

TU-Dortmund Fakultät für Mathematik

Abschlussbericht

Studienprojekt-Technomathematik 2015-2016

Studienprojektgruppe Technomathematik

22. März 2016

betreut durch Dipl.-Inf. Markus Geveler und Prof. Dr. Markus
Geveler

Inhaltsverzeichnis

| | |
|--|-----------|
| 1 Motivation und Problembeschreibung | 3 |
| 2 Einleitung | 4 |
| 3 Hardware | 5 |
| 3.1 Hardwareauswahl und Motivation | 5 |
| 3.2 Aufbau des Clusters | 5 |
| 3.2.1 Anordnung der Rechenknoten | 5 |
| 3.2.2 Strom und Netzwerkanbindung | 6 |
| 3.3 Übersichtsseite | 7 |
| 3.3.1 Einrichtung Webserver | 7 |
| 3.3.2 Sammeln der Messdaten | 8 |
| 4 Software | 9 |
| 4.1 Finite-Elemente-Methode (FEM) | 9 |
| 4.2 Löser | 10 |
| 5 Ergebnisse | 12 |

1 Motivation und Problembeschreibung

Im Bereich des wissenschaftlichen Hochleistungsrechnen gibt es seit 1993 die „TOP500 List“, welche die Supercomputer, geordnet nach ihrer Leistung in Floating Point Operationen pro Sekunde auflistet. Der momentane Spitzenreiter, der Tianhe-2, erreicht dabei bei voller Auslastung einen Energieverbrauch von 155.998,08 MWh jährlich. Dieser extrem hohe Wert bedeutet gleichzeitig große Betriebskosten, welche im Zeitalter der Energiewende kaum zu rechtfertigen sind. Daher wurde als Konkurrenzobjekt im Jahr 2007 die „Green500 List“ eröffnet, welche die Supercomputer entsprechend ihrer Rechenleistung pro Watt sortiert. Und genau an dieser setzt das Studienprojekt Technomathematik 2015/2016 der TU Dortmund an, welches begleitend zum I.C.A.R.U.S.-Projekt des Lehrstuhl 3 der Fakultät Mathematik der TU Dortmund läuft. Ziel ist es, die Hardware- und Softwareentwicklung eines energieeffizienten *High-Performance Computing* (HPC) Systems nachzuvollziehen und zu begleiten. Grundlage dafür ist der mobile Grafikprozessor NVIDIA Tegra K1, welcher höchste Energieeffizienz verspricht und, nach dem Vorbild der Energiewende, durch Solarzellen betrieben wird. Die Nutzung dieses mobilen, auch eingebetteten Systems genannt (engl.: „embedded systems“), erklärt sich durch die Anwendungsbereiche dieser Geräte. Sie werden in der Unterhaltungsindustrie, Automobilindustrie oder sogar als Herzschrittmacher in der Medizin eingesetzt. Diese Anwendungsbereiche verlangen ein enorm energiesparsames Verhalten, da die mobile Energiezufuhr über Batterien nicht endlos ist. Daher sind diese Systeme eine attraktive Möglichkeit beim sogenannten *Green Computing*. Das Hochleistungsrechnen spielt in der Mathematik eine große Rolle, da Differentialgleichungen Naturvorgänge verschiedener Art, wie zum Beispiel Schwingungen, Wellenausbreitungen oder Strömungen beschreiben können. Die Lösung dieser Gleichungen erfordert einen, je nach Genauigkeit des Resultats, erheblichen numerischen Aufwand, welcher nur mit Hilfe von geeigneter Hardware erreichbar ist. Da es das Vorhaben unseres Studienprojekts ist, den Luftstrom und die Wärmeleitung um und in einem geometrischen Körper wie dem des Tegra K1-Boards zu simulieren, spielen Differentialgleichungen eine wichtige Rolle für uns. Außerdem ist es dafür wichtig, die Software nach dem Paradigma der hardware-orientierten Numerik zu konstruieren, um die Leistung des Grafikprozessors effizient auszunutzen. Dazu werden 60 der NVIDIA Boards zu drei Racks angeordnet, um den Code parallelisiert laufen lassen zu können. Man kann also im Umkehrschluss auch davon sprechen, dass die Hardware der Software angepasst wird, also *software-orientierte Hardware*.

Der Aufbau des Hauptteils des Berichts lässt sich dabei in drei große Themen gliedern. Zunächst die Hardware in seinen Einzelteilen, also die Hardwareauswahl, der Aufbau des Racks und die Energieverbrauchsmessung erklärt. Darauffolgend werden die Softwarekomponenten, welche die Assemblierung mit finiten Elementen und finiten Differenzen, sowie den Löser umfassen, erläutert. Abschließend werden verschiedene Testergebnisse, welche mit der Anordnung des Studienprojekts erzielt wurden, vorgestellt und die Rechenleistung anhand dieser Resultate bewertet.

2 Einleitung

Im Bereich der hardware-orientierten Numerik wurden bereits in den vergangenen Jahren einige Untersuchungen gemacht. So wurde in den Jahren 2010 und 2011 durch Geveler et al. die Lattice-Boltzmann-Methode zur Lösung von Navier-Stokes- und Flachwassergleichungen auf unterschiedlicher Hardware analysiert. Dabei stellte sich heraus, dass GPUs Multi-Core CPUs mit einem Speedup von bis zu acht überlegen sind, ohne dabei an Genauigkeit der numerischen Lösung einzubüßen. In „Efficient Finite Element Geometric Multigrid Solvers for Unstructured Grids on GPUs“ wurde bereits 2011 nach dem Paradigma der hardware-orientierten Numerik ein geometrisches Mehrgitterverfahren mit finiter Elemente Assemblierung entwickelt, welches nur aus einer Reihe von Sparse Matrix-Vektor Multiplikationen besteht. Durch diese Implementierung, welche keinen Leistungsverlust nach sich zog, war es möglich, die Parallelität von Rechenarchitekturen noch besser auszunutzen. Besonders durch die Verwendung von GPUs statt Multi-Core CPUs ergab sich auch hier ein durchschnittlicher Speedup von acht. Geveler et al. betrachtete 2013 in „Towards a complete FEM-based simulation toolkit on GPUs: Geometric Multigrid solver“ die Lösung partieller Differentialgleichungen mit finiten Elementen und Mehrgitterverfahren auf unstrukturierten Gittern. Es wurde gezeigt, dass sich die Laufzeit der Anwendung erheblich reduzieren lässt, sobald GPUs anstatt von Multi-Core CPUs benutzt wurden. Des Weiteren wurde in „Energy efficiency vs. Performance of the numerical solution of PDEs: An application study on a low-power ARM-based cluster“ ein Cluster aus 96 ARM Cortex-A9 Dual-Core Prozessoren einer Rechenarchitektur, basierend auf x86-Prozessoren gegenüber gestellt. Dabei wurde die verbrauchte Energie zur Lösung von drei wissenschaftlichen Anwendungen, einem Finite-Elemente Code mit Mehrgitter-Löser, einer strömungsmechanischen Anwendung und einem Code zur Ausbreitung von Schallwellen mit Hilfe der Spektral-Elemente Methode analysiert. Schließlich wurde gezeigt, dass die verbrauchte Energie zur Lösung erheblich gesenkt werden kann, im Einklang mit einer akzeptablen Erhöhung der Laufzeit. In „Porting FEASTFLOW to the Intel Xeon Phi: Lessons Learned“ zeigten Geveler et al. die Leistungsverbesserung von axpy-Vektor Operationen und Sparse Matrix-Vektor Multiplikationen durch Nutzung des Intel Xeon Phi Koprozessors. Diese Analyse, sowie die in „FFF2: Future-proof High Performance Numerical Simulation for CFD with FEASTFLOW (2)“ durchgeführten Untersuchungen, welche sich mit der Entwicklung von numerischen Methoden zur parallelen Lösung von partiellen Differentialgleichungen für realitätsnahe industrielle und wissenschaftliche Probleme befassen, basierten auf der Software Infrastruktur „FEASTFLOW“. Dieses Paket umfasst Software zur numerischen Lösung der Navier-Stokes-Gleichungen in 2D und 3D.

3 Hardware

Der zweite wichtige Aspekt beim Aufbau eines unkonventionellen Supercomputers ist der physikalische Aufbau des Computers. Hierbei muss man darauf achten ein Gleichgewicht zwischen den Faktoren Energieverbrauch, Rechengeschwindigkeit, und Kühlung zu finden. Zusätzlich soll unabhängig von der Größe des Clusters eine einfache Lösung für das Monitoring bereitgestellt werden.

3.1 Hardwareauswahl und Motivation

HIER NOCH INHALT EINFÜGEN! (wenn nicht redundant mit der Einleitung)

3.2 Aufbau des Clusters

Für ein Jetson-TK1-Cluster mit insulärer Stromversorgung ist das Design des Racks der entscheidende Punkt um auch unter dauerhafter Höchstbelastung eine stabile Funktion der Rechenknoten, zu gewährleisten. Durch die Verwendung eines Lithium-Ion-Akkus zur Energiespeicherung und das Risiko eines Kurzschlusses durch Kondensationswasser muss ein geschickt aufgebautes Rechencluster gewährleisten, dass die Temperatur der gesamten Anlage in dem Bereich von 18°C bis 40°C gehalten wird.

3.2.1 Anordnung der Rechenknoten

Hinsichtlich der oben beschriebenen Problematik ist die Anordnung der Knoten das zentrale Instrument mit dem man die Bildung potenzieller Wärmenester gegen den Platzverbrauch des Clusters abwägen kann.

Wegen zu hoher Energiekosten muss auf den Einsatz einer Wasserkühlung verzichtet werden, um eine möglichst hohe Energieeffizienz zu erreichen. Die einfachsten Ansätze zum Aufbau des Supercomputers, welche sich mittels Luftkühlung umsetzen lassen, sind:

- 1) Rechenknoten in horizontalen Schichten anzurichten
- 2) Rechenknoten vertikal anzurichten und nebeneinander aufzustellen

Je nach Aufbau ergeben sich dabei einige Vor- und Nachteile für den Großrechner, welche hier anhand der Wärmebildern gezeigt werden.

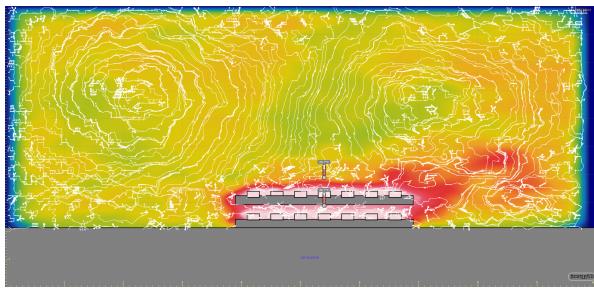


Abbildung 1: Horizontaler Aufbau

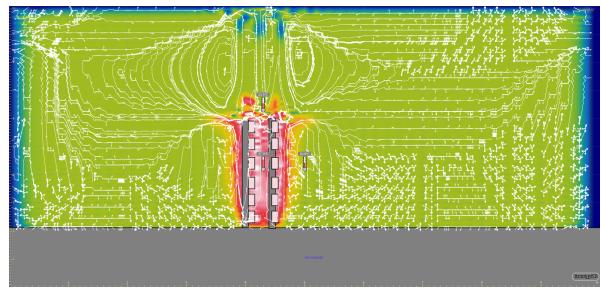


Abbildung 2: Vertikaler Aufbau

Vergleicht man diese beiden Ansätze, so sieht man, dass im horizontalen Szenario (1) zwischen den Platten, auf denen die Rechenknoten angebracht sind, Wärmenester entstehen. Zudem liefert die Analyse des Stromlinien-Diagramms, dass besonders in der Mitte der Platten die Wärme nur schlecht abtransportiert werden kann.

Im Gegensatz dazu ist der Wärmeabtransport beim vertikalem Ansatz (2) durch aufsteigende warme Luft besser möglich. Hierbei sollte allerdings bemerkt werden, dass durch die schiefe Lage des Ventilators die Gravitationskraft Unregelmäßigkeiten bei der Rotation verursachen würde, was schlussendlich zu einer verkürzten Lebensdauer des Ventilators führen könnte.

Um die Vorteile der beiden Ansätze zu kombinieren und gleichzeitig für Wartbarkeit des Rechenclusters zu sorgen, wählt man einen Ansatz, bei dem die Boards in einer Doppelhelix-Struktur angeordnet werden.

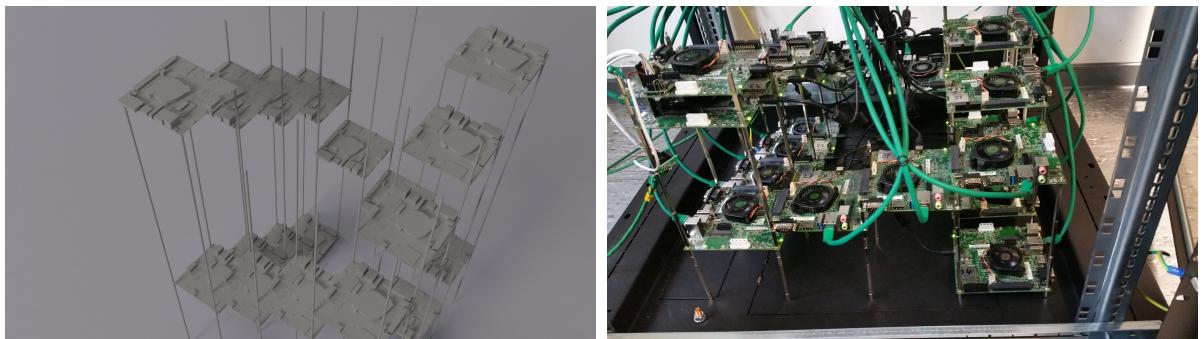


Abbildung 3: Render des Serveraufbaus

Abbildung 4: Serveraufbau (7. März 2016)

Wie auf diesen Abbildungen (3 , 4) zu erkennen ist, wird durch die Verwendung der Doppelhelix-Struktur ein größerer Abstand zwischen den Bords für eine verbesserte Kühlung, bei gleichzeitig geringem Platzverbrauch ermöglicht.

3.2.2 Strom und Netzwerkanbindung

Über die Anordnung hinaus muss nun die Stromversorgung und die Netzwerkanbindung der Bords geregelt werden. Eine besondere Herausforderung hierbei ist die Anordnung der Netzteile, der einzelnen Rechenknoten. Um zu gewährleisten, dass sich durch das Aufheizen der Netzteile keine gefährlichen Wärmenester bilden, wird eine spezielle Halterung verwendet. Unter Nutzung moderner 3D-Druck Technologien, wie sie häufig in der 'Maker-Szene' eingesetzt werden, wurde eine Halterung (5)entworfen und produziert, welche trotz geringem Materialaufwand eine sehr hohe Stabilität und eine gute Luftzufluss ermöglicht.

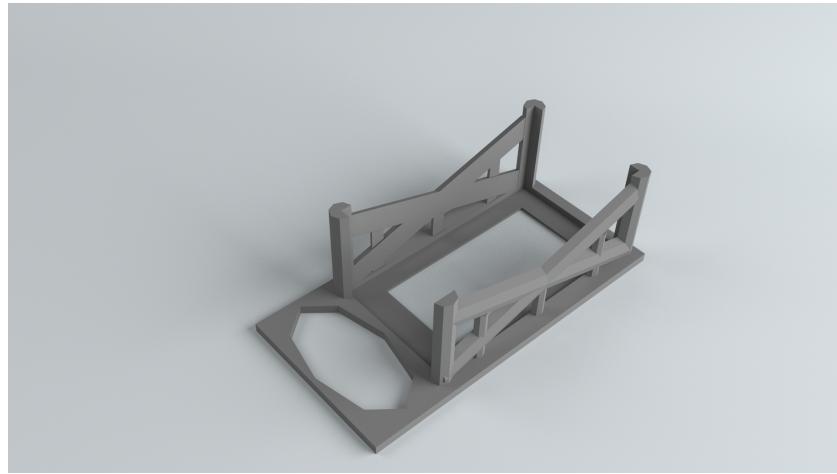


Abbildung 5: Halterung für die Netzteile

Hinsichtlich des Aspektes 'Green-Computing' ist es wichtig, ein Druckfilament zu verwenden, welches biokompatibel ist. Hierfür bietet sich der aus Maisstärke gewonnene Biokunststoff Polylactide (PLA) an. Dieser hat einen Schmelzpunkt von 150°C bis 160°C und ist daher ohne weiteres für die Halterung von Netzteilen verwendbar.

3.3 Übersichtsseite

Um das Monitoring und die Fernwartung des Clusters möglichst komfortabel zu bewerkstelligen wurde ein Dashboard eingerichtet. Hierfür ist ein RaspberryPi der ersten Generation ausreichend.

3.3.1 Einrichtung Webserver

Um die gemessenen Daten verwalten und darstellen zu können wird auf dem RaspberryPi ein LAMP-Server eingerichtet (Linux-Apache-MySQL-PHP). Die Daten werden folglich in der MySQL Datenbank abgespeichert und über PHP-Skripte ausgelesen und verarbeitet. Die Darstellung der Daten erfolgt mittel HTML, CSS und Javascript. Des Weiteren muss der RaspberryPi so konfiguriert werden, dass erstens das Webinterface von außen einsehbar ist , und vor allem, dass man von außen in die Datenbank schreiben kann.



Abbildung 6: Dashboard Frontend

3.3.2 Sammeln der Messdaten

Die Temperaturen können lokal auf den einzelnen Boards ausgelesen werden. Dies wird verwendet um mittels C++ ein Programm laufen zu lassen, welches sich auf den einzelnen Rechenknoten nacheinander einloggt, die Temperatur Daten an verschiedenen Stellen des Jetson-Boards ausliest und anschließend eben jene in die MySQL-Datenbank auf dem Webserver einfügt.

Um die restringierte Hardware des Raspberry Pi's nicht zu überlasten wird das Programm auf dem Gateway-Knoten des Clusters ausgeführt, weshalb der oben Genannte Zugriff von außen des RaspberryPi's wichtig ist.

Um qualitative Aussagen über die Energieeffizienz machen zu können wurde eine PDU angeschafft, mit welcher man den Stromverbrauch der einzelnen Knoten messen können sollte.

— HIER TEXT WARUM DAS MIT DER PDU NICHT GEHT —

— HIER TEXT BZGL NEUER MESSMETHODEN —

4 Software

4.1 Finite-Elemente-Methode (FEM)

Zur Diskretisierung des Poisson-Problems, das betrachtet wird, kann man mit der implementierten FEM ein LGS $Au = b$ aufstellen, welches zur näherungsweisen Berechnung der Lösung dient.

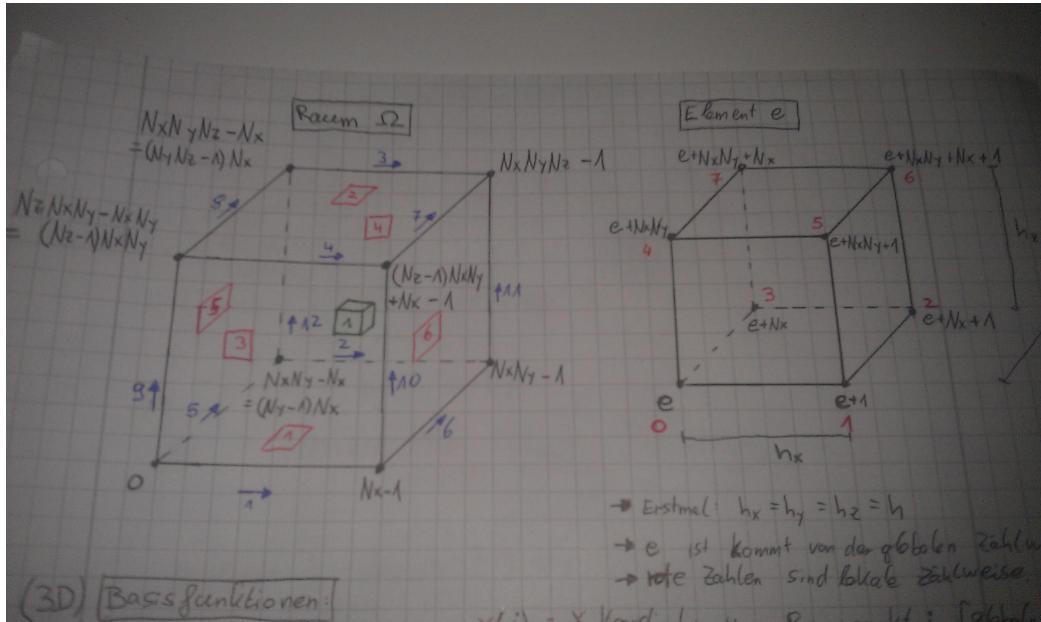


Abbildung 7: Raum Ω und Element mit Nummerierung

Als Gebiet wird ein Quader $\Omega \subset \mathbb{R}^3$ betrachtet, der mit einem äquidistanten Gitter so überdeckt wird, dass eine Triangulierung T_h aus Würfeln mit der Kantenlänge h entsteht.

Die Punkte des Gitters werden nun global durchnummeriert, angefangen mit dem linken, unteren, vorderen Punkt des Quaders, er erhält den Index 0. Man zählt zunächst in x-Richtung (N_x = Anzahl Gitterpunkte in x-Richtung, ebenso N_y und N_z) durch, dann in y-Richtung und dann in z-Richtung. Insgesamt gibt es also $N_x \cdot N_y \cdot N_z$ Gitterpunkte. Der aktuell betrachtete, globale Index erhält die Variable „Zeile“. Diesem Index wird eindeutig ein Element zugewiesen, indem der Punkt des Gitters mit diesem Index der linken unteren, vorderen Ecke des Elementes entspricht. Ebenso muss eine lokale Nummerierung eines einzelnen Elementes vorgenommen werden, diese entnehme man Abbildung 7.

Bevor man nun die Matrix A aufstellen kann, muss man sich mit der Theorie der FEM für das vorliegende Problem beschäftigen. Aus der Poisson-Gleichung erhält man die schwache Formulierung

$$a(u, \varphi) = l(\varphi) \quad \forall \varphi \in V := H_0^1 \quad (1)$$

mit der Bilinearform $a(u, \varphi) = \int_{\Omega} \nabla u \cdot \nabla \varphi \, dx$ und der Linearform $l(\varphi) = \int_{\Omega} f \varphi \, dx$. Anschließend wird die schwache Form mit einem endlich-dimensionalen Raum $V_h \subset V$ diskretisiert. Mit einer Basis $(\varphi_i)_{i=1,\dots,N}$ von V_h und Ausnutzen der Bilinearität ist (4.1) im diskreten Fall äquivalent zu

$$\sum_{i=1}^N u_i a(\varphi_i, \varphi_j) = l(\varphi_j), \quad j = 1, \dots, N, \quad (2)$$

wobei dann $Au = b$ mit $a_{ji} = a(\varphi_i, \varphi_j)$ und $b_j = l(\varphi_j)$.

Es bleiben die Einträge der Systemmatrix und der rechten Seite zu berechnen. Die Basis wird in 3D so gewählt, dass sie auf einem Element trilinear ist. Zur Einbindung von Randbedingungen sind außerdem Basisfunktionen in 2D aufzustellen, diese sind bilinear auf einem Quadrat. Die Basen werden durch den Lagrangeschen Basispolynomansatz gebildet. Bei der FEM koppelt man die Basis φ_i von einem Punkt i mit den Basisfunktionen aller anderen Punkte des Netzes. Aufgrund der Wahl der Basis reicht es jedoch, die direkt umliegenden Punkte zu betrachten, da für alle anderen Punkte j gilt: $\text{supp}(\varphi_i) \cap \text{supp}(\varphi_j) = \emptyset$. Allerdings ergeben sich für Punkte, die auf dem Rand $\partial\Omega$ liegen, Sonderfälle, weil es dort im Gegensatz zu den inneren Punkten weniger umliegende Punkte gibt. Man muss dann unterscheiden, ob sich ein Punkt in einer Ecke, auf einer Kante oder auf einer Seitenfläche des Quaders Ω befindet. Dies erklärt die unterschiedlichen Schleifen in der Implementierung der FEM.

Die Integrale, die zum Aufstellen des LGS bestimmt werden müssen, rechnet man mit einer Quadraturformel aus, die in diesem Fall einer 27-Punkt-Gauss-Formel in 3D und einer 9-Punkt-Gauss-Formel in 2D entspricht (auch hier: zweidimensionale Integrale müssen beispielsweise bei Berücksichtigung der Randbedingungen berechnet werden).

Die Berechnung der Gauss-Punkte und die Auswertung der Funktionen werden auf einem Referenzelement durchgeführt. In 3D entspricht dieses $RE3 = [0, h]^3$ und in 2D $RE2 = [0, h]^2$. Dabei sind die aus diesem Ansatz resultierenden Transformationen vom Referenzelement auf die Elemente der Triangulierung reine Translationen. Die Referenzelemente haben den Nutzen, dass die Gauss-Punkte lediglich ein einziges Mal evaluiert bzw. die Funktionen nur ein Mal ausgewertet werden müssen.

Insgesamt hat die Verwendung der FEM den Vorteil, dass die Assemblierung der Matrix A leicht parallelisierbar ist, indem man die einzelnen Raumpunkte durchläuft, da sich durch einen Raumpunkt eine gesamte Zeile des LGS berechnen lässt. Des Weiteren lassen sich, falls bei einem Problem vorhanden, Neumann-Randbedingungen wesentlich einfacher einbinden als z.B. bei den Finiten Differenzen.

4.2 Löser

Das als Löser verwendete Verfahren des Projektes ist das sogenannte BiCGStab-Verfahren (engl: „biconjugate gradient stabilized method“), welches zur Klasse der Krylov-

Unterraum-Verfahren zählt. Diese iterative Methode wurde von H.A. van der Horst als Löser für nicht-symmetrische lineare Gleichungssysteme entwickelt und benötigt keine zusätzlichen Anforderungen an die Systemmatrix A wie zum Beispiel das CG-Verfahren. Sein Name ist davon abgeleitet, dass die im nicht-vorkonditionierten Algorithmus verwendeten Residuen biorthogonal sind, also $(r_i, \hat{r}_j) = 0 \forall i \neq j$, und die Suchrichtungen bikonjugiert sind bezüglich der Systemmatrix A , also $(A p_i, \hat{p}_j) = 0 \forall i \neq j$. Man verwendet dieses Verfahren, da das konkurrierende GMRES-Verfahren weniger spechereffizient arbeitet und das BiCGStab-Verfahren leichter zu implementieren ist. Außerdem kann es mit beliebigem Vorkonditionierer und ohne Vorkonditionierer implementiert werden. Der Nachteil des angewendeten Verfahrens ist, dass die Konvergenz nicht allgemein bewiesen ist.

1. $r_0 = b - Ax_0$
 2. $\hat{r}_0 = r_0$
 3. $\rho_0 = \alpha = \omega_0 = 1$
 4. $v_0 = p_0 = 0$
 5. for $i=1,2,\dots$
 6. $\rho_i = (\hat{r}_0, r_{i-1})$
 7. $\beta = (\rho_i / \rho_{i-1})(\alpha / \omega_{i-1} - 1)$
 8. $p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1}v_{i-1})$
 9. $y = K^{-1}p_i$
 10. $v_i = Ay$
 11. $\alpha = \rho_i / (\hat{r}_0, v_i)$
 12. $s = r_{i-1} - \alpha v_i$
 13. Wenn $\|s\| < TOL$, setze $x_i = x_{i-1} + \alpha p_i$ und beende
 14. $z = K^{-1}s$
 15. $t = Az$
 16. $\omega_i = (K_1^{-1}t, K_1^{-1}s) / (K_1^{-1}t, K_1^{-1}t)$
 17. $x_i = x_{i-1} + \alpha y + \omega_i z$
 18. $r_i = s - \omega_i t$
-

Der von rechts vorkonditionierte BiCGSTAB-Algorithmus startet ausgehend von einem Anfangsvektor x_0 , welcher immer als Nullvektor implementiert wurde. Des Weiteren wurde die Toleranz mit $TOL = 10^{-9}$ festgelegt. Der Vorkonditionierer hat dabei die allgemeine Form: $K = K_1 K_2 \approx A$

5 Ergebnisse