shock structure motion and production of related acoustics. Thus, in observation point P1 the highest deviations with respect to experiments are observed. These issues can be improved considerably by the use of a 5^{th} order DG approach. Results match well with experimental data at P3 and P4 and the deviations at P1 and P2 are within a acceptable confidential interval of 5dB.

The direct noise simulation with Ansys CFX results in all observation points in sound pressure levels increased by about 10dB over the whole frequency range. In addition to the effects explained in the preceding paragraph, errors in the transport of acoustical waves are supposed to be responsible for the over-prediction of acoustics. This issue can only be overcome by a successive grid refinement resulting in an immense increase of computational costs.

6.4.3. Efficiency and Assessment of Turn-Around-Time

The overall Turn-Around-Time from scratch to a numerical result consists 4 major steps:

- Geometry and Grid Generation
- Pre-Processing
- Solution of Governing Equations
- and Post-Processing

As we are interested in reducing the overall Turn-Around-Times by using the STEDG solver HALO, the required effort for each step is evaluated and compared to the generation of a simulation model based on Ansys CFX with LES Smagorinsky turbulence model.

Geometry and Grid Generation: In industrial simulation models like the gas injection process, the relevant geometries are more or less complex. In terms of typically studies like Design of Experiments (DoE) the parameters of the design slightly change. In commercial CAD tools (Catia, ProE, etc.) as well as integrated tools like Ansys Workbench an automated and parameterized geometry generation is provided. Thus, only the initial design requires an increased effort (typically 1-3 days) by the user. All subsequent changes can be performed in a few minutes by changing the parameterized values.

The following grid generation mechanism is a little more complex. As the spatial resolution of the used grid has to take into account the different turbulent scales in different flow regions, the grid generation mechanism is iterated with each simulation solution until the required accuracy is reached. As LES simulations require a high mesh quality, this procedure can take some days up to some weeks dependent on the problem size. After the initial mesh is found, the grid generation can also be automated based on parameterized variables as long as the impelled variations hardly change the flow physics. If they do so, the grid has to be adapted again by the user in some iterations.

Both, Ansys CFX as well as the HALO code are able to read different mesh formats (e.g. CGNS), so that different mesh generation tools like Ansys ICEM or Ansa can be used. In all these tools, the import of common geometry formats like *.stl and *.igs is provided allowing a workflow in which the user effort for geometry and grid generation is comparable for both tools. But a major advantage of the STEDG solver HALO is the possibility to use polynomials of higher order to approximate the solution in each cell. Instead of a complex, iterated, time consuming grid refinement one can adjust the solution order in regions with high flow dynamics user-defined or automated by order adaption.

Pre-Processing: In the preprocessing step, the flow physics, boundary conditions and numerical preferences are defined. In Ansys CFX a graphic user interface provides an easy and user friendly input. All settings are finally written in a definition file, which

can be converted in a text file, changed by a special user script using a text editor and reconverted into a definition file.

All preprocessing information required by the HALO code are set in a setup text file containing all information concerning the numerical preferences and an initial text file containing all information concerning the flow physics. The setup file is read before the code is compiled for a specific problem and a recompilation is only needed if changes in the setup file are made. The initial file is read by the code in the beginning of each computation. Both files can be easily changed in a text editor or using a script.

Thus, the preprocessing step of both approaches should not take more than a few hours and can easily be automated in terms of a DoE or optimization study.

Solution of Governing Equations: The solution of the governing equations is the most time consuming step in LES calculations and has to be performed for each design in DoE/optimization studies. Thus, the required time to calculate the results is crucial and a reduction would shorten the overall required time of major studies drastically. It depends on:

- numerical scheme
- number of degrees of freedom in space (DoF)
- number of calculated updates for each DoF
- available computational resources
- available software licenses
- scalability of the software

The computational effort is predominantly determined by the overall performed updates of the present degrees of freedom. Thus, the computational effort increases with increasing number of grid elements and increasing order in space of the numerical scheme. Based on the used numerical scheme, the number of degrees of freedom and performed updates is different. In Ansys CFX the degrees of freedom is equivalent to the number of grid elements and the performed updates are calculated by the overall number of time steps and striations in the used implicit time integration scheme. As the HALO code uses an fully explicit local time stepping the number of updates agrees with the total number of performed time steps. The degrees of freedom in space are calculated by adding all DoFs of each grid cell i dependent on the used polynomial order p:

$$DoF = \sum_i \frac{p_i \cdot (p_i + 1) \cdot (p_i + 2)}{6} \tag{6.3}$$

Additionally, the sets of equations and algorithms to calculate the DOF updates differ in the schemes, which leads to different calculation times. Thus, the computational effort is measured by the overall needed CPU time to perform a full simulation.

As the computational effort to achieve a high solution quality of the presented testcase is of about 4000 Core days, a calculation on one CPU is not effective. Thus, the physical simulation time can be reduced by parallelization on HPC Cluster systems. However, the parallelization is limited by the number of available resources, available HPC software licenses (especially in terms of Ansys CFX) and on the scalability of the software executables.

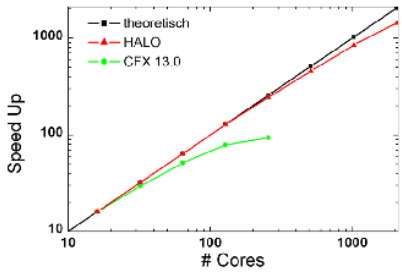


Figure 6.42.: Speed Up of software executables on Cray XE 6

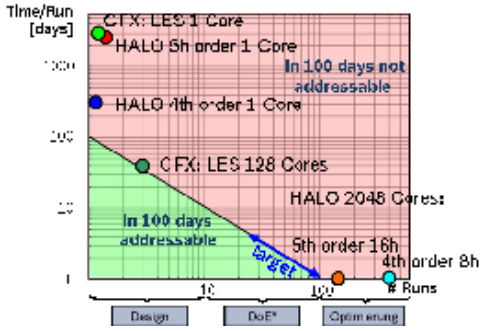


Figure 6.43.: Comparison of calculation times of different simulations

Our investigations show that the overall computational effort can be reduced with the STEDG code by a factor of about 2-5 with a comparable solution quality using a 4th and 3rd polynomial order approach, respectively. Additionally, an immense improvement of the solution quality is provided with an approach of 5th polynomial order in space which requires the same computational effort than an LES performed with Ansys CFX.

An outstanding speed up of the physical simulation time can be acquired by the effective parallelization of the HALO code. In Figure 6.42 the scalability plot shows a very good performance up to 2048 cores while Ansys CFX only can be used effectively up to 128 Cores. Thus, assuming an availability of up to 32768 Cores at once (e.g. at the Cray XE6 system on the HLRS) a total of 16 designs can be computed in 1 day using the HALO code, which is not limited with licenses up to now. The poor scalability and the additional restriction of available software licenses in Ansys CFX (ca. 256 on the Cray XE6 at once) prevents the calculation of multiple designs at once in a short time.

Post-Processing: The required time for the post-processing of the flow field solutions provided by Ansys CFX and HALO are comparable as they can be read by common tools like CFX-Post, Tecplot or Ensight. Transient results containing special

information over the whole simulation time can be exported by both codes into different formats (e.g. CGNS, text) that can be post-processed with Excel, Matlab, etc. All these tools provide a scripted batch post-processing reducing the effort in terms of DoE and optimization studies.

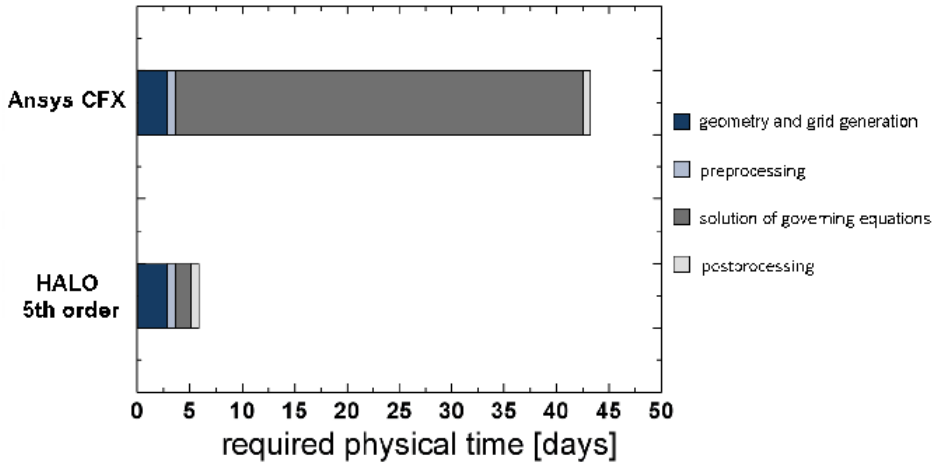


Figure 6.44.: Turn around times for one calculation

6.5. Simulations of Parameter and Geometry Variations in Firestorm Configuration

6.5.1. Variation of operating pressure

Based on the results of chapter 6.4.2 showing that the fluid flow and acoustics of the audible frequency range are accurately predicted with a 5^{th} order DG simulation, the simulation setups with operation pressures 2 and 4 bar are solved with these code settings. In Figure 6.45 the resulting center-line velocity profiles are compared with PIV results. Both velocity profiles are predicted well with deviations smaller than 5% concerning experimental data.

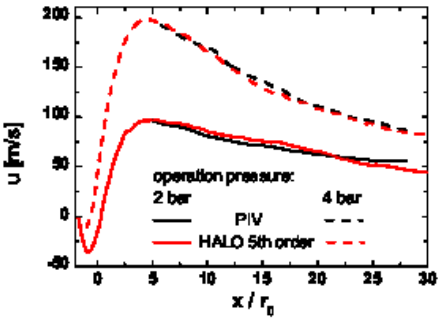


Figure 6.45.: Comparison of center-line velocity profiles

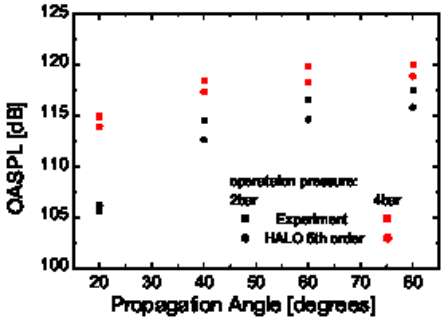


Figure 6.46.: Comparison of overall sound pressure levels

The respecting evaluation of the predicted acoustics is also very promising. For both operation pressures the directivty pattern of the OASPL in the acoustic frequency range is predicted within a confidence interval of 2dB. In Figure 6.47 and Figure 6.47 the third octave sound pressure level spectra are depicted for propagation angles of 20 degree (P1) and 80 degree (P4). One can see that the levels in P4 are predicted very well and a slight over-prediction of levels at lower frequencies in P1 exist. The deviations of spectra in P2 and P3 resemble to those observed with 7 bar operation pressure supporting the conclusions made in 6.4.2.

In conclusion these results show, that the simulation model is able to accurately predict the acoustics under variations of parameters within a confidence interval of 2 dB. Thus the model is considered to be validated and can be used to evaluate geometry variations which could reduce the sound production and emission.

6.5.2. Variation of silencer length

In order to evaluate first measures to reduce the sound production during the compressed natural gas injection, the length of the silencer geometry was extended from 3.7

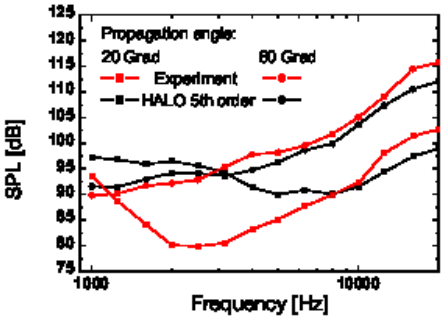


Figure 6.47.: Third octave spectra with 2 bar operation pressure

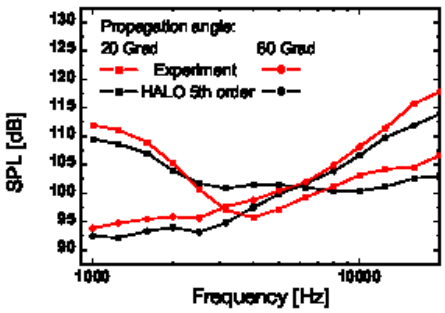


Figure 6.48.: Third octave spectra with 4 bar operation pressure

to 5 and 10mm. Figure 6.49 and Figure 6.50 show snapshots of the respecting mach number contours (colormap 0-2) and pressure contours (colomap 99500-100500Pa). One can see in the mach number distribution that the evolution of the supersonic jet is not noticeably affected as the shock structures and the combination into the turbulent jet do not change much in their position. In the respecting instantaneous pressure distribution it is obvious that the propagation of sound waves starts right above the end of the silencer. The directivity of the sound propagation seems to be slightly affected by the prolongation of the silencer, as the propagation characteristic with a silencer length of 10mm seems to be shifted from downstream to radial directions by about 20 degrees.

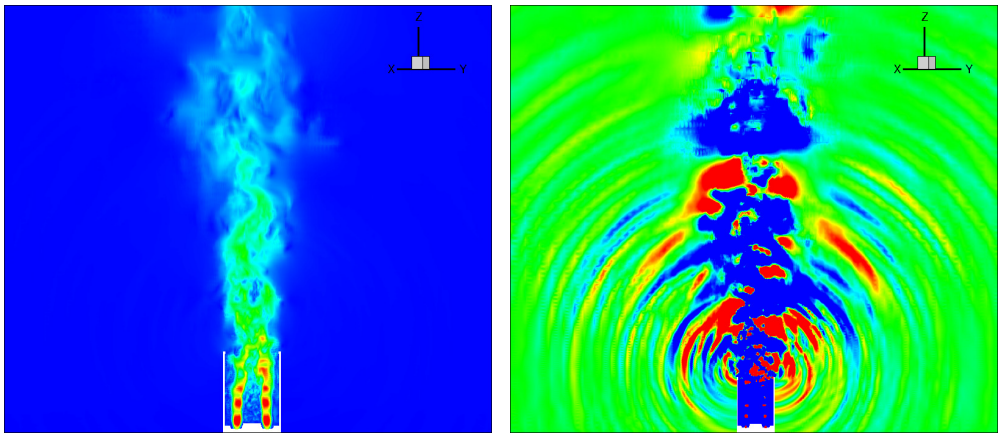


Figure 6.49.: Mach and Pressure Contours with silencer length 5mm

The observed shift can also be detected in the OASPL levels at different observation

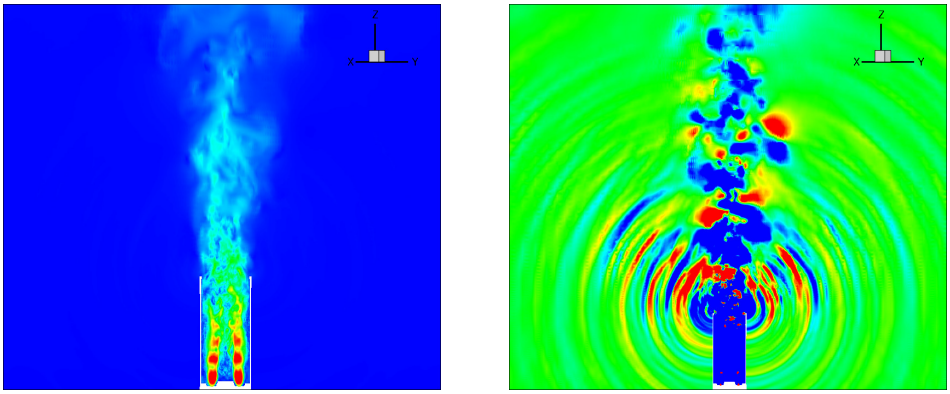


Figure 6.50.: Mach and Pressure Contours with silencer length 10mm

points which are compared with acoutics calculated with the original silencer length in Figure 6.5.2. Although a slight reduction of 2dB at 20 degree propagation angle can be detected, at all other observation points the levels are increased by the extension of the silencer geometry. In the third octave spectra depicted in Figure 6.52 one can see that the sound pressure levels in the low frequency range are in fact reduced at all observation points with increasing silencer length but simultaneously the levels in the high frequency range are increased. As result, the extension of the silencer geometry is not able to reduce the sound emission effectively. Nevertheless, we can conclude that the simulation model is able to evaluate different geometries concerning the propagated acoustics which enables further studies to detect sound reducing measures.

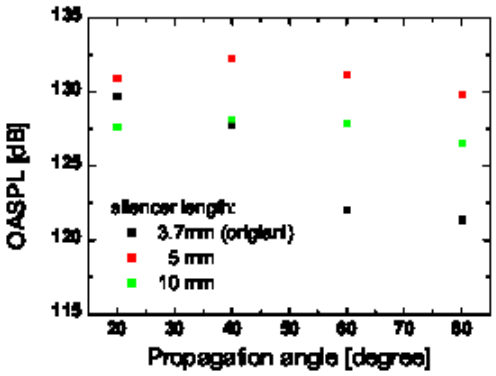


Figure 6.51.: Comparison of overall sound pressure levels

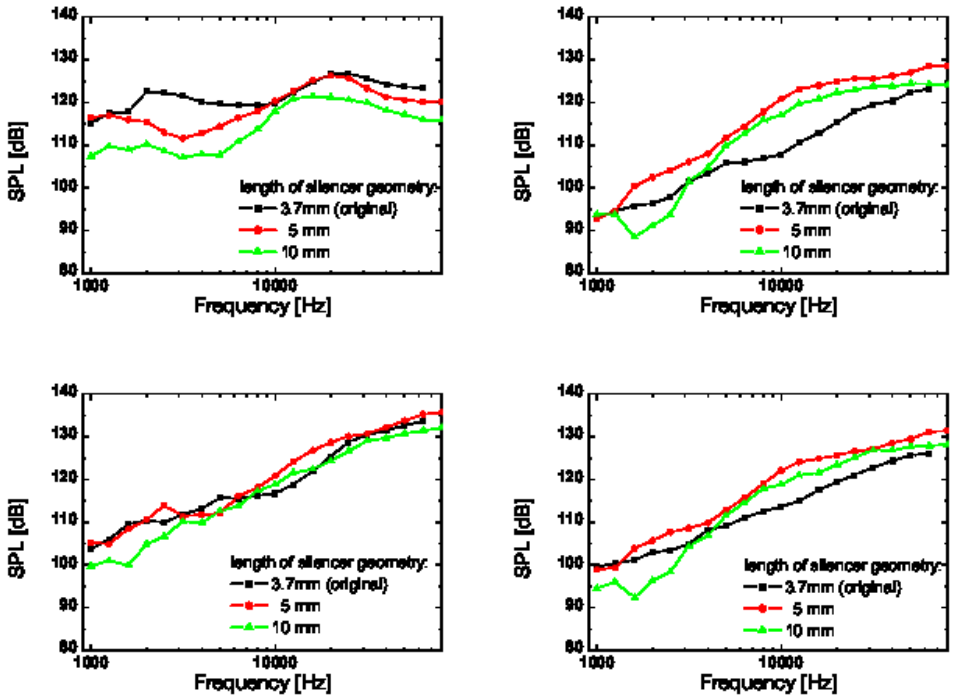


Figure 6.52.: Comparison of third octave spectra at observation points P1 and P4

6.6. Simulations of Intake Manifold Configuration

We finally transferred the HALO code settings identified in the free-stream configuration simulations to the intake manifold configuration to identify its applicability in this complex setup. The simulation model consists an actual geometry of an intake manifold with a mounted CNG injection nozzle. The subsequent combustion room with the intake valve are not yet considered in this model. The inlet and outlet boundary conditions of the duct are defined with ambient conditions. As inlet boundary of the injection nozzle, the specialized model described in 6.4.1 is used and all wall boundaries are defined by no slip conditions. The direct aeroacsoutic calculation was performed with a 4^{th} order DG scheme using a grid with 344833 elements and run about 70h on 2048 Cores to simulate a total of 10ms physical time.

In Figure 6.53 the instantaneous mach number distribution on a cut plane in the center of the intake manifold duct after 3ms and 8ms simulation time are depicted, respectively. One can easily detect the mounting of the injection nozzle at the manifold geometry where the supersonic jet enters the duct and mixes with the surrounding fluid after the break up of shock structures. The produced acoustics can also be visualized

by the instantaneous density and pressure distribution as shown in Figure 6.54 and Figure 6.55. In these pictures one can observe that the produced sound waves first propagate into the duct and are reflected back by its walls.

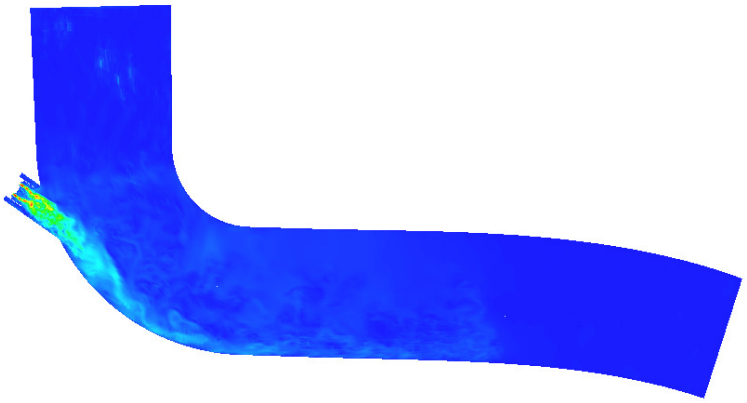


Figure 6.53.: Instantaneous Mach number contours 0-2

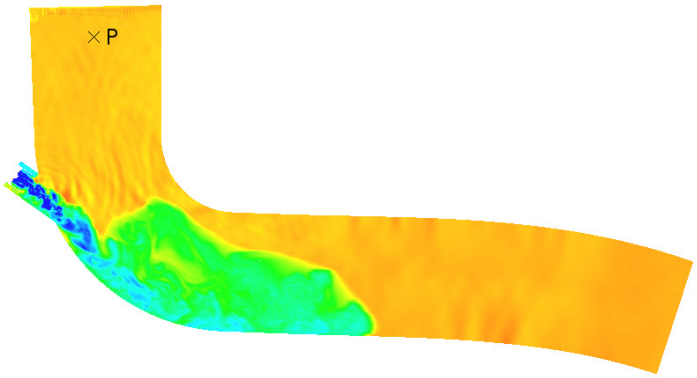


Figure 6.54.: Instantaneous density contours (0.8-1.3 $kg/m3$

As a consequence of the interference effects, acoustical duct modes establish and result in increased sound pressure levels at discrete frequencies in the respecting sound pressure level spectra. For example Figure 6.56 shows a narrow band spectra of the pressure fluctuations recorded at observation point P near the duct inlet which is dominated by discrete peaks at about 5.6, 12.0, 16, 21.2, 31.2, 41.3, 49.8 and 63.5 kHz. The cut on frequencies of the present duct f^c_{mn} can be estimated with a state of the art duct mode theory are found at $f^c_{00} = 11.86kHz$, $f^c_{01} = 21.6kHz$, $f^c_{02} = 31.5kHz$,

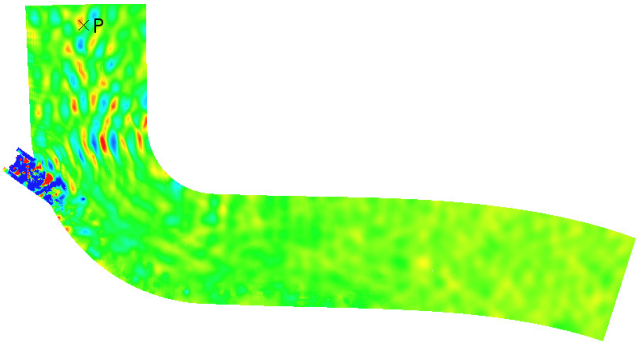


Figure 6.55.: Instantaneous pressure contours (95000-105000Pa

$f^c_{03} = 41.2kHz$, $f^c_{04} = 50.1kHz$ and $f^c_{10} = 5.6kHz$, $f^c_{11} = 16.5kHz$ which correspond quite well with the observed peaks in the frequency spectrum.

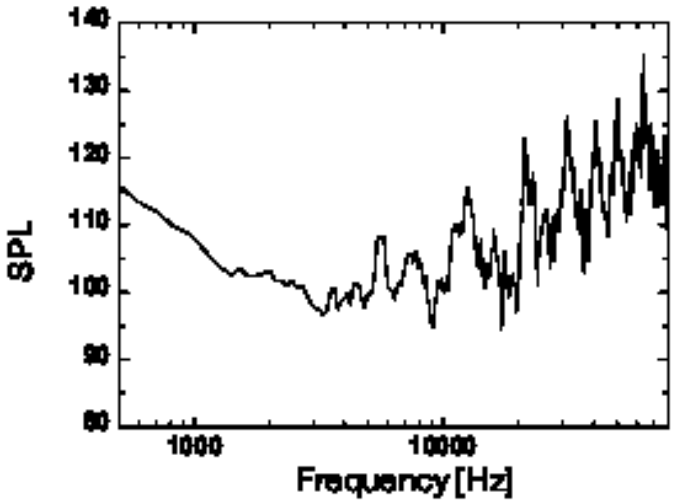


Figure 6.56.: SPL Spectrum recorded at observation point P

In conclusion, an accurate direct aeroacoustic simulation of the gas injection process could be performed for the first time in an acceptable turn around time of 3 days. Thus, the STEDG solver is a promising tool to perform further studies evaluating sound reduction strategies with different nozzle mountings and duct geometries.

6.7. Further Utilization of Results

The acquired results in the STEDG project will be used in further research and product development activities concerning gas powered engines (CNG, Biogas and hydrogen). Despite increasing developments in the context of electric vehicles, gas vehicles remain a promising alternative to strikingly reduce CO_2 emissions and to make use of renewable energies. The utilization within the Bosch company is segmented in a scientific, technical and economic part:

6.7.1. scientific utilization

First, the knowledge about the requirements, constraints and performance of the STEDG scheme in realistic industrial applications and conditions acquired through the numerous simulation results allows an ensured evaluation of its future application potential in the Bosch Company. Compared to the commercial tool Ansys CFX the efficient parallelized STEDG scheme enables a speed up of calculation time by a factor of 100 on HPC clusters with more than 1000 Cores. Thus, a further expansion of our in house cluster is reasonable to provide the resources for new developed, parallelized software tools.

Secondly, the technical results acquired in the project were presented in June 2012 to the scientific community on the 18th AIAA/CEAS Aeroacoustics Conference in Colorado Springs (USA, CO) in terms of a presentation and a technical paper called "Direct Aeroacoustic Simulation of near field noise during a gas injection process with a Discontinuous Galerkin Approach". The participation in the conference enabled Bosch to contact leading scientists in the research area of aeroacsoutics and provided insight into the current state of research.

6.7.2. technical utilization

The technical utilization includes four parts:

- Expert knowledge in the application of HPC applications on cluster systems of the High Performance Computing Centers: During the project, simulation studies commonly used in the product development process were defined, implemented and evaluated with the research code HALO. Based on the elaborated and finally established workflow, further simulation studies can be performed or delegated.

- Know How of porting parallelized (research) code on the Bosch cluster system: As the research code HALO could be ported on the in house cluster and successfully tested on smaller test-cases, the future usage of the HALO code is ensured. Currently, the cluster provides about 2300 cores while a maximum of 300 cores can be requested by one user at once. Thus, the simulation of smaller test-cases (e.g. transient effects in injection nozzles with inviscid flows) will be possible. The future extension of the provided resources and the expert knowledge acquired during the STEDG project will also enable simulation studies with bigger

and complex test-cases like the presented aeroacoustics during a gas injection process.

- Technical understanding of the sound producing mechanisms and sound propagation during gas injection: The data provided by numerical and experimental evaluations of the aeroacoustics during the gas injection process enabled a detailed insight into the sound production mechanisms and propagation characteristics. For example, we could analyze that the interaction of the supersonic jet with the silencer geometry leads to a distinct sound emission in the high frequency range. With selective variations of physical (operation pressure) and geometrical parameters (length of silencer duct), the manipulation and suppression of sound emission could be tested by numerical approaches. These results are an initial basis for further effective sound reduction measures.

- A long-term usage of the HALO code in context of our corporate research and product development in business units, formulated in the project application with accordance to an outstanding project success, would be desirable. Especially the promising performance of the code in comparison with commercial tools engages great interest. However, the intended long term use can not be ensured from a present perspective caused by the nonexistence of a realizable and affordable concept of source code maintenance. Nevertheless, the ongoing activities at the Institute of Aerodynamics and Gas dynamics at the University of Stuttgart, where the results of the project are used to further develop the DG Numerics, are watched with great interest. Thus, it is rather likely that the results acquired during the STEDG project will be reused later on.

6.7.3. economic utilization

The simulation studies concerning the aeroacoustics of the gas injection process performed on the High Performance Computing Centers would not have been possible with the in house cluster. Thus, the costs of trial product samples, experimental studies or delegated simulation studies could be avoided. Additionally, the acquired knowledge about the sound production mechanisms could result in future nozzle designs. Especially noise reducing design variations resulting in minor additional costs in the production process are favored in the business Unit Gasoline Systems (GS) and would raise the competitive advantage. By the end of the project such measures could not be identified, yet.

7. Laser Cutting Device

As a leading supplier of laser cutting machine, TRUMPF was able, in the past, to set itself apart from its competitors through its high cutting speeds. This was achieved primarily due to the development of ever more powerful cutting lasers. In doing so the cut quality played only a subordinate role, as long as there was no burr formation on the underside of the sheet. There have been more and more indications recently that increases in laser power can only be converted into increases in feed rate to a certain degree. At the same time the quality of the cutting edge is becoming increasingly important. The cut quality does not depend only on the beam quality of the laser, but also on the nature of the gas flow with which the molten metal is blown out of the kerf. The extent to which the nature of the flow, determined by the design of the cutting nozzle and the cutting gas pressure used in operation, influences the cutting quality is still today not completely understood. A better understanding of the process is expected through the use of a cutting gas flow simulation which encompasses the complete area from the nozzle right through the kerf to the area below the kerf. These transient calculations were not possible in the necessary quality until the beginning of the BMBF project STEDG.

7.1. Experiments

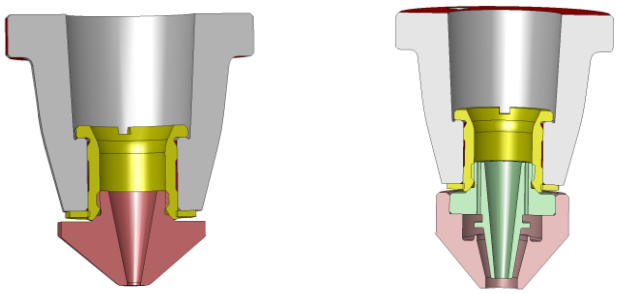
7.1.1. Cutting nozzles examined

The formation of the cutting gas flow and the related cut quality are influenced by, among other things, the nozzle geometry. In the project, the hole nozzle EAA $\varnothing 2.3mm$ (TRUMPF Mat. no. 1324866), introduced as series part, and the NSD 12 (bypass flow nozzle) which is still a test piece were examined. Figure 7.1 shows sectional views of both nozzles.

The hole nozzle is characterized by a truncated cone connected to a $0.5mm$ long cylinder piece with a diameter of $2.3mm$. The NSD 12 is also a truncated cone in which 12 bore holes with a $0.7mm$ diameter are arranged. The lengths of the bore holes is $9.9mm$. In the extension of the bore holes there is a plate around which the cutting gas must flow, before becoming united to the gas which flows directly through the nozzle. The output diameter of the nozzle is $6.0mm$.

7.1.2. Influence of the gas flow on the cut quality

The cut quality depends on many parameters. There are those parameters which characterize the laser beam, and there are others which characterize the formation of



(a) Standard hole nozzle (EAA ∅2.3mm)

(b) Bypass nozzle (NSD 12)

Figure 7.1.: Sectional views of the investigated nozzles.

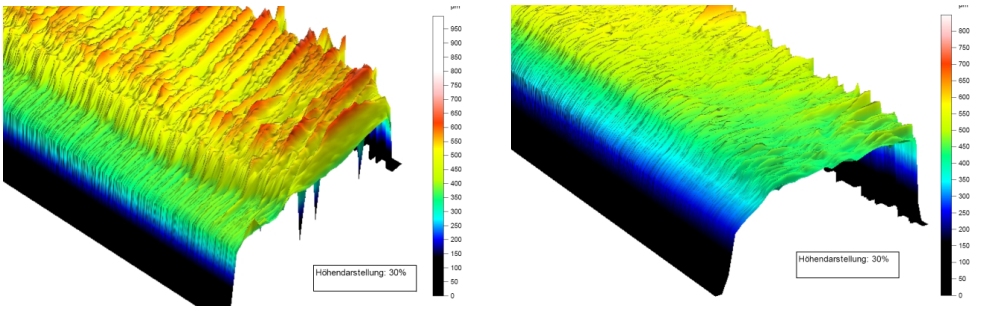
the gas flow. For laser cutting, the necessary gas flow is set through the selection of a suitable nozzle, the standoff height between the nozzle and the work-piece surface and the selected cutting gas pressure. The cutting process determines the gas type. Oxygen is used for flame cutting and nitrogen for fusion cutting. In order to determine the influence of the cutting gas flow on the cut quality, the cutting parameters for the EAA ∅2.3mm were used which would get the best cutting quality in 15mm aluminum. The parameters for the bypass nozzle were subsequently determined. In doing so, however, efforts were made to make the volume flow rate as similar as possible in both cases, so that a comparison of both gas flows is possible. Table 7.1 shows the cutting parameters which resulted in the best cut quality.

Nozzle	Sample	VL $[\frac{m}{min}]$	rel. press. $[bar]$	Flow rate $[\frac{Nm^3}{h}]$	ADB $[mm]$	PL $[kW]$	Set-ting $[mm]$	∅ Raw beam $[mm]$
EAA ∅2.3mm	12	0.7	21	64	1.0	6	7	17
NSD 12	41	0.7	10	60	1.0	6	-27	14

Table 7.1.: Cutting parameters

In order to be able to objectively demonstrate the better cutting procedure if the NSD 12, the surface s of both cuts were probed by a chromatic white light sensor from Precitec and analyzed using the program “MountainsMap“ from Digital Surf. The comparison of the two surfaces in Figure 7.2 shows that when using basically the same gas consumption, the surface quality can be improved through only a modification of the internal geometry of the nozzle.

The measured surface roughness values for both samples confirms the impression gained from Figure 7.2 of the smoother surface for cutting sample 41. Table 7.2 summarizes the R_z values for the upper middle and lower measuring positions.



(a) Surface of the sample no. 12 (EAA $\varnothing 2.3mm$; $15mm$ AlMg3)

(b) Surface of the sample no. 41 (NSD 12, $15mm$; AlMg3)

Figure 7.2.: Surface samples.

Nozzle	EAA $\varnothing 2.3mm$	NSD 12
R_z upper	40	25
R_z middle	124	47
R_z lower	606	247

Table 7.2.: Surface roughness values of the cutting samples.

7.1.3. Pressure profile of both nozzles

Dynamic pressure profiles give an initial image of the flow created by a cutting nozzle. They give information concerning the efficiency of the nozzle, whereas the ratio of the total pressure on the deflector plate to the pressure in the nozzle is understood when speaking of the efficiency. Moreover the cover of the kerf can be removed from the width of the pressure profile. The cover of the kerf is a measurement for how far molten material can be driven out of the kerf in the lag. In order to record the dynamic pressure, a deflector plate is systematically moved under the nozzle in regular distances and the pressure is measured. To do this there is a hole in the plate with a pressure sensor under it. The hole has a diameter of $0.5mm$ and the grid width is $0.25mm$. Figure 7.3 shows the measuring stand at which the dynamic pressure profiles were recorded.

7.1.4. The simplified kerf

For the validation of numerical results, a good correlation between the models used in the flow simulations and the measuring arrangement is necessary. The transfer of the fine grooved structure to the side surfaces of the kerf in the flow model would imply an extremely fine mesh which would be unfeasible to calculate. Therefore the examinations are done using simplified kerf models. For every nozzle a simplified kerf and an associated CAD model was developed. The cutting front of the simplified kerf

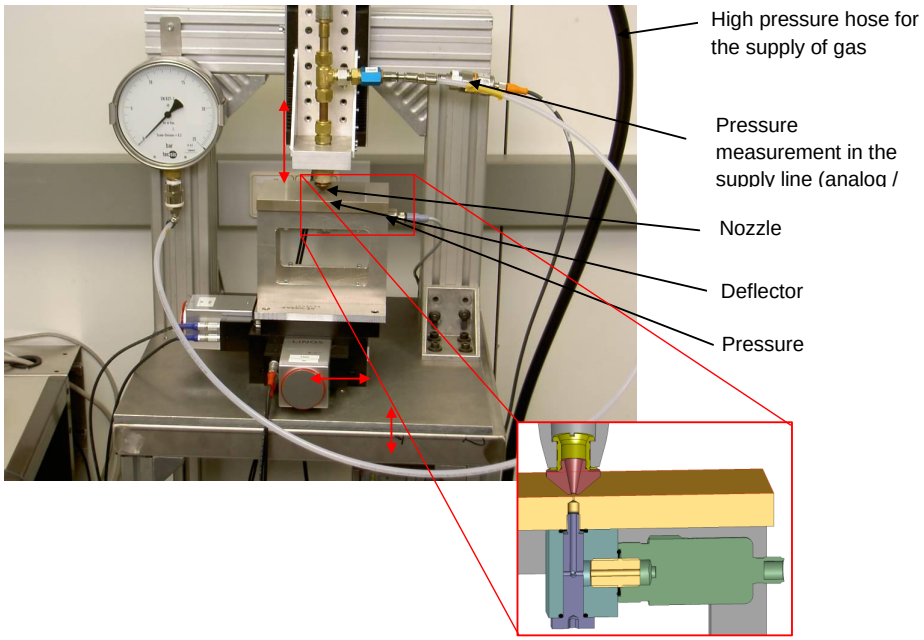


Figure 7.3.: Measuring stand for the recording of the dynamic pressure profiles.

corresponds to that of the real kerf. The side surfaces of the kerf are even, flat walls. For the determination of the front of the cut, a "frozen cut" is generated. Therefore the laser beam is shut off during the cutting process, with the gas flow maintained. This kerf is cut off and the specimen is grind down the middle of the kerf. Figure 7.4 shows the ground kerf of the bypass nozzle NSD 12 with its cutting front and its striations. The grooved structure can be divided into 3 sections. In the upper quarter of the cutting surface, there are the striations of the 1^{st} arrangement. If the cutting speed is correlated with the distance between the striations, these occur at a frequency of up to $150Hz$. A narrow striation section of the 2nd arrangement borders the 1st striation section, as the striation frequency changes. In the lower area of the cutting surface there is a 3rd striation arrangement whose frequency is about $75Hz$.

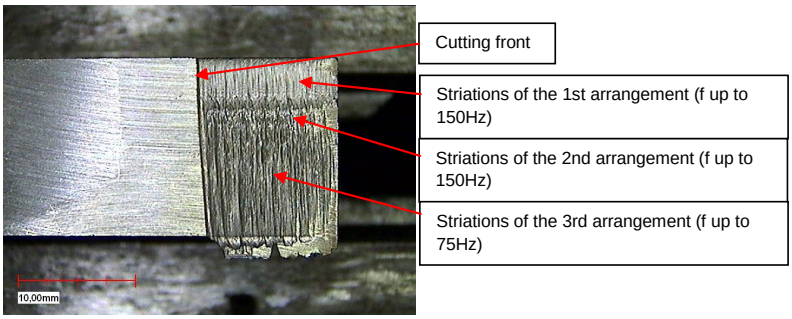


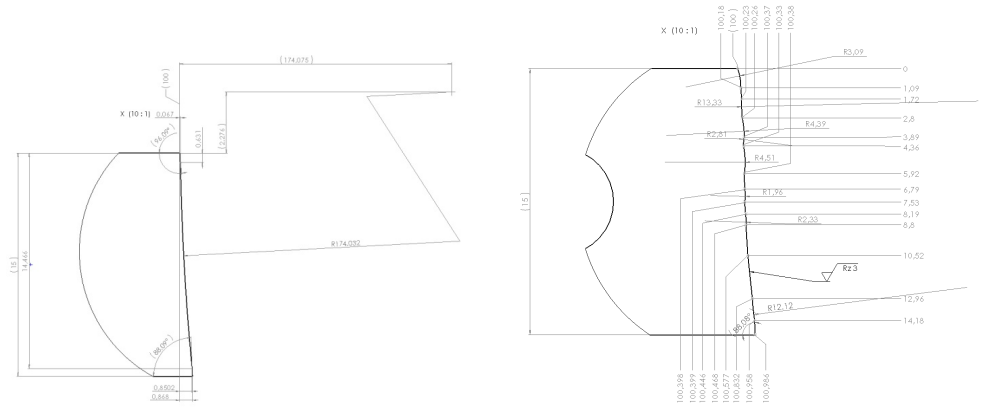
Figure 7.4.: Ground kerf with cut progression and grooved structure.

The development of straight and circular segments of the cutting front is transferred into a technical drawing by adding straight and circular segments. The individual geometries show clear differences. The cutting front of the hole nozzle EAA $\varnothing 2.3mm$ shows a relatively homogeneous progression and can be put together with a curve using two straight lines. The cutting front of the of the NSD 12 nozzle has a face which must be composed of a multitude of straight and curved lines. In Figure 7.5 both of the different cutting front progressions are contrasted.

The "simplified kerf" consists of 2 side plates between which a spacer plate with the same width as the kerf is clamped, see Figure 7.6. The front of the spacer plate has the same geometry as the front of the kerf. The left side plate has 5 holes, with which the pressure at a distance of $0.5mm$ from the front of the kerf hole axis - kerf front edge) can be measured. The hole diameter is $0.5mm$ up to a depth of $2mm$ and then widens to $2.3mm$.

On the right side plate, holes are arranged in a 4×3 matrix so that a pressure profile of $12 \times 15mm^2$ can be measured. The positions and designations of the individual holes can be seen from Figure 7.7. The X axis of the coordinate system from Figure 7.7 progresses in the symmetry plane of the kerf. The Y axis is on the upper side of the model and the Z axis at the upper edge of the cutting front.

The simplified kerf can be positioned in relation to the nozzle using an XY table.



(a) Drawing of the cutting front of the hole nozzle EAA $\varnothing 2.3mm$.

(b) Drawing of the cutting front of the nozzle NSD 12.

Figure 7.5.: Cutting front comparison.

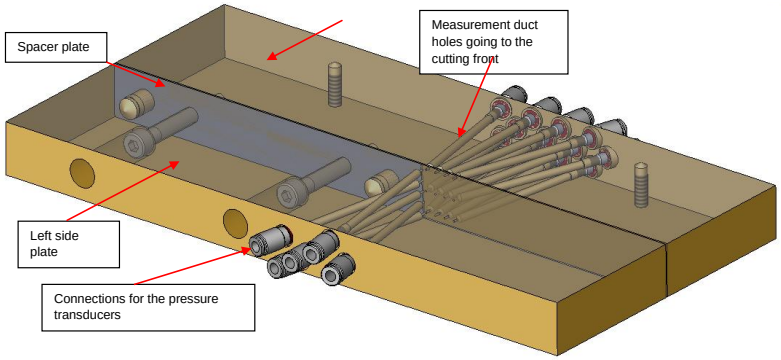
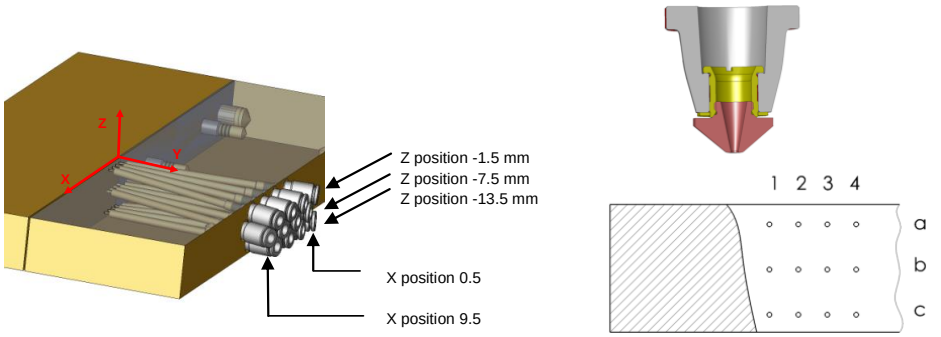


Figure 7.6.: Semi-transparent portrayal of the simplified kerf model.



(a) Position of the measuring holes.

(b) Designation of the measuring position.

Figure 7.7.: Measuring holes.

The adjustment axes have a resolution and axis position reproducibility of $0.5\mu m$ each. Figure 7.8 shows the measuring stand integrated into a sound-proof cabin. The distance between the kerf and the nozzle can be set using the Z axis, on which the nozzle holder is fixed. For all of the measurements and simulations, the nozzle is situated over the middle of the kerf and the distance between the mouth of the nozzle and the kerf is $1mm$. During the cutting process, the cutting front moves to half of the kerf width along the beam axis. According to the coordinate system of Figure 7.7(a) the nozzle is in position $(x, y, z) = (0.25mm, 0.0mm, 1.0mm)$.

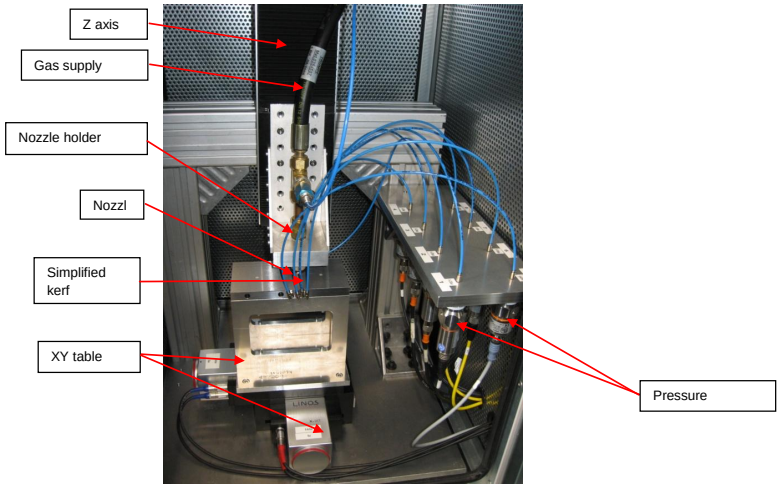


Figure 7.8.: The measuring stand for the pressure measurement in the kerf.

With the Labview program pictured Figure 7.9 in the axis and gas valves can be controlled and the pressure cells can be read. The excess pressure found in the kerf was recorded with pressure cells of type PA9023 from the manufacturer IFM, the low pressure with the pressure sensor type P3291S072020 manufactured by tecsis.

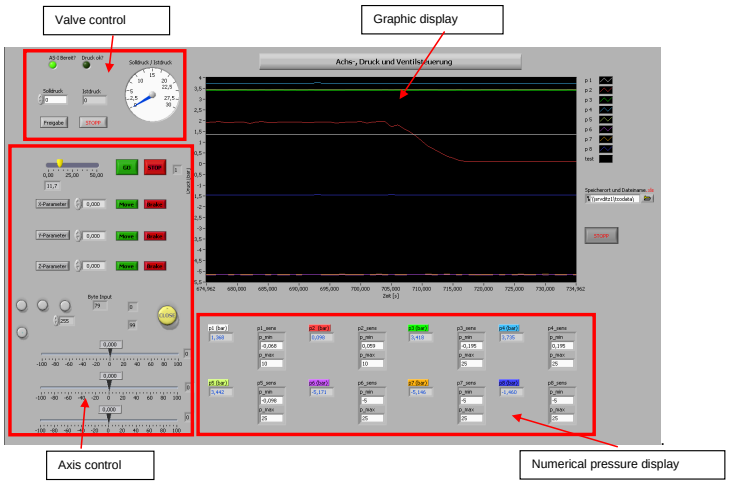
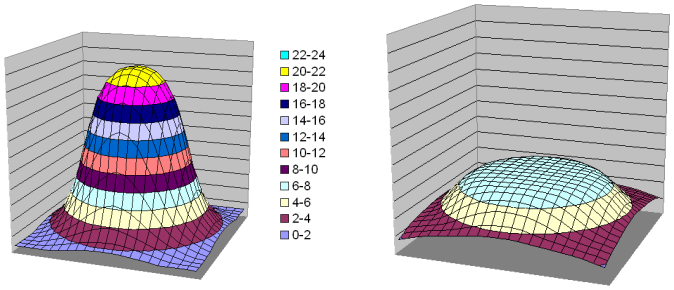


Figure 7.9.: User interface of the program for axis, valve and pressure control and analysis.

7.1.5. Dynamic pressure profile

With similar gas consumption, the pressure profile of the hole nozzle EAA $\varnothing 2.3mm$ is significantly different from that of the bypass nozzle NSD 12, as seen in Figure 7.10. The profile of the hole nozzle looks like a sugar loaf mountain, whereas the bypass nozzle looks more like a plateau.

The different total pressures of $21.8bar$, or $7.52bar$ are owing to the different operating pressures used in operation with the nozzles. In order to compare both profiles, it is recommended to consider the efficiencies more closely. For the hole nozzle it is about 93% ($\frac{\text{max. pressure on plate}}{\text{pressure in supply line}} = \frac{21.8}{23.5}$, whereas it is only 72% with the bypass nozzle. Previous experiments have shown that if the hole of the hole nozzle is enlarged, it has a similar pressure profile to that of the bypass nozzle. A similarly good cutting result like that of the bypass nozzle is, however, not attained. In order to find out what causes the difference in the cutting behavior, the flow in the kerf should be examined both experimentally and numerically.



(a) Pressure profile of the hole nozzle ∅2.3mm at 23.5bar and 64Nm^3/h.

(b) Pressure profile of the NSD 12 at 10.513bar and 60Nm^3/h.

Figure 7.10.: Total pressure profiles in [bar].

7.1.6. Time regime for laser cutting

When assessing a laser cut, beside the absence of a burr on the lower cut edge, the surface quality of the cut flank, and especially the grooved structure is of importance. Because of this, studies will be made as to whether there is a correlation between striation frequency and pressure fluctuations in the cutting gas.

With the counting of the striation frequency a general frequency of several 10Hz results. In the example shown in Figure 7.11, this corresponds to about 40Hz via calculation with the cutting feed rate. The time needed for the creation of a striation is therefore 20ms.

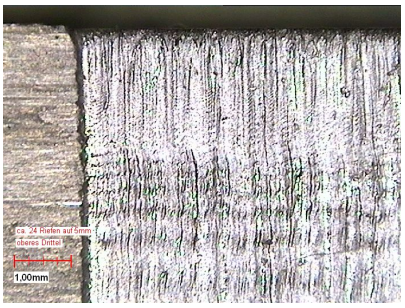


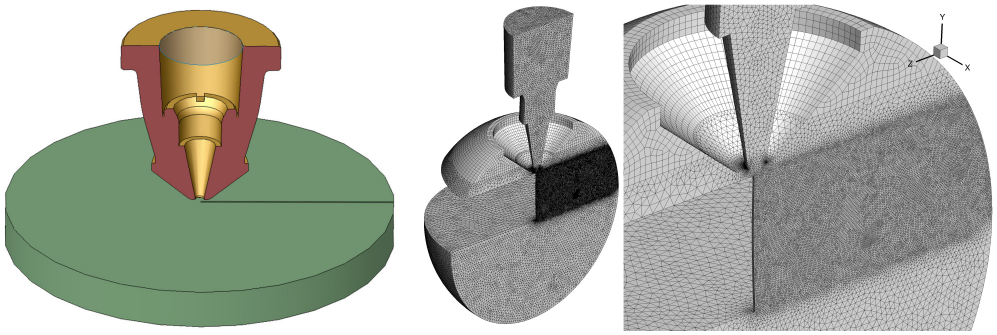
Figure 7.11.: Counting the striation (frequency) under a microscope.

7.2. Numerical investigation

In this section, we will present the setup and results of the simulations conducted with the discontinuous Galerkin code HALO, especially regarding to the unsteady behavior of the flow. The above mentioned simple kerf models with two different nozzles are

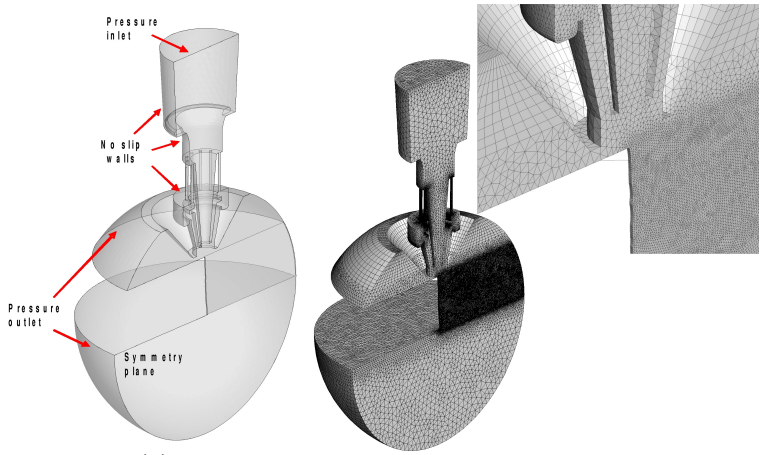
investigated. The model consists of the nozzle positioned over a plate ($t = 15mm$) with an existing kerf, see Fig. 7.12a. The simulation is done for the half model with a symmetry boundary condition in the symmetry plane. The nozzle has a fixed position and all wall boundary conditions are isothermal at ambient conditions, in order to reproduce the experimental setup.

7.2.1. Meshes and boundary conditions



(a) (b)

Figure 7.12.: CAD geometry and volume mesh of the half model for the hole nozzle.



(a) (b)

Figure 7.13.: Boundary Conditions and volume mesh of the half model for the NSD 12 nozzle.

nitrogen, gas constant	$R = 296.8 J/(kgK)$	viscosity	$\mu = 1.663 \times 10^{-5} m^2/s$
ambient pressure	$p_\infty = 101325 Pa$	inlet pressure	$p_0 = p_\infty + p_{\text{gauge}}$
ambient temp.	$T_\infty = 293.15 K$	inlet & wall temp.	$T_0 = T_{\text{wall}} = T_\infty$
ambient density	$\rho_\infty = 1.164 kg/m^3$	inlet density	$\rho_0 = p_0/(RT_0)$

Table 7.3.: Fluid properties and boundary conditions

The meshes are displayed in Fig. 7.12band Fig. 7.13b. The hole nozzle is a high pressure configuration, in which the gauge pressure at the nozzle inlet is at $p_{\text{gauge}} = 21$ bar, and the NSD 12 nozzle has a gauge pressure of $p_{\text{gauge}} = 10$ bar. Both setups yield high Mach numbers and strong shocks inside the kerf. The boundary conditions are listed in Table 7.3, together with the fluid properties. The hybrid mesh consists of 755,908 cells (747,116 tetrahedra) for the hole nozzle and 979,079 cells for the NSD 12 nozzle, resulting in 3.02 and accordingly 3.92 Million DOF using a 2^{nd} order discretization. For a third order computation, the number of degrees of freedom increases by a factor of 2.5.

7.2.2. Initialization and shock capturing

The initialization of the unsteady simulation has to be done carefully. First, we simply initialized the domain with a constant pressure. Due to the high pressure ratio, this strongly increased the simulation time, since the transient start of the flow was be reproduced, making the simulation nearly unfeasible. Instead, a steady state solution produced by FLUENT, which is already available for these test cases, was used to initialize the unsteady simulation. Now, the unsteady flow is fully developed after $0.8ms$ simulation time, or accordingly $\approx$ 1000 core-h. The initial flow field and the fully developed unsteady flow through the hole nozzle and the kerf is shown Fig. 7.14.

In areas of strong shocks, artificial viscosity is applied for stabilization of the HALO code. The location and the strength, detected by a density indicator is shown in Fig. 7.15. The shocks are well detected. In addition, high values at the nozzle exit, where no strong shocks occur, may indicate that the mesh in this area is too coarse.

7.2.3. Parallel Performance

Several simulation runs were performed on the HLRS CRAY XE6 system with 512 and 2048 cores. The simulation runs are listed in Table 7.4. For the hole nozzle and the second order simulation, the overall computational effort for simulating $t = 3.09ms$ sums up to 4326 core-h. Looking at the core-h needed for $1ms$ simulation time, the strong scaling from 512 to 2048 cores is still 84%. The computational effort per ms simulation time is about 1500 core-h. A third order computation was also conducted for the hole nozzle, thus the number of DOF increased by a factor of 2.5 and the

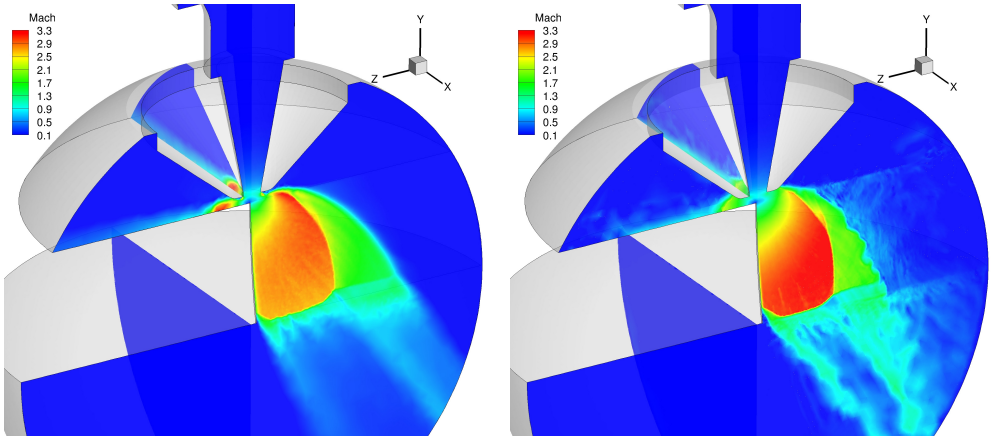


Figure 7.14.: Mach number distribution in the yz and xy plane. Left: steady state FLUENT solution. Right: instantaneous HALO solution after $t = 0.85ms$.

computational effort increases by a factor of 6. In the test case description, d_0 is the strength of the shock capturing viscosity. It can be reduced for the third order computation due to the higher resolution. However, the viscosity reduces the time-step inside the grid cell, but since the code is able to do local time stepping, this only leads to a higher local time stepping factor (LTS = ratio of the smallest time-step to the mean overall time-step). The LTS factor gives us the gain in computational effort compared to a global time stepping scheme. The computational effort for the NSD 12 nozzle was nearly the same, about 1400 core-h per ms simulation time.

7.2.4. Results

In this section, the HALO simulation results are compared to the steady state FLUENT solution and the measurements. First we will show the results for the NSD 12 nozzle and proceed then with the evaluation of the hole nozzle.

The DG scheme has a factor of 4 time more DOF on the same mesh for second order. The comparison of the velocity magnitude in Fig. 7.2.4 between the FLUENT and the HALO simulation shows the higher resolution of the DG scheme. Especially on the upper plate surface and the cutting edge, finer structures can be seen. Since the HALO solution is started from the stationary FLUENT solution, the shock positions fluctuate strongly for approximately $0.8ms$ and reach then quasi-stationary positions with small fluctuations due to the turbulent flow. To be able to compare the data, the instationary solution is averaged in time, starting at $0.9ms$, to skip the transient starting phase. In Fig. 7.17, the results are compared between FLUENT and HALO at a gauge pressure of $10bar$ and a second simulation was conducted with HALO for a changed gauge pressure of 9.5 bar. This computation shows on the one hand the influence of small pressure changes in the shock positions, on the other hand it helps to

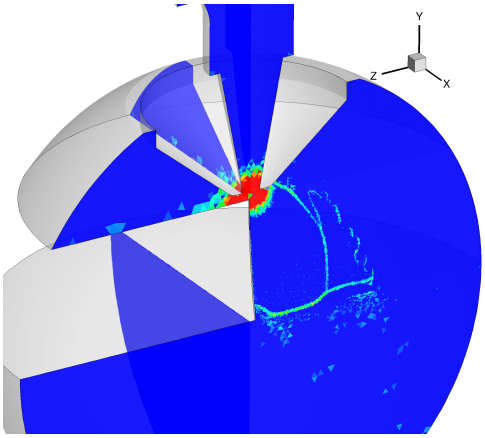


Figure 7.15.: Shock indicator showing strong variations in density

compare with the measurement data, which was also recorded at this gauge pressure. Due to the change in pressure, the lower shock position moves approximately $0.5mm$ upwards, and the angle between the shock and the nozzle axis also changes about 4 degrees. The circles in Fig. 7.17 show the position of the pressure tappings.

If a shock is near the tapping, large pressure variation over the diameter occurs $(1c, 3a, 4c)$. Since small difference in shock position can cause strong deviations if we only compare the pressure at the tappings, the mean flow data was extracted along horizontal lines a,b,c going through the tappings, from position 1-4 from left to right. The pressure is normalized to the inlet pressure, so that different inlet pressures can be compared. In Fig. 7.2.4 this is plotted versus the experimental data.

For the uppermost line (a), the pressure distribution of both simulations match well with experimental data. For the middle and the lower line (b,c) the differences at the pressure tappings are very high, mainly caused again by small differences in shock positions. This effect can be seen in particular at position 4b, where the FLUENT simulation under-predicts the pressure and both HALO solutions over-predict the pressure. Thus due to the different numerical schemes, both on a very coarse mesh, small differences in the flow pattern are produced. Also on the experimental side, the alignment between the nozzle and the kerf will have strong influence on the measured data.

In the following part, the same investigations are shown for the hole nozzle. The mesh resolution inside the kerf is the same as for the NSD 12 nozzle. We first show the difference of the FLUENT and the HALO simulation in Fig. 7.2.4. Again, the HALO simulation tends to resolve finer flow features.

The comparison between FLUENT and HALO is shown Fig. 7.20a and b. The pressure distribution is very similar, the overall pressure level is slightly lower in the HALO Solution and at the cutting edge, the flow separates later, in the region of

#cores	DOF per core	start - end sim.time $[ms]$	wallclock time $[h]$	core-h	core-h per $[ms]$ sim.time	t_{CPU} per (ΔtDOF) $[\mu s]$
Hole nozzle, $\mathcal{O}2$, $d_0 = 500$. $t_{\text{sim}} = 3ms$. LTS: 100						
512	5906	0.00 - 0.89	2.34	1197	1344	28.06
2048	1476	0.89 - 1.57	0.53	1092	1605	32.94
512	5906	1.57 - 3.09	3.98	2037	1340	27.99
Hole nozzle, $\mathcal{O}3$, $d_0 = 300/250$. $t_{\text{sim}} = 1.2ms$. LTS: 150						
512	14764	4.00 - 4.28	7.46	3818	13636	34.14
512	14764	4.28 - 4.49	3.86	1975	9403	27.28
2048	3691	4.49 - 5.26	3.99	8176	10618	29.30
NSD 12 nozzle, $\mathcal{O}2$, $d_0 = 500$. $t_{\text{sim}} = 3.7ms$. LTS: 32						
512	7649	0.13 - 1.53	3.99	2042	1459	20.94
512	7649	1.53 - 2.22	1.90	972	1409	20.27
512	7649	2.22 - 3.68	3.99	2045	1401	20.15

Table 7.4.: Parallel laser cutting device simulations on the HLRS CRAY XE6 system.

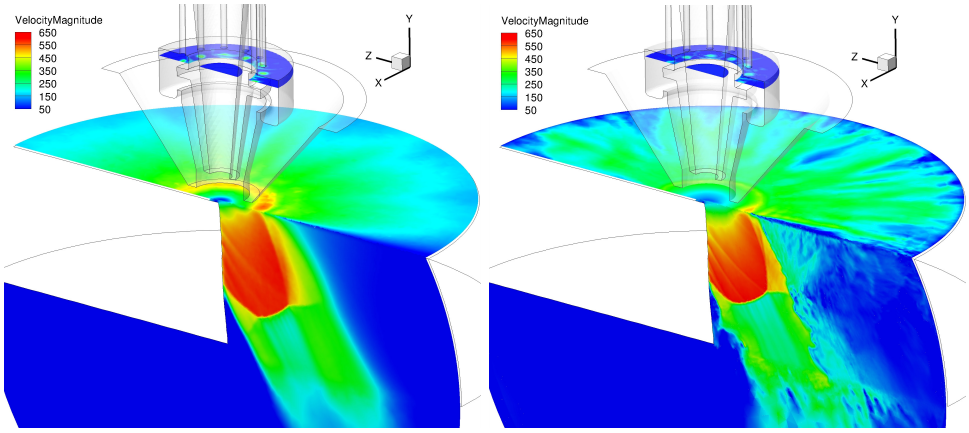
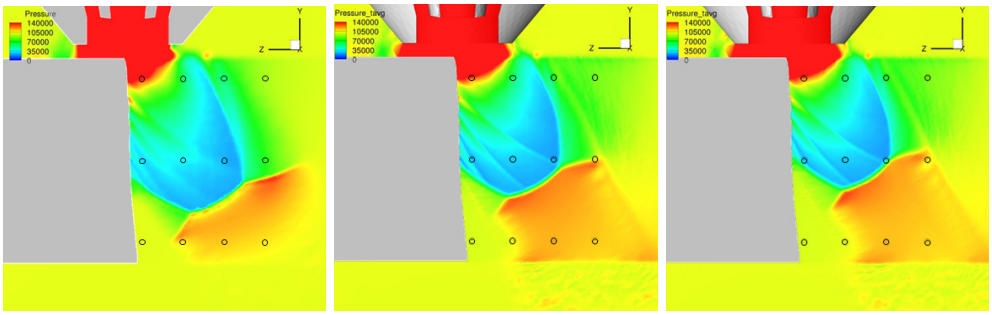


Figure 7.16.: Distribution of the velocity magnitude for the NSD 12 nozzle, computed with FLUENT (left) and the instationary solution with HALO(right) after $3.68ms$.



(a) (a) (b) (b) (c) (c)

Figure 7.17.: Mean pressure distribution in the symmetry plane for (a), HALO $\mathcal{O}2$ (b) and HALO $\mathcal{O}2$ with $p_{\text{gauge}} = 9.5$ bar (c) for the NSD 12 nozzle.

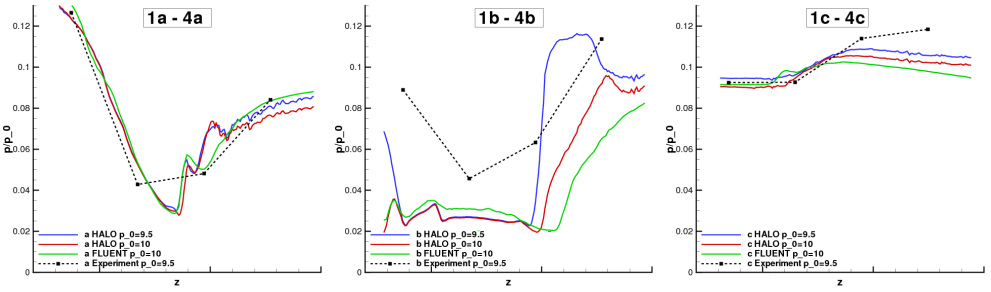


Figure 7.18.: Pressure profiles along the lines a, b and c for the NSD 12 nozzle.

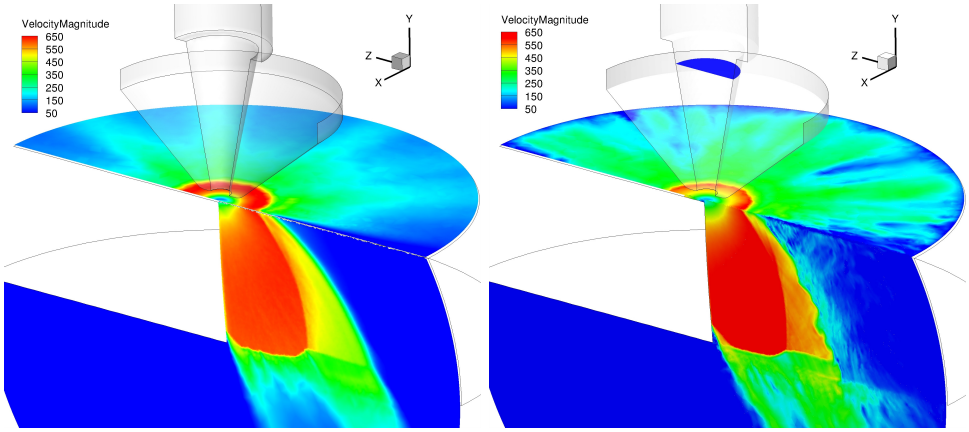
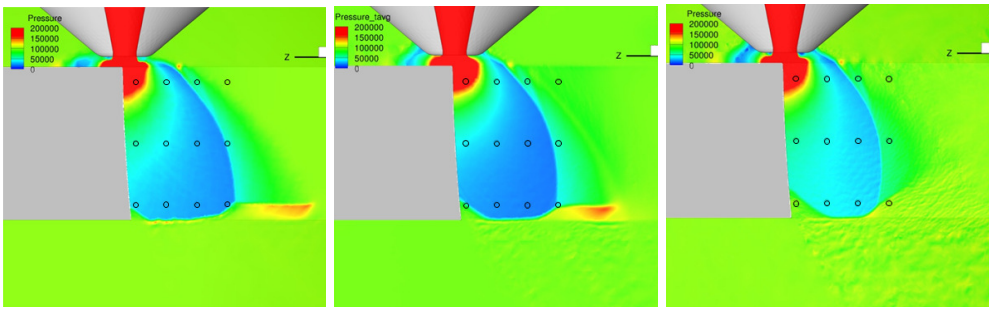


Figure 7.19.: Distribution of the velocity magnitude for the hole nozzle, computed with FLUENT (left) and the instationary solution with HALO(right) after $3ms$.

pressure tapping 1c. For this test case, a third order computation was conducted on the same mesh, yielding a higher resolution. The difference to the second order solution can be seen in Fig. 7.20b and c. Already at the nozzle exit, boundary layers can be resolved for the third order case, and thus the flow pattern changes. Also the shock on the right of position 4c disappears, and the shock positions change, and the overall pressure level is higher.



(a) (a) FLUENT (b) (b) HALO $\mathcal{O}2$ (c) (c) HALO $\mathcal{O}3$

Figure 7.20.: Mean pressure distribution in the symmetry plane for the hole nozzle.

It is difficult to judge which simulation now represents the correct pressure distribution in the kerf. The third order computation is the closest to the computations. see Fig.7.2.4. However, higher resolution simulations must be done to ensure this claim, either by refining the mesh of increasing the order of the DG scheme. The uppermost

line (a) shows again good agreement with the experiment. The differences in the middle line (b) are much smaller in comparison with the results from the NSD 12 nozzle. This may be caused by the absence of a dominant shock and a homogeneous pressure distribution. Similarly, this carries over to line c. In general, the pressure distributions of the simulations are very similar and only differ by a constant offset.

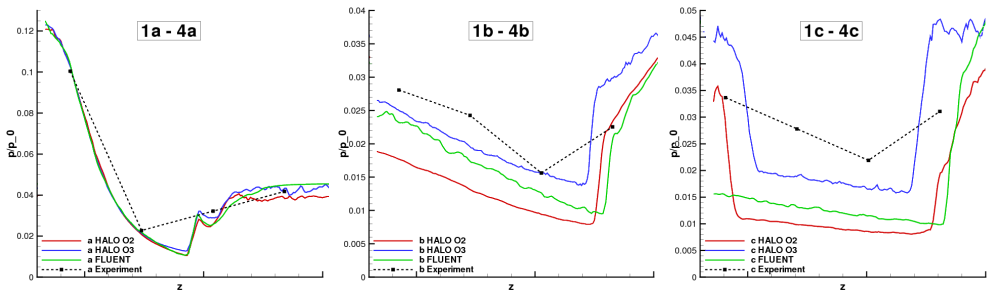


Figure 7.21.: Pressure profiles along lines a, b and c for the hole nozzle test case.

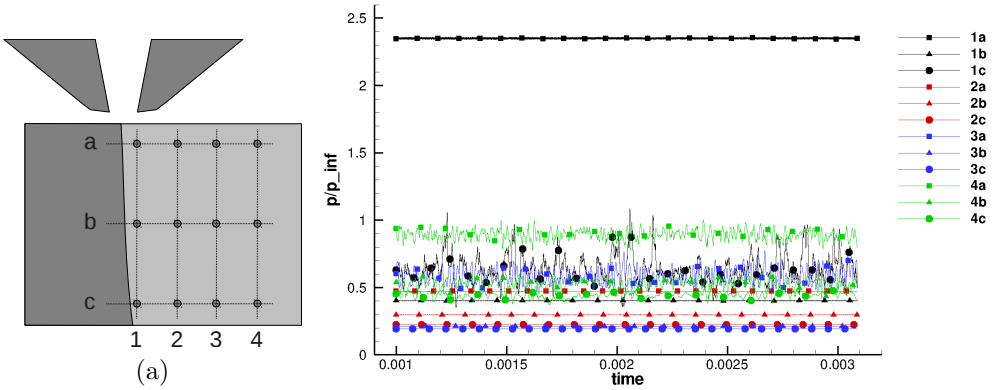


Figure 7.22.: Nomenclature of the pressure tappings and temporal evolution of the pressure at these positions.

At specific record points, pressure data is tracked in time during the simulation. They are chosen to match the pressure tappings of the experiment, with a diameter of $d = 0.5mm$, see Fig. 7.22a. In the HALO simulation, it was found that the flow is inherently unsteady inside the kerf, with a turbulent shear layer and oscillating shocks. In Fig. 7.22b, the temporal evolution of the pressure is plotted. In Table 7.5, the root mean square of the pressure fluctuations is shown. The largest fluctuations are found at $1c, 3a, 4a, 4b, 4c$, which are all tappings lying in proximity to shocks or shear layers. At position $1c$, the strongest fluctuations are found. Here, we transformed the

pressure fluctuations in the time interval $t = 0.1 - 0.3ms$ by a Fourier transform, see Fig. 7.23. The gray shaded area indicates the limit of the frequency resolution. The lowest resolved frequency for a time interval of $2ms$ is $10/2ms = 5000Hz$ (a minimum of 10 modes). For the tapping 1c, there is no unique peak frequency, but the maximum peak is around $10,000Hz$.

	1a	1b	1c	2a	2b	2c	3a	3b	3c	4a	4b	4c
mean [*bar*]	2.36	0.41	0.63	0.48	0.30	0.23	0.61	0.22	0.20	0.91	0.54	0.48
RMS [*mbar*]	7.5	0.1	**138**	0.3	0.09	0.05	**70.7**	0.06	0.05	**45.6**	**44.4**	**84.9**

Table 7.5.: Pressure fluctuations at the pressure tappings.

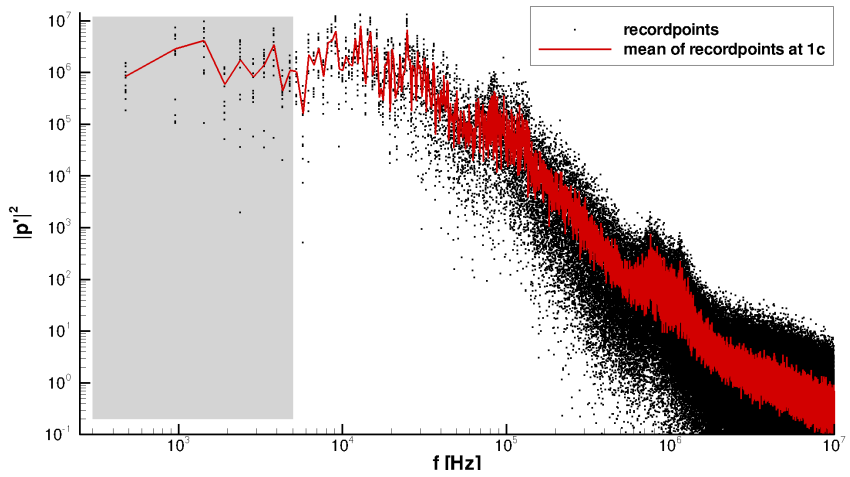


Figure 7.23.: Fourier transform of pressure fluctuations at tapping 1*c* (gray: unresolved frequencies).

7.2.5. Speed-up experiments with FLUENT

In the framework of the BMBF project, not only should the understanding of the process be improved, but knowledge should be gained concerning what a technical solution at TRUMPF should look like in the medium-term, in order to do parameter research profitably and effectively and with its help optimize nozzle geometry. For the evaluation of the profitability or effectiveness, the necessary computing time or the necessary number of licenses are, beside the physical quality of the calculation results, also a decisive factor. At present, TRUMPF's obtaining of licenses of commercial programs is minimal, as the benefits for daily business have not justified the relatively

high costs for flow solvers. Because of this it has become a project target for TRUMPF to investigate which physical result quality can be achieved with which hardware and software means in a given time frame. In order to obtain the first estimate of the potentially necessary resources, an investigation was made as to how the computing machine used and the number of processors affect the computing time for a given test case using Fluent. The stationary calculations for the hole nozzle serve here as the test case. The time needed for each case will certainly be the time needed to solve 5000 iterations. The test case was calculated on a "TRUMPF CAD computer" and on the Nehalem Cluster of the HLRS with varying numbers of processors.

The TRUMPF computer has an Intel Core2 E6600 processor with $2.4GHz$ and $3.5GB$ RAM. The computing time for the 5000 iterations was 9.23 hours on the TRUMPF system using both processors.

An experiment was done on the Nehalem Cluster to determine the influence of the number of processors on the computing time. Table 7.6 provides an overview of the calculations performed and the computing time of the test case depending on the number of processors used.

Number of processors	1	2	4	6	8	16	28	32
Computing time $[h]$	9.56	4.73	2.52	2.07	1.57	0.8	0.5	0.48

Table 7.6.: The computing duration of the test case depending on the number of processors used.

The comparison of the computing time needed by 2 processors shows that simply due to the change from the existing desktop architecture to processors with Nehalem architecture the computing time can be halved. Beyond that the speed-up experiment at HLRS reveals that the computing time is reduced in a linear manner up to the use of 4 processors, see Figure 7.24. When using many processors the computing time lags behind the theoretically needed time or the calculations aborted which can certainly be traced back to the smaller domain sizes.

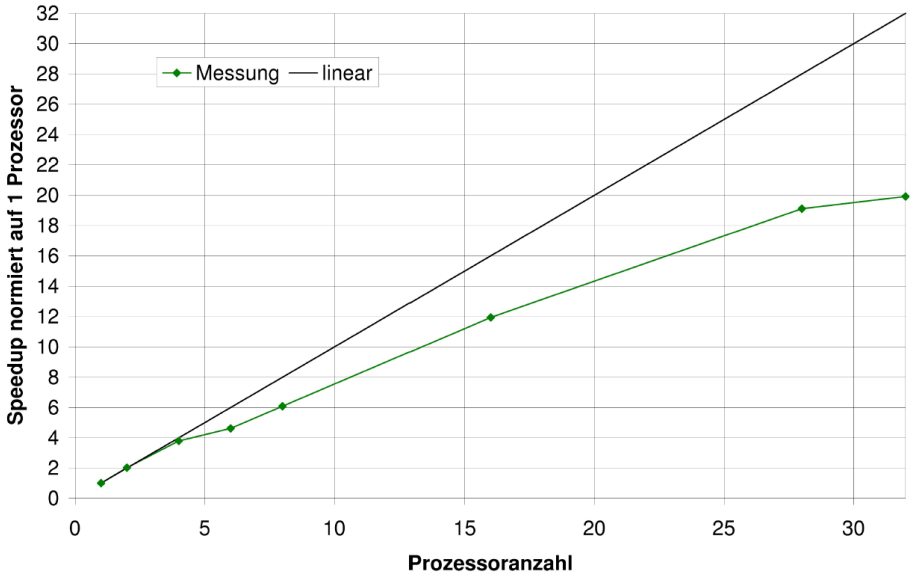


Figure 7.24.: Evaluation of the speedup test for FLUENT.

7.3. ZFS

To demonstrate the general applicability of the Cartesian flow solver "ZFS" developed at AIA for Large-Eddy simulations of compressible flows to the considered laser cutting nozzle, a first feasibility study for the hole nozzle EAA has been carried out using simplified boundary conditions. The flow solver is based on Cartesian meshes and applies a finite volume cut-cell method to treat embedded boundaries. This enables automatic grid generation and boundary treatment even for complex technical geometries with sharp edges and complex details [11, 10] . For the feasibility study the nozzle exit of the EAA nozzle has been positioned 18mm above a solid surface. The region between the nozzle and this surface where the fluid streams out of the nozzle is assumed to be block-shaped and extends up to half of the nozzle height in the surface normal direction. The extension in the directions parallel to the solid surface is 90mm. The automatically generated Cartesian grid which is displayed in Fig. 7.3 is refined on all boundaries at which a no-slip condition is applied and consists of approximately 2.5 million cells.

To minimize the computational costs and to enable the application of the existing subsonic boundary conditions, the pressure difference between the inflow boundary at the upper nozzle cross section and the outflow boundaries at the boundary surfaces of the block-shaped domain has been prescribed such that the resulting flow field has a Mach-number $M = 0.11$ and a Reynolds number $Re = 600$ with respect to the maximum velocity in the nozzle exit and the nozzle exit diameter. The resulting flow

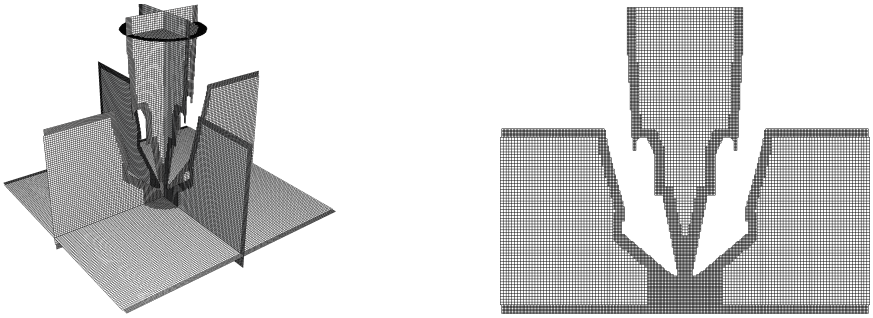


Figure 7.25.: Cartesian grid with boundary refinement for the simulation of the flow through the hole nozzle EAA.

field and the pressure distribution are shown in Fig. 7.3. A simulation of the flow field at boundary conditions which are comparable to the boundary conditions of the realistic application would require the implementation of suitable supersonic boundary conditions as well as a higher mesh resolution and therefore would be considerably more involved. These works were not scheduled in the work program and could not be pursued in more detail due to a lack of time. However, in principle flow simulations with the respective extensions seem to be possible.

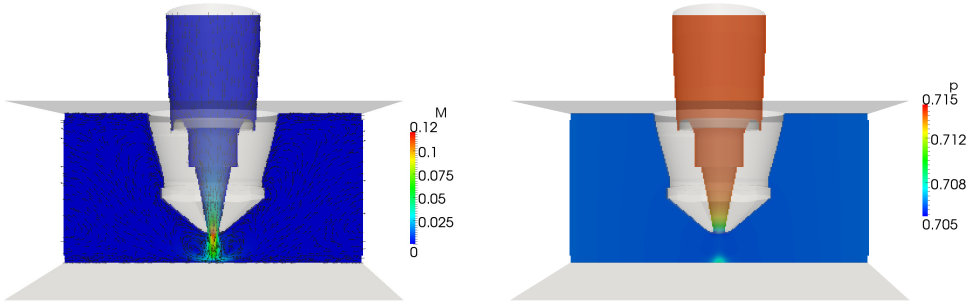


Figure 7.26.: Flow through the hole nozzle EAA: (left) Mach number distribution and flow direction, (right) pressure distribution.

8. High-Lift Aerodynamics

8.1. HGR-01 Profile at High Angle of Attack

In this study, the configuration at an angle of attack of 12^o with a laminar separation bubble and trailing-edge separation is simulated at a Reynolds number of $Re_c = 0.65 \cdot 10^6$ based on the chord length c. Results of a pure LES computation are used as reference data for a fully coupled zonal RANS-LES solution. The transition at the upper surface is predicted by the LES while the lower surface is entirely laminar.

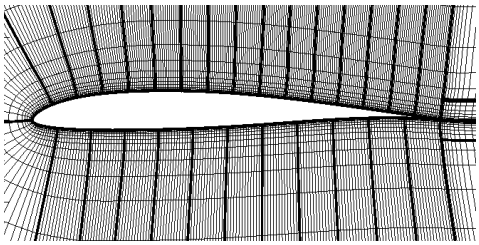


Figure 8.1.: LES grid with 32 blocks
51M grid point

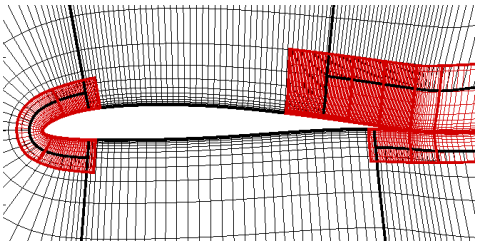
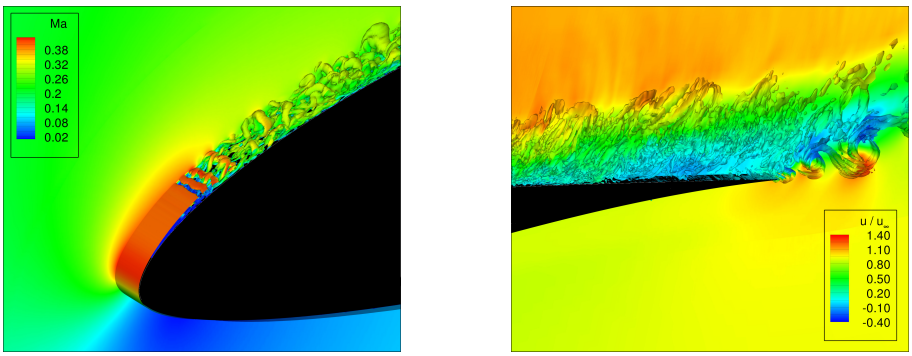


Figure 8.2.: Zonal grid (red = LES, black = RANS)
13M grid points

The grid resolution for the pure LES computation and the LES domain of the zonal RANS-LES simulation is chosen according to Zhang *et al.* [35]. The resolution of the pure LES grid in the stream-wise, wall normal, and span-wise direction of $\Delta x^+ \approx 100$, $\Delta y^+_{min} \approx 1$ and $\Delta z^+ \approx 20$, respectively, results in a mesh with $51.4 \cdot 10^6$ grid points. The span-wise extension of the grid is $0.02\,c$. Using the same grid resolution and span-wise extension for the LES domains in the zonal RANS-LES grid, the total number of grid points was reduced by a factor of 4 to $13.2 \cdot 10^6$ grid points. The grid for the pure LES computation is depicted in Fig. 8.1 and exist out of 32 blocks.
The complexity of this test case for a zonal RANS-LES approach not only lies in the simulation of the different flow phenomena but also in positioning the LES domains and the transition from the RANS into LES domains and vice versa. These transition regions are located at positions in the flow where different conditions exist, such as laminar and turbulent flow and both positive and negative pressure gradients. One LES domain surrounds the leading edge to capture the LSB and the laminar-to-turbulent transition. A second LES region is located at the trailing edge to accurately predict the highly unsteady behavior of the trailing edge separation. The rest of the computa-

tion domain is meshed with a RANS grid. An overview of the grid lay-out around the HGR-01 airfoil can be found in Fig. 8.2, which applies 6 RANS and 10 LES blocks. The flow dynamics simulated in the LES domains around the leading edge and the trailing edge are visualized in Fig. 8.3. The LSB and the laminar-to-turbulent transition are visualized by λ_2 structures in Fig. 8.3(a). The vortex shedding of the LSB is clearly visible as well as the three-dimensionality of the flow after transition. Fig. 8.3(b) visualizes the flow at the trailing-edge separation and the large vortex structures in the wake.



(a) Close-up of leading edge

(b) Close-up trailing edge

Figure 8.3.: λ_2 structures from the zonal computation showing the LSB and laminar-to-turbulent transition at the leading edge and the turbulent separation at the trailing edge, mapped on Mach number and stream-wise velocity respectively.

Fig. 8.4 shows the averaged pressure coefficient c_p. The gray shaded areas represent the LES domains around the leading and the trailing edge. A smooth transition from the RANS- to the LES zone can be observed. Precise simulation of the LSB at the leading edge is essential for the flow dynamics of the entire airfoil. The difficulty herein comes from the position of the LES inflow boundary upstream of the leading edge, with a large negative pressure gradient from the incoming flow. The close-up clearly visualizes the ability of the zonal method to capture the position and length of the LSB. The small deviation with respect to the experiments depends on the absence of free-stream turbulence in the pure LES and the zonal computation. From the pressure distribution in Fig 8.5(a) and the skin-friction coefficient in Fig. 8.5(b) it can be seen that the LSB is slightly shorter than for the pure LES data, however, still somewhat longer than the experimental data. The skin-friction coefficient also shows a slightly smaller negative friction peak at the end of the LSB, indicating the smaller height of the bubble as seen from the velocity profiles shown in Fig. 8.6.

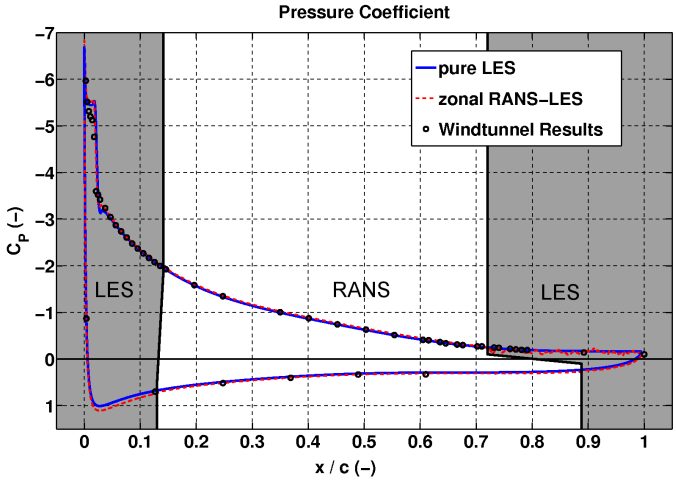
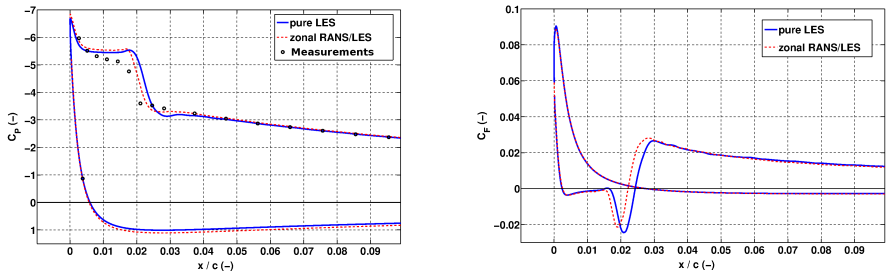


Figure 8.4.: Pressure coefficient c_p at upper and lower side of the HGR-01 airfoil for the zonal RANS-LES, pure LES computations, and experiments.



(a) Close-up of c_p over x/c

(b) Close-up of c_f over x/c

Figure 8.5.: Pressure coefficient c_p and skin-friction coefficient c_f close-up of the leading edge of the HGR-01 airfoil for the zonal RANS-LES and pure LES computations.

The velocity profiles in Fig. 8.6 are positioned at several stream-wise positions on the upper surface of the HGR-01 airfoil. The profiles from left to right represent the LSB at $0.012\,c$, the LES-to-RANS transition at $0.14\,c$ and three profiles in the trailing edge LES region, i.e. at $0.68\,c$, $0.85\,c$ and $0.95\,c$. The results of the averaged zonal RANS-LES and the pure LES are compared, at the trailing edge separation region particle-image velocimetry (PIV) data are used to validate the numerical results. The PIV results depend on the span-wise position and show a small three-dimensional effect in the trailing-edge separation. The maximum span s of the experimentally investigated airfoil is $3.25\,s/c$ and the visualized PIV results represent the velocity profiles at 1.6, 1.9 and $2.6\,s/c$ respectively. Both the reference pure LES computation as well as the zonal RANS-LES simulation show good agreement with the PIV measurements.

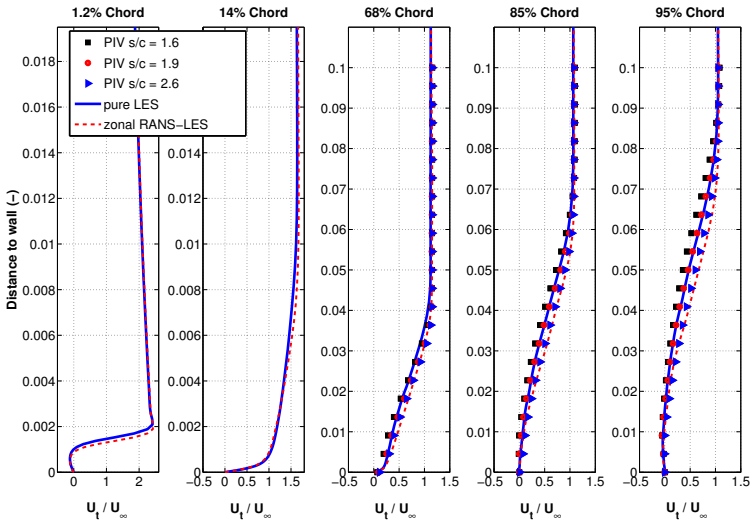


Figure 8.6.: Velocity profiles at different stream-wise positions for zonal RANS-LES, pure LES computations and experiments.

The importance of this test case can be found in the correct simulation of the LSB together with the laminar-to-turbulent transition, as these phenomena influence the entire flow field around the airfoil. The position of the trailing-edge separation and the velocity profiles in the re-circulation zone show that flow characteristics are well transferred from the LES region around the leading edge to the RANS domain at the upper surface and again into the LES region at the trailing edge.

The fact that the zonal RANS-LES method is capable of reproducing these phenomena with high precision demonstrates the capabilities of the zonal RANS-LES method to determine the airfoil aerodynamics such as the lift and drag coefficients at high angles of attack. Table 8.1 shows a comparison of the characteristic values with respect to

	LES	ZONAL	Experiments	RANS
Lift C_l	1.366	1.426	1.370	1.53
Drag C_d	0.0403	0.0414	0.032	0.028

Table 8.1.: Lift and drag comparison

the pure LES computation, the experiments, and RANS data [34]. The lower drag coefficient for the experiments can be explained by the fact that only the pressure component of the drag is measured with pressure probes on the upper and lower surface. The RANS overestimates the lift and underestimates the drag due to the fact that the RANS model predicts the turbulent separation point too far backwards to the trailing edge.

8.2. 2-Element High-Lift Configuration

A second validation for the industrial application of airfoil aerodynamics, is the simulation of the flow around a 2-element high-lift configuration of an airfoil with extended flap at the leading edge. Using the zonal RANS-LES method for complex geometries is translated here in a reduction in grid size with a factor 10 with respect to a pure LES grid. Figure 8.7 shows the grid topology of the high-lift test-case.
In figure 8.8, the time-averaged Mach number contours and streamlines around the profile leading edge and slat are shown. Smooth transitions between RANS and LES domains are attained

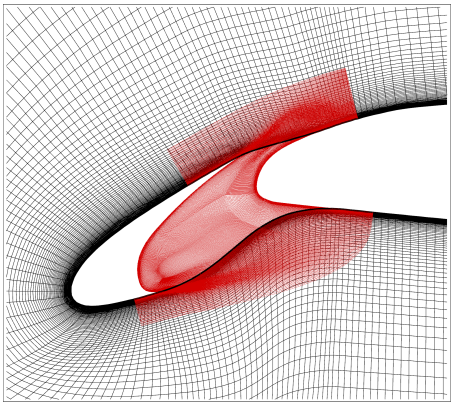


Figure 8.7.: Zonal grid for a swept 2-element high-lift configuration, Red = LES grid, Black = RANS grid.

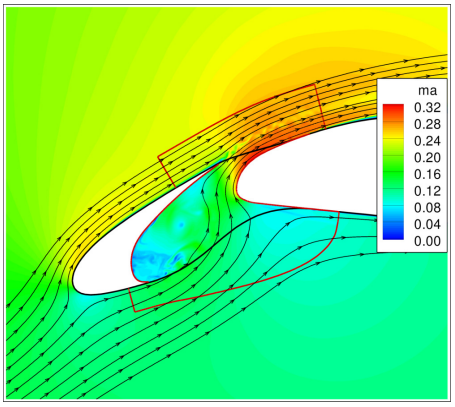
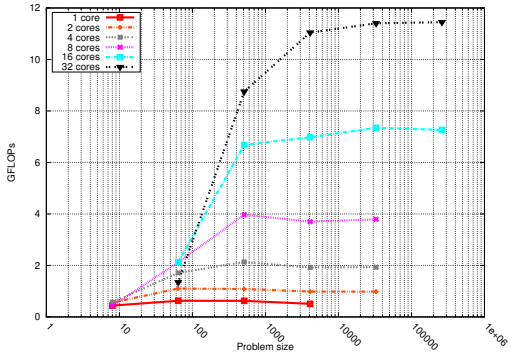


Figure 8.8.: Mach number contours and streamlines at the profile leading edge and slat

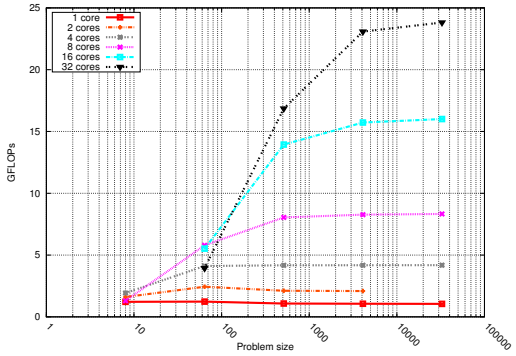
9. Summary

A. Appendix

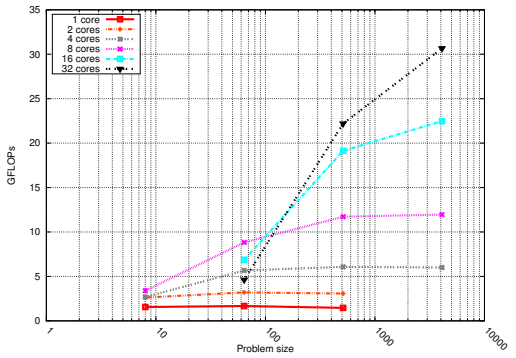
A.1. Performance maps for the HALO Code on current supercomputing architectures



(a) 2nd-order

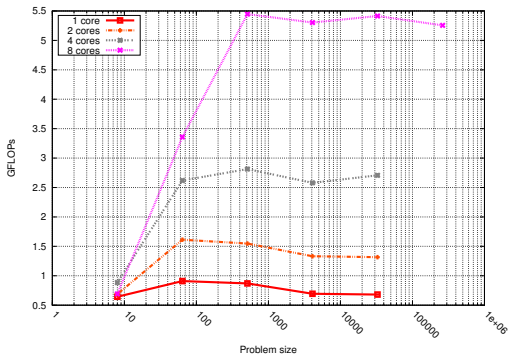


(b) 4th-order

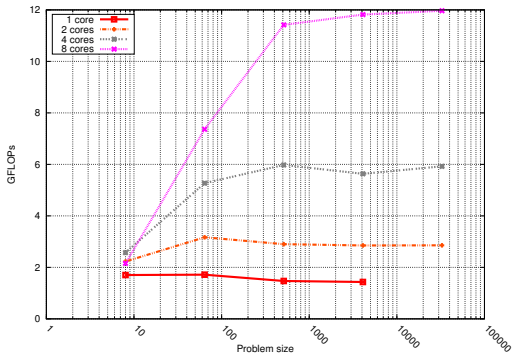


(c) 6th-order

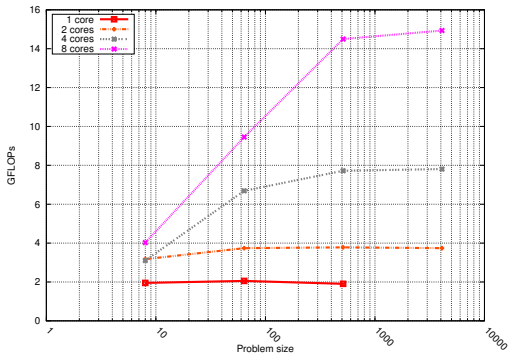
Figure A.1.: Intra-node performance maps for the Hermit system for different spatial orders and a fixed temporal order of three.



(a) 2nd-order

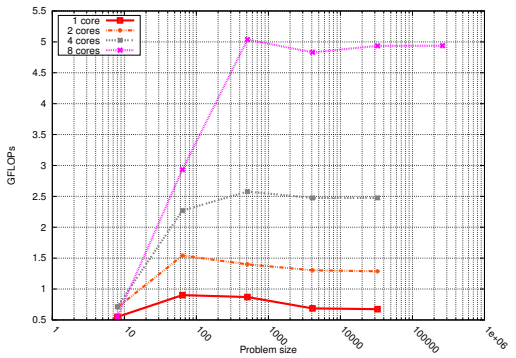


(b) 4th-order

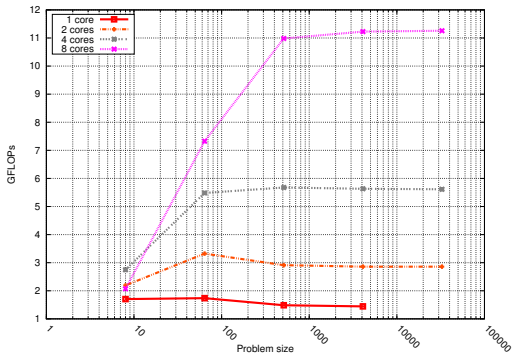


(c) 6th-order

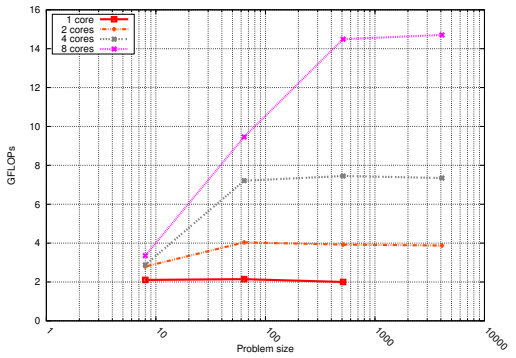
Figure A.2.: Intra-node performance maps for the Laki system for different spatial orders and a fixed temporal order of three.



(a) 2nd-order

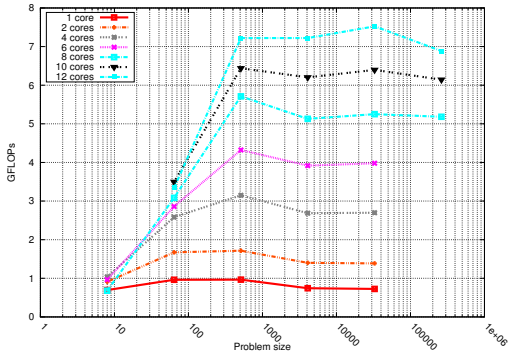


(b) 4th-order

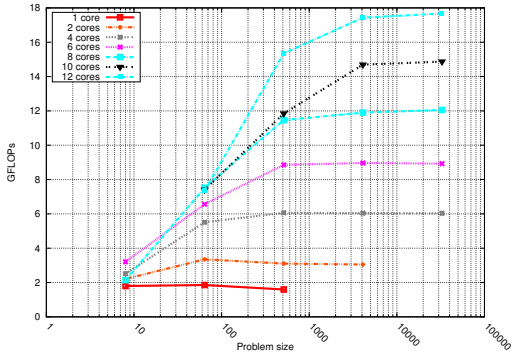


(c) 6th-order

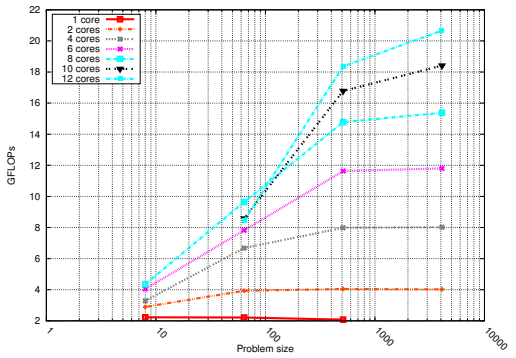
Figure A.3.: Intra-node performance maps for the Juropa system for different spatial orders and a fixed temporal order of three.



(a) 2nd-order

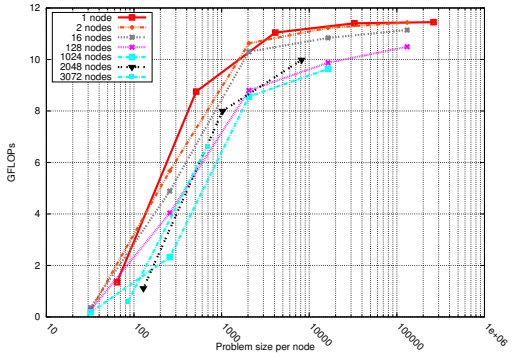


(b) 4th-order

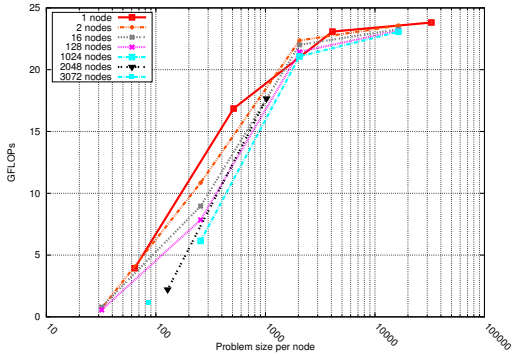


(c) 6th-order

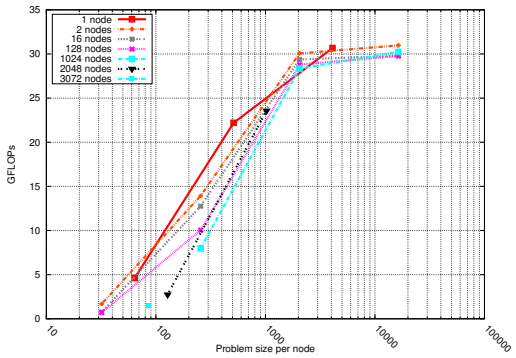
Figure A.4.: Intra-node performance maps for the Cluster at the RZ of RWTH Aachen University for different spatial orders and a fixed temporal order of three.



(a) 2nd-order

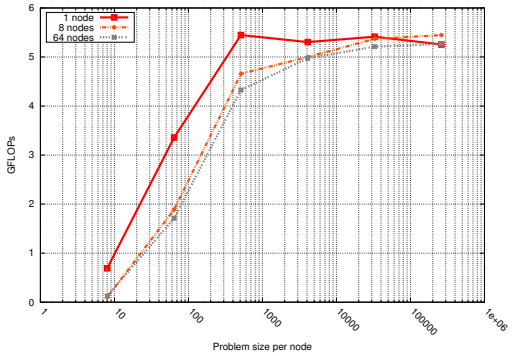


(b) 4th-order

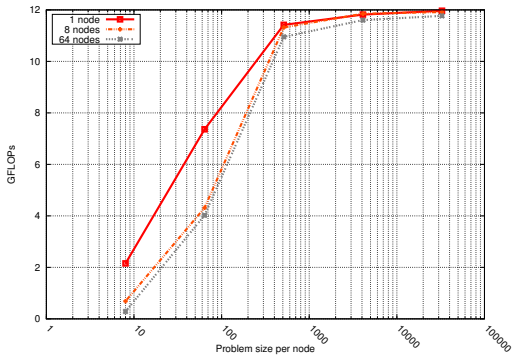


(c) 6th-order

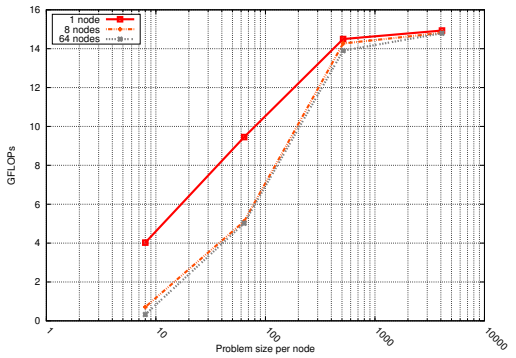
Figure A.5.: Inter-node performance maps for the Hermit system for different spatial orders and a fixed temporal order of three.



(a) 2nd-order

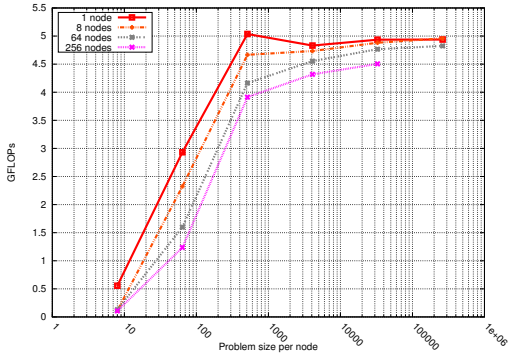


(b) 4th-order

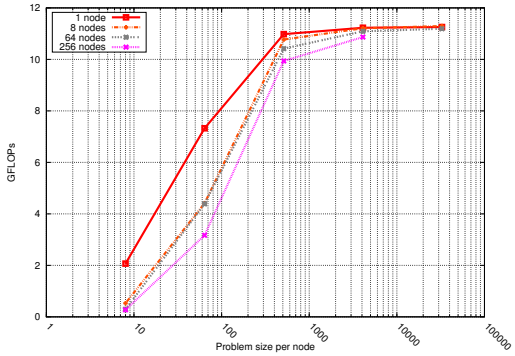


(c) 6th-order

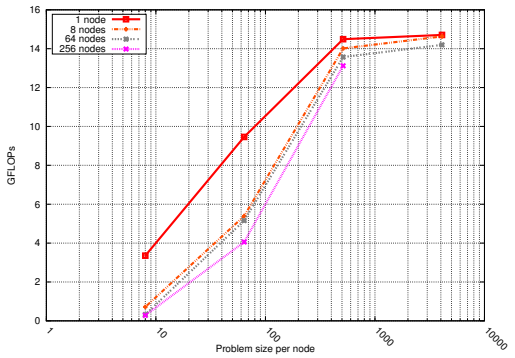
Figure A.6.: Inter-node performance maps for the Laki system for different spatial orders and a fixed temporal order of three.



(a) 2nd-order

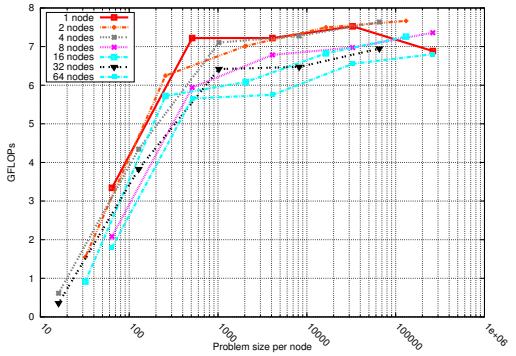


(b) 4th-order

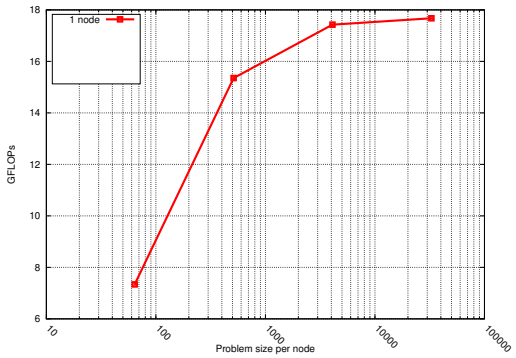


(c) 6th-order

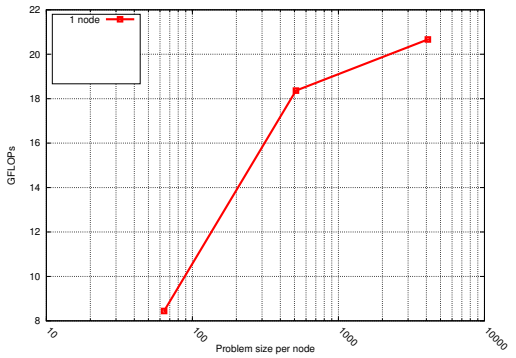
Figure A.7.: Inter-node performance maps for the Juropa system for different spatial orders and a fixed temporal order of three.



(a) 2nd-order



(b) 4th-order



(c) 6th-order

Figure A.8.: Inter-node performance maps for the cluster at the RZ of RWTH Aachen university for different spatial orders and a fixed temporal order of three.

Bibliography

[1] D. J. Bodony. Analysis of sponge zones for computational fluid mechanics. *Journal of Computational Physics*, 212:681–702, 2006.

[2] C. Bogey and C. Bailly. Three–dimensional non–reflective boundary conditions for acoustic simulations: far field formulation and validation test cases. *ACTA Acustica United With Acustica*, 88:463–471, 2002.

[3] C. Bogey and C. Bailly. Large eddy simulations of round free jets using explicit filtering with/without dynamic smagorinsky model. *International Journal of Heat and Fluid Flow*, 27(4):603 – 610, 2006.

[4] J. P. Boris, F. F. Grinstein, E. S. Oran, and R. L. Kolbe. New Insights into Large Eddy Simulation. *Fluid Dynam. Res.*, 10:199–228, 1992.

[5] M. Brachet. Direct simulation of three-dimensional turbulence in the Taylor–Green vortex. *Fluid Dynamics Research*, 8(1-4):1 – 8, 1991.

[6] R. Ewert and W. Schröder. Acoustic perturbation equations based on flow decomposition via source filtering. *Journal of Computational Physics*, 188:365–398, 2003.

[7] G. Gassner, F. Lörcher, and C.-D. Munz. A contribution to the construction of diffusion fluxes for finite volume and discontinuous Galerkin schemes. *J. Comput. Phys.*, 224(2):1049–1063, 2007.

[8] G. Gassner, F. Lörcher, and C.-D. Munz. A discontinuous Galerkin scheme based on a space-time expansion. II. Viscous flow equations in multi dimensions. *J. Sci. Comp.*, 34(3):260–286, 2008.

[9] C.-D. M. Gregor J. Gassner, Florian Hindenlang. *A Runge-Kutta based Discontinuous Galerkin Method with Time Accurate Local Time Stepping*, volume 2 of *Advances in Computational Fluid Dynamics*. World Scientific, 2011.

[10] C. Günther, D. Hartmann, L. Schneiders, M. Meinke, and W. Schröder. A cartesian cut-cell method for sharp moving boundaries. *AIAA Paper*, 2011-3387, 2011.

[11] D. Hartmann, M. Meinke, and W. Schröder. A strictly conservative cartesian cut-cell method for compressible viscous flows on adaptive grids. *Comput. Meth. Appl. Mech. Eng.*, 200:1038–1052, 2011.

[12] S. Hickel. *Implicit turbulence modeling for large-eddy simulation*. PhD thesis, TU Dresden, 2005.

[13] F. Hindenlang, G. Gassner, T. Bolemann, and C. Munz. Unstructured high order grids and their application in discontinuous Galerkin methods. In *Proceedings of the V European Conference on Computational Fluid Dynamics ECCOMAS CFD 2010 J. C. F. Pereira, A. Sequeira and J. M. C. Pereira (Eds),Lisbon, Portugal,14-17 June 2010*, 2010.

[14] T. Hoefler, C. Siebert, and A. Lumsdaine. Scalable communication protocols for dynamic sparse data exchange. In *Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '10, pages 159–168, New York, NY, USA, 2010. ACM.

[15] F. Hu, M. Hussaini, and J. Manthey. Low-dissipation and low-dispersion runge-kutta schemes for computational acoustics. *J. Comput. Phys.*, 124:177–197, 1996.

[16] N. Jarrin, N. Benhamadouche, S. Laurence, and D. Prosser. A synthetic-eddy-method for generating inflow conditions for large-eddy simulations. *Journal of Heat and Fluid Flow*, 27:585–593, 2006.

[17] S. Johansson. High order finite difference operators with the summation by parts property based on DRP schemes. Technical Report 2004-036, it, Aug. 2004.

[18] H. Klimach and S. Roller. Distributed coupling for multi-scale simulations. In P. Ivanyi and B. Topping, editors, *Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*. Civil-Comp Ltd., 2011.

[19] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger. Optimization of All-to-All Communication on the Blue Gene/L Supercomputer. In *Proc. of the 37th International Conference on Parallel Processing*, pages 320–329, Washington, DC, USA, 2008. IEEE Computer Society.

[20] M.-S. Liou and C. J. Steffen. A new flux splitting scheme. *J. Comput. Phys.*, 107:23–39, 1993.

[21] F. Lörcher. *Predictor Corrector DG*. PhD thesis, University of Stuttgart, 2008.

[22] F. Lörcher, G. Gassner, and C.-D. Munz. A discontinuous Galerkin scheme based on a space-time expansion. I. Inviscid compressible flow in one space dimension. *J. Sci. Comp.*, 32(2):175–199, 2007.

[23] F. Lörcher, G. Gassner, and C.-D. Munz. An explicit discontinuous Galerkin scheme with local time-stepping for general unsteady diffusion equations. *J. Comput. Phys.*, 227(11):5649–5670, 2008.

[24] A. Mani. Analysis and optimization of numerical sponge layers as a nonreflecting boundary treatment. *Journal of Computational Physics*, 231:704–716, 2012.

[25] M. Meinke, W. Schröder, E. Krause, and T. Rister. A comparision of second- and sixth-order methods for large-eddy simulations. *Comput. Fluids*, 31:695 – 718, 2002.

[26] S. Miguet and J.-M. Pierson. Heuristics for 1d rectilinear partitioning as a low cost and high quality answer to dynamic load balancing. In B. Hertzberger and P. Sloot, editors, *High-Performance Computing and Networking*, volume 1225 of *Lecture Notes in Computer Science*, pages 550–564. Springer Berlin / Heidelberg, 1997.

[27] A. Pinar and C. Aykanat. Fast optimal load balancing algorithms for 1d partitioning. *Journal of Parallel and Distributed Computing*, 64(8):974 – 996, 2004.

[28] R. Rabenseifner. Optimization of Collective Reduction Operations. In M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, editors, *International Conference on Computational Science*, volume 3036 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2004.

[29] P. Sanders and J. Träff. Parallel Prefix (Scan) Algorithms for MPI. In B. Mohr, J. Träff, J. Worringen, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 4192 of *Lecture Notes in Computer Science*, pages 49–57. Springer Berlin / Heidelberg, 2006.

[30] O. Schönrock. *Numerical Prediction of Flow Induced Noise in Free Jets of High Mach Numbers*. PhD thesis, The University of Stuttgart, 2009.

[31] A. Spille-Kohoff and H.-J. Kaltenbach. Generation of turbulent inflow data with a prescribed shear-stress profile. August 2001. Third AFSOR Conference on DNS and LES.

[32] C. K. W. Tam. Computational aeroacoustics: Issues and methods. *AIAA Journal*, 33:1788–1796, 1995.

[33] C. K. W. Tam. Advances in numerical boundary conditions for computational aeroacoustics. *Journal of Computational Acoustics*, 6:377–402, 1998.

[34] R. Wokoeck, N. Krimmelbein, J. Ortmans, V. Ciobaca, R. Radespiel, and A. Krumbein. RANS Simulations and Experiments on the Stall Behaviour of an Airfoil with Laminar Separation Bubbles. AIAA Paper 2006-0244, 2006. 44th AIAA Aerospace Sciences Meeting and Exhibit.

[35] Q. Zhang, W. Schröder, and M. Meinke. A zonal RANS/LES method to determine the flow over a high-lift configuration. *Comput. Fluids*, 39:1241–1253, 2010.