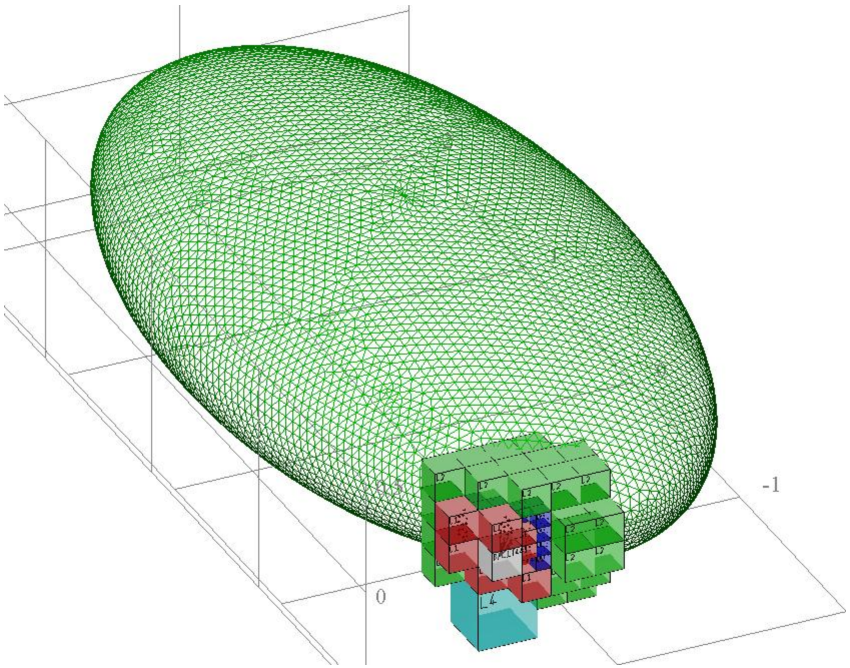
BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
Beuth University of Applied Sciences Berlin

in Kooperation mit dem FWG (Forschungsbereich für Wasserschall und Geophysik) der WTD 71, Kiel

Performance-Optimierung und Grenzen eines Multi-Level Fast Multipole Algorithmus für akustische Berechnungen



Ralf Burgschweiger, Martin Ochmann, Ingo Schäfer und Bodo Nolte

DAGA 2012, 19.-22. März 2012, Darmstadt

Zum Inhalt

Die Multi-Level Fast Multipole Methode (MLFMM) ermöglicht die numerische Berechnung akustischer Problemstellungen auf Basis der Randelementemethode (BEM), bei denen die diskretisierten Modelle aus sehr großen Anzahlen von Elementen bestehen.

Lösungszeit und Speicherbedarf liegen im Vergleich mit konventionellen Lösungsmethoden in der Regel deutlich niedriger, da ein potentialbasierendes Clustering-Verfahren zur approximativen Berechnung der für iterative Löser benötigten Matrix-Vektor-Produkte verwendet wird.

Im Rahmen eines Forschungsprojekts wurde ein zuvor entwickelter Code, basierend auf einer Multi-Level/Single-Order-Variante des Algorithmus, auf eine Multi-Level/Adaptive-Order-Version mit adaptiver Interpolation erweitert und hinsichtlich Lösungsqualität, Parallelisierbarkeit sowie Performance untersucht und optimiert.

Vortragsinhalte:

- Grundlagen der Multi Level Fast Multipole Method (MLFMM)
- Adaptive Variante
- Ergebnisse / offene Fragen

Fast Multipole Methode (MLFMM)

Die Multi-Level Fast-Multipol-Methode stellt ein Verfahren zur beschleunigten Bildung eines Matrix-Vektor-Produktes (MVP) dar, ohne dabei die Matrix vollständig bilden zu müssen.

Sie eignet sich daher in Verbindung mit iterativen Lösern (hier: GMRES) zum Einsatz bei „großen" Gleichungssystemen, bei denen die Wechselwirkungen zwischen vielen Quell- (N_x) und Zielpunkten (N_y) berücksichtigt werden müssen.

Aus Abb. 1 und 2 lässt sich die Abnahme der Anzahl der so zu berücksichtigenden Interaktionen gut erkennen.

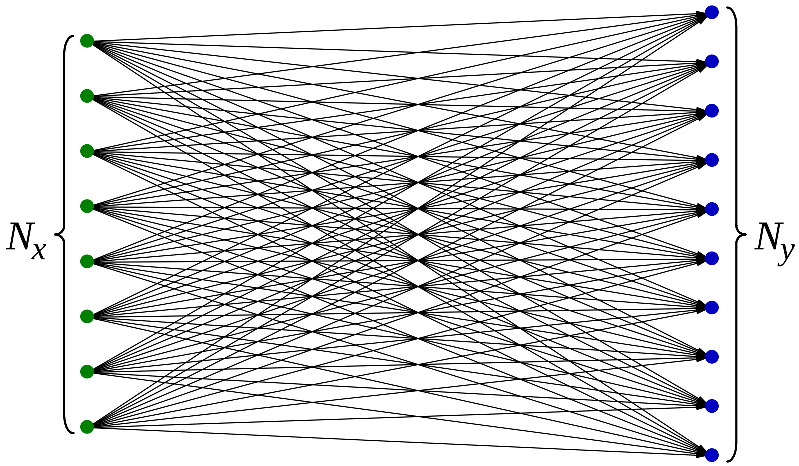


Abb. 1: Interaktionen zwischen Quell- und Zielpunkten bei konventioneller Berechnung

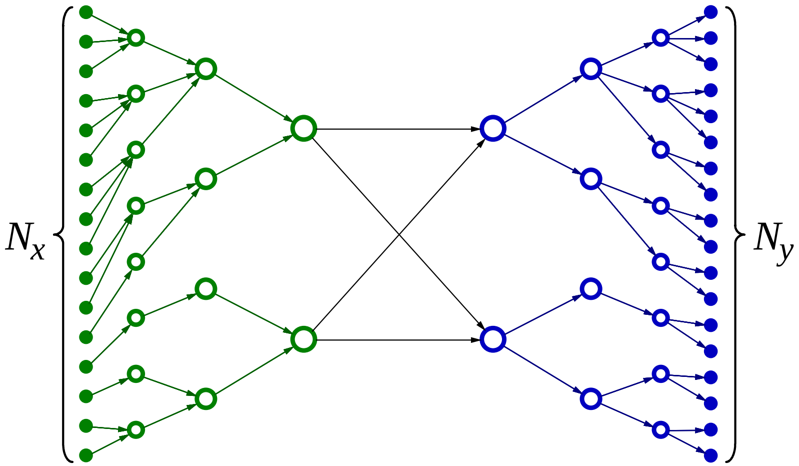


Abb. 2: Cluster-basierende Interaktionen bei der MLFMM (für einen maximalen Cluster-Level von $N_{lc,max} = 3$)

Grundlagen der MLFMM

Die Wirkungen einzelner Quellen x_i werden innerhalb eines „Quell"-Clusters C_x mit dem Radius R zu einer Multipol-Quelle am Punkt z_x zusammengefasst.

Deren Potential wird zu einem entfernten „Ziel"-Cluster C_y mit dem Mittelpunkt z_y transformiert und dort auf die Zielpunkte y_j verteilt (Abb. 3).

Grün: Quellpunkte bzw. –Cluster
Blau: Zielpunkte bzw. -Cluster

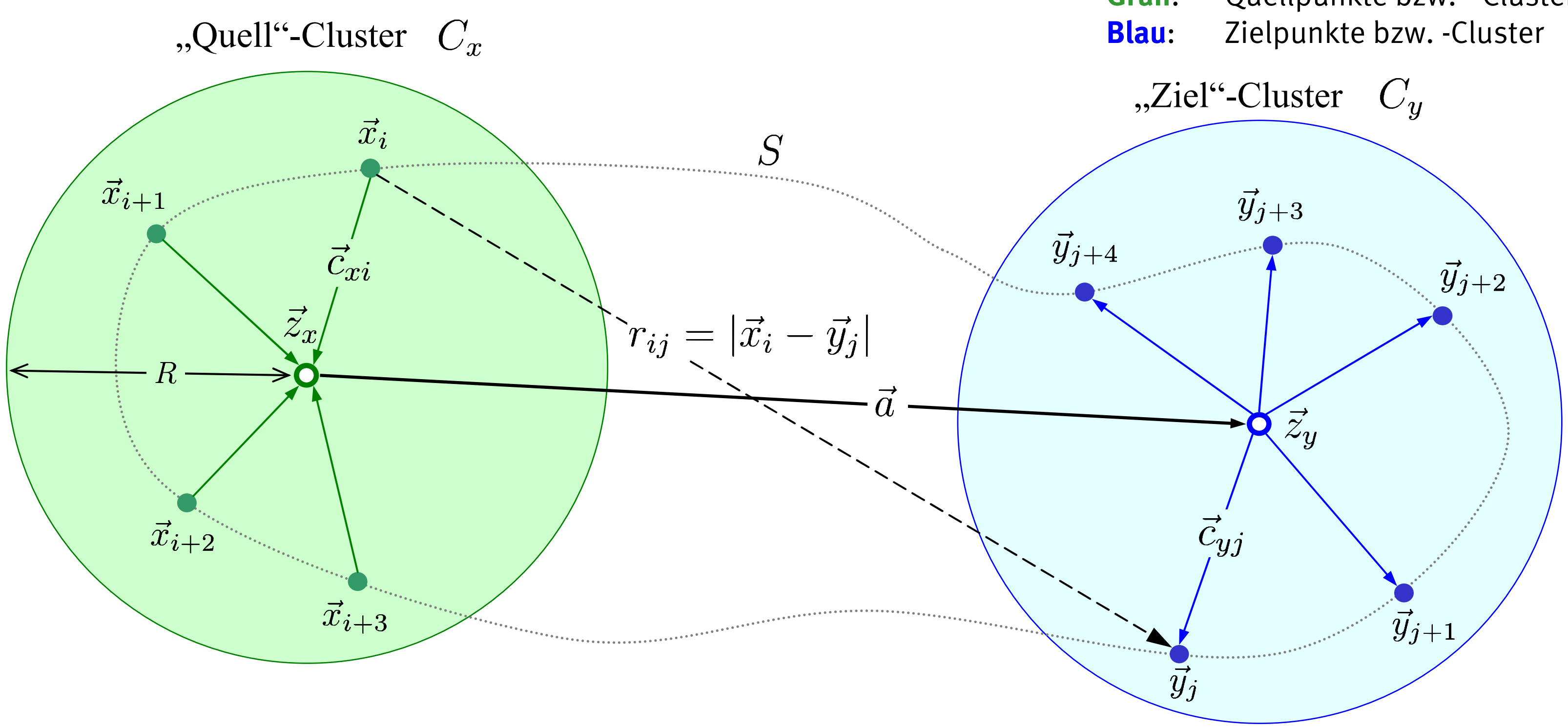


Abb. 3: Zerlegung des Weges zwischen Quell- und Zielpunkten

Grundlagen der MLFMM

Für jeden Teil des Wegs zwischen dem Quell- und Zielpunkt wird eine entsprechende Übertragungsfunktion (h, μ^M und f) verwendet (Abb. 4).

Die vollständigen Details hierzu sind z.B. bei **[4**, T. Sakuma, S. Schneider and Y. Yasuda: *Fast Solution Methods*, Kap. 12, in *Computational Acoustics of Noise Propagation in Fluids*, 2008, Springer Verlag**]** zu finden.

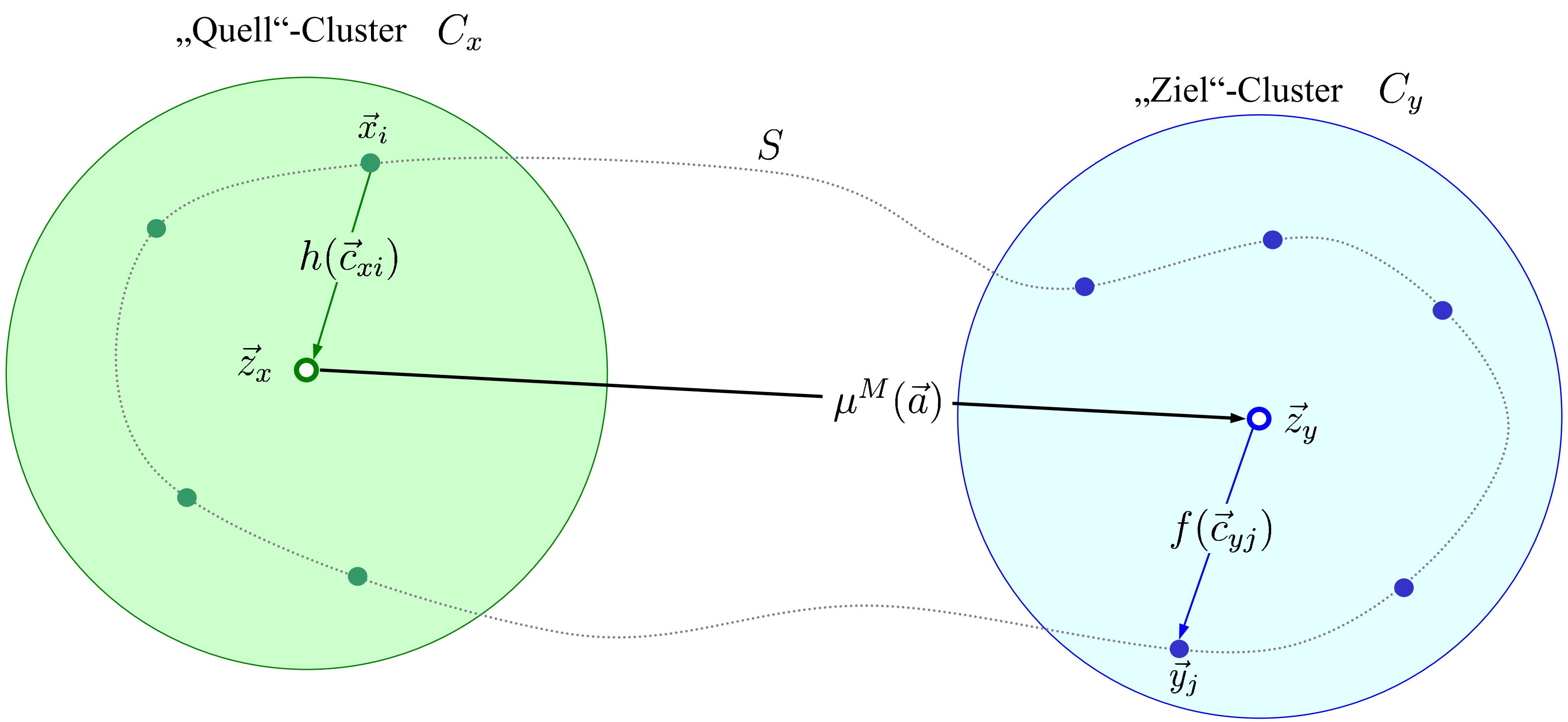


Abb. 4: wegabhängige Übertragungsfunktionen h, μ^M und f

Grundlagen der MLFMM

Eine der kritischsten Funktionen hierbei ist der Translations-Operator μ^M, der das Multipol-Potential (die sog. Signaturen) zwischen den Clusterzentren überträgt.

Dieser Operator kann als Reihenentwicklung mit einer maximalen Entwicklungsordnung O_{mp} („Multipol-Ordnung“) dargestellt werden:

$$\mu^M(\vec{a}, \hat{s}) = \frac{1}{4\pi} \sum_{l=1}^{O_{mp}} (2l+1)\, \mathrm{i}^l\, h_l(k\,|\vec{a}|)\, P_l(\hat{s}\cdot\hat{a})$$

mit

$\vec{a}$	Abstandvektor zwischen den Cluster-Zentren
$\hat{s}$	Vektor der Einheitskugel (Anzahl i. d. R.: $N_{us} = 2\,(O_{mp}+1)^2$)
k	Wellenzahl
$h_l(k\,\|\vec{a}\|)$	Hankel-Funktion l-ter Ordnung
$P_l(\hat{s}\cdot\hat{a})$	Legendre-Polynom l-ter Ordnung

Ergebnisse

In Vorprojekten wurde zunächst eine Version des MLFMM-Algorithmus unter Verwendung einer fest vorgegebenen Ordnung [1, 2, 4] implementiert (Vs. 1.x), um die Grenzen hinsichtlich der Elementanzahlen und Frequenzobergrenze ermitteln zu können.

Zwischenzeitlich wurde der Algorithmus zu einer adaptiven Version (Vs. 2.x) auf Grundlage von [5, 6, 8] ausgebaut, in dem die Multipol-Ordnung als Funktion von Cluster-Abstand und Wellenzahl definiert und zur Laufzeit berechnet wird.

Zusätzlich wurden weitere Optimierungen zur Verkürzung der Lösungszeit vorgenommen, vor allem bei der Vorausberechnung und Parallelisierung.

Beispiele:

- Schallharte Kugel (Rechenzeiten und Hardwarelimits)
- Modell eines U-Bootes (MVP-Abweichungen)
- Ellipsoid (Entwicklungsordnungen)

Ergebnisse – schallharte Kugel

Für dieses Beispiel wurde eine schallharte Kugel (r = 0,5 m) in Wasser mit einer einfallenden ebenen Welle bei einer Frequenz f = 1 kHz verwendet.

Allg. Größen		Direkter Löser (Intel MKL)	Iterativer Löser (GMRES)		Fast Multipol Methode, feste Ordnung, O_{mp} = 6 (GMRES/MLFMM, Vs. 1.030)			
N_{elem}	s_{mx}	Δt_{IMKL}	Δt_{GMRES}	N_{iter}	Δt_{FMM}	N_{iter}	Δt_{MVP0}	$\Delta t_{\oslash MVP}$
1.000	16 MB	0,20 s	0,16 s	12	0,64 s	13	0,093 s	0,045 s
2.500	90 MB	1,23 s	0,73 s	11	0,62 s	12	0,141 s	0,042 s
5.000	0,6 GB	7,44 s	3,03 s	11	2,06 s	13	0,328 s	0,142 s
10.000	1,5 GB	43,99 s	11,92 s	11	2,59 s	12	0,562 s	0,180 s
20.000	6,1 GB	310,75 s	52,56 s	11	7,63 s	13	1,202 s	0,530 s
50.000	38 GB	4.352,43 s	329,37 s	11	15,13 s	12	3,433 s	1,053 s
100.000	153 GB	n.v.	*11.710,95 s	11	31,87 s	12	5,975 s	2,331 s
200.000	612 GB	n.v.	*46.635,42 s	11	57,16 s	12	12,823 s	3,989 s
500.000	3,8 TB	n.v.	n.v.		138,26 s	12	28,205 s	9,869 s
1.000.000	15,4 TB	n.v.	n.v.		373,61 s	12	68,500 s	27,596 s
2.000.000	61,2 TB	n.v.	n.v.		561,82 s	12	114,973 s	40,332 s
5.000.000	383 TB	n.v.	n.v.		1.922,26 s	13	315,325 s	133,179 s

Tab. 1: Ergebnisse zur schallharten Kugel mit MLFMM (Vs. 1.030)
System: Dual Intel XEON E5550, 2,66 GHz, 8 CPU-Kerne, 96 GB DDR-3 RAM

Ergebnisse – schallharte Kugel

Für dieses Beispiel wurde eine schallharte Kugel (r = 0,5 m) in Wasser mit einer einfallenden ebenen Welle bei einer Frequenz f = 1 kHz verwendet (O_{mp} = 6).

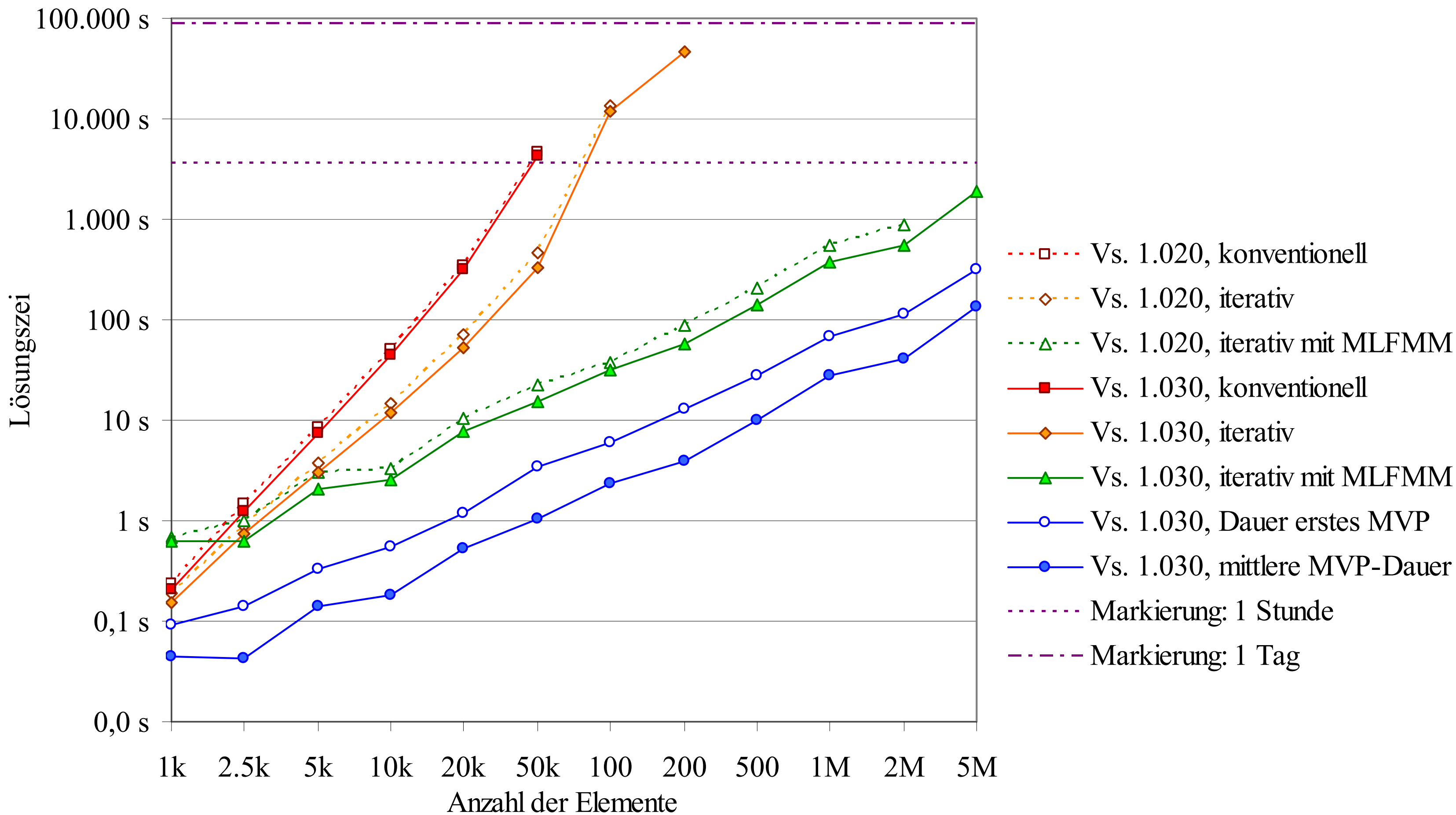


Abb. 5: Ergebnisse zur schallharten Kugel mit MLFMM (Vs. 1.020 und Vs. 1.030)

Ergebnisse – schallharte Kugel

Durch die Umstellung auf die adaptive Version, Optimierungen der Vorabberechnungen und Parallelisierung weiterer Schritte konnte die Lösungzeit auf bis zu 40% gegenüber der Vs. 1.030 (mit fester Ordnung) reduziert werden (Tab. 2/ Abb. 6).

	MLFMM, feste Ordnung, $O_{mp} = 6$ (Vs. 1.030, 8 cores, Windows)		MLFMM, adaptive Ordnung, $O_{mp} = 6 \ldots 10$ (Vs. 2.030, 12 cores, LINUX)			
N_{elem}	Δt_{FMM}	N_{iter}	Δt_{FMM}	N_{iter}	Δt_{MVP0}	$\Delta t_{\oslash MVP}$
1.000	0,64 s	13	0,29 s	13	0,071 s	0,015 s
2.500	0,62 s	12	0,42 s	13	0,120 s	0,024 s
5.000	2,06 s	13	0,91 s	12	0,319 s	0,047 s
10.000	2,59 s	12	1,33 s	12	0,560 s	0,067 s
20.000	7,63 s	13	2,57 s	12	1,115 s	0,127 s
50.000	15,13 s	12	7,38 s	12	4,037 s	0,291 s
100.000	31,87 s	12	12,79 s	12	6,557 s	0,543 s
200.000	57,16 s	12	25,88 s	12	13,630 s	1,083 s
500.000	138,26 s	12	63,74 s	12	33,053 s	2,661 s
1.000.000	373,61 s	12	134,61 s	12	71,360 s	5,491 s
2.000.000	561,82 s	12	257,12 s	12	140,226 s	10,100 s
5.000.000	1.922,26 s	13	856,73 s	12	532,744 s	25,987 s

Tab. 2: Ergebnisse zur schallharten Kugel mit MLFMM (Vs. 1.030) bzw. adaptiver MLFMM (VS. 2.030)

Ergebnisse – schallharte Kugel

Die Reduzierung der Lösungszeit ist deutlich zu erkennen, ebenso der größere Zeitaufwand für das 1. MVP und der deutlich geringere Zeitaufwand bei allen folgenden MVPs.

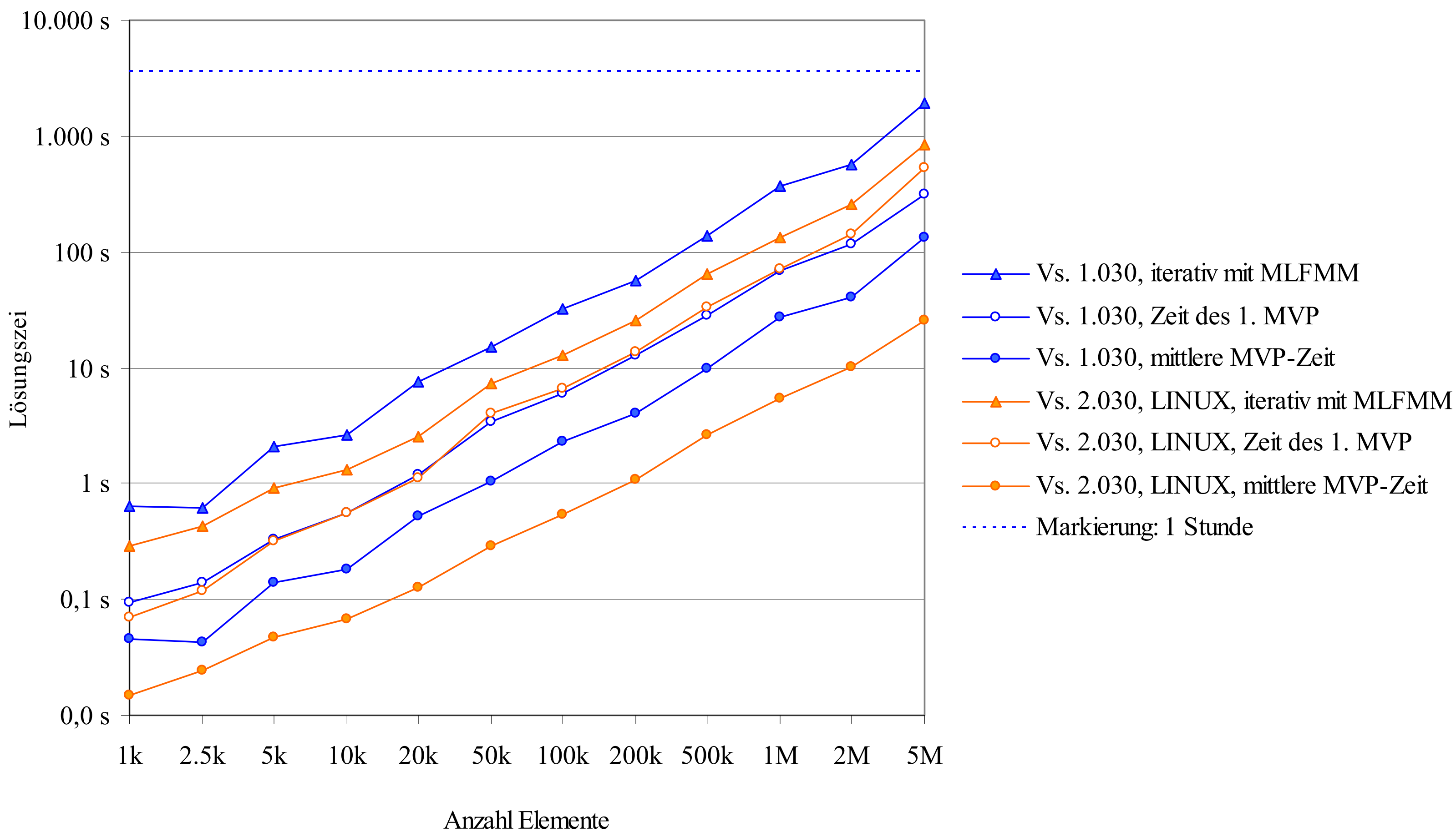


Abb. 6: Ergebnisse zur schallharten Kugel mit MLFMM (Vs. 1.030) bzw. adaptiver MLFMM (VS. 2.030)

Ergebnisse – U-Boot-Modell

Für dieses Beispiel wurde ein einfaches Modell eines U-Boots mit einer Länge von ca. 57 m, bestehend aus 175.000 Elementen, verwendet. Bei einer Frequenz von 250 Hz konnte ein sehr gutes Ergebnis erzielt werden (Abb. 7).

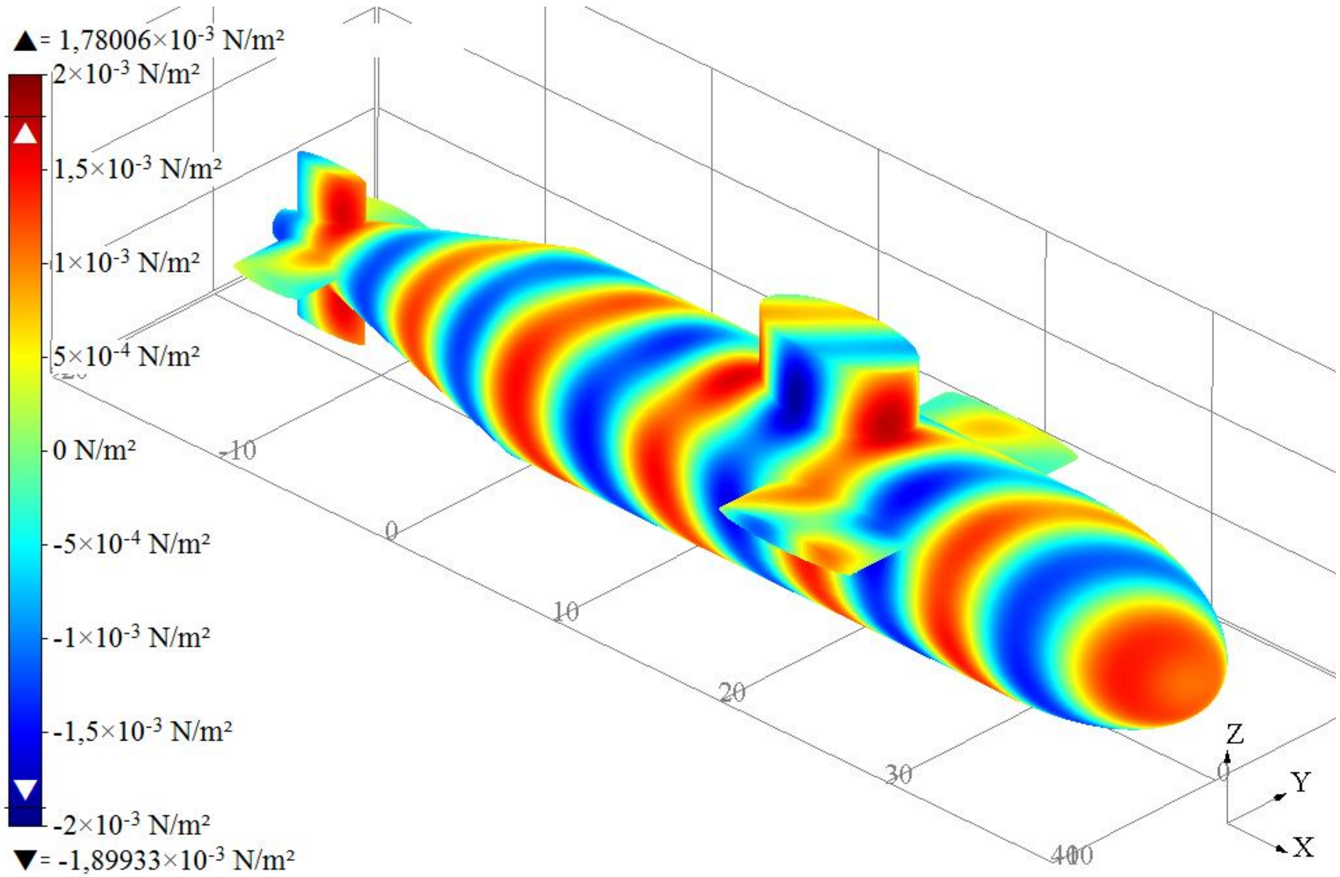


Abb. 7: Modell eines U-Boots, $\mathrm{Re}(p_{surf})$, $N_{elem} = 175.412$, $f = 250$ Hz, $\Delta t_{solve} = 182$ s, $e_{iter} < 10^{-4}$

Ergebnisse – U-Boot-Modell

Die Verdopplung der Frequenz auf 500 Hz führt zu ersten sichtbaren Abweichungen und auch das vorgegebene Fehlerlimit von $e_{iter} < 10^{-4}$ wurde mit 300 Iterationen nicht erreicht (Abb. 8). Diese Abweichungen treten für $O_{mp} > 100$ auf und führen zu kumulativen Fehlern im MVP.

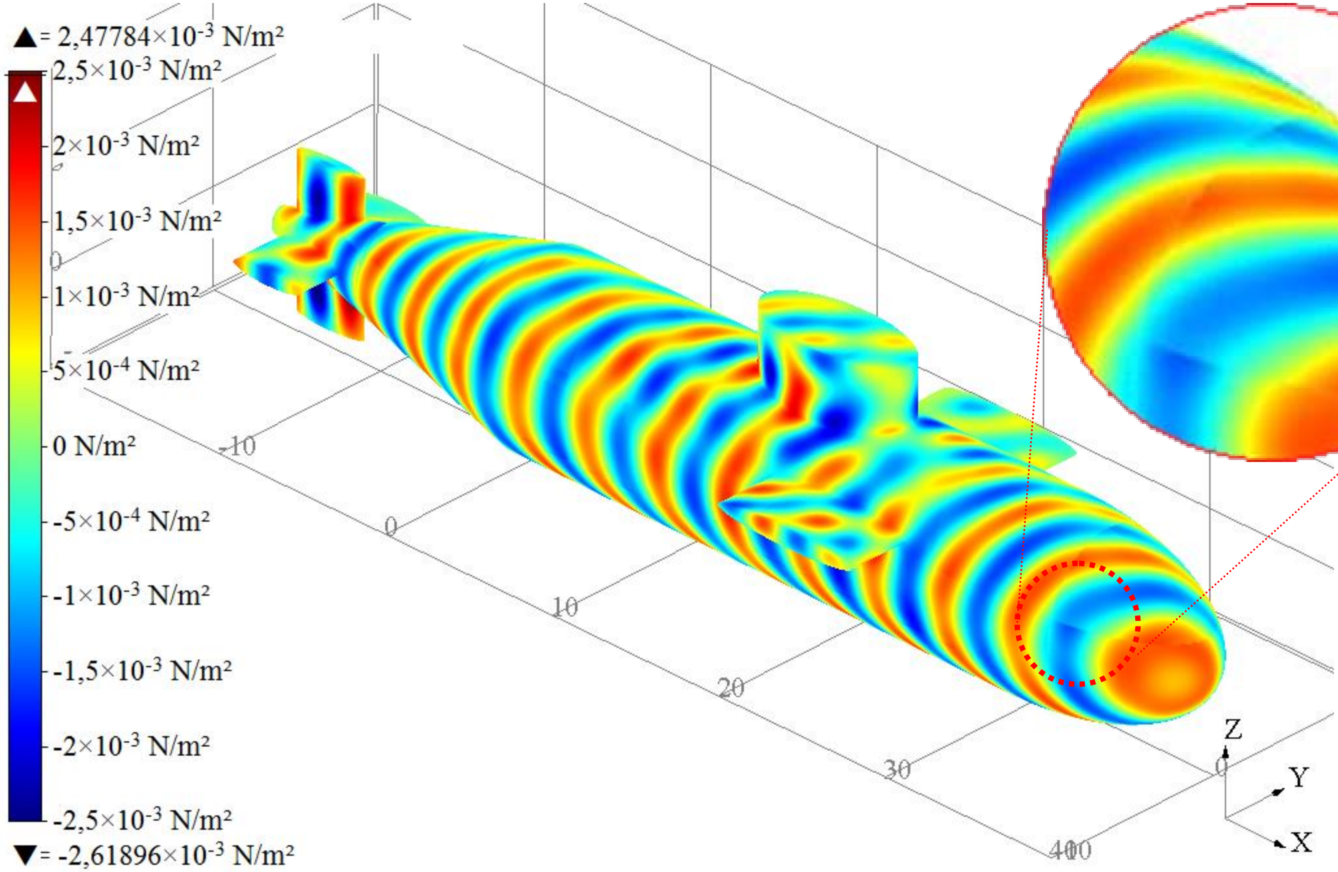


Abb. 8: Modell eines U-Boots, $\mathrm{Re}(p_{surf})$, $N_{elem} = 175.412$, $f = 500$ Hz, $\Delta t_{solve} = 1.165$ s, $e_{iter} \approx 2{,}7\times10^{-4}$

Ergebnisse – U-Boot-Modell

Bei einer Frequenz von 1.000 Hz sind deutliche Abweichungen an den Cluster-Kanten zu erkennen (Abb. 9a), das Resultat ist kaum noch verwendbar. Die maximale Multipol-Ordnung lag hier bei O_{mp} = 266.

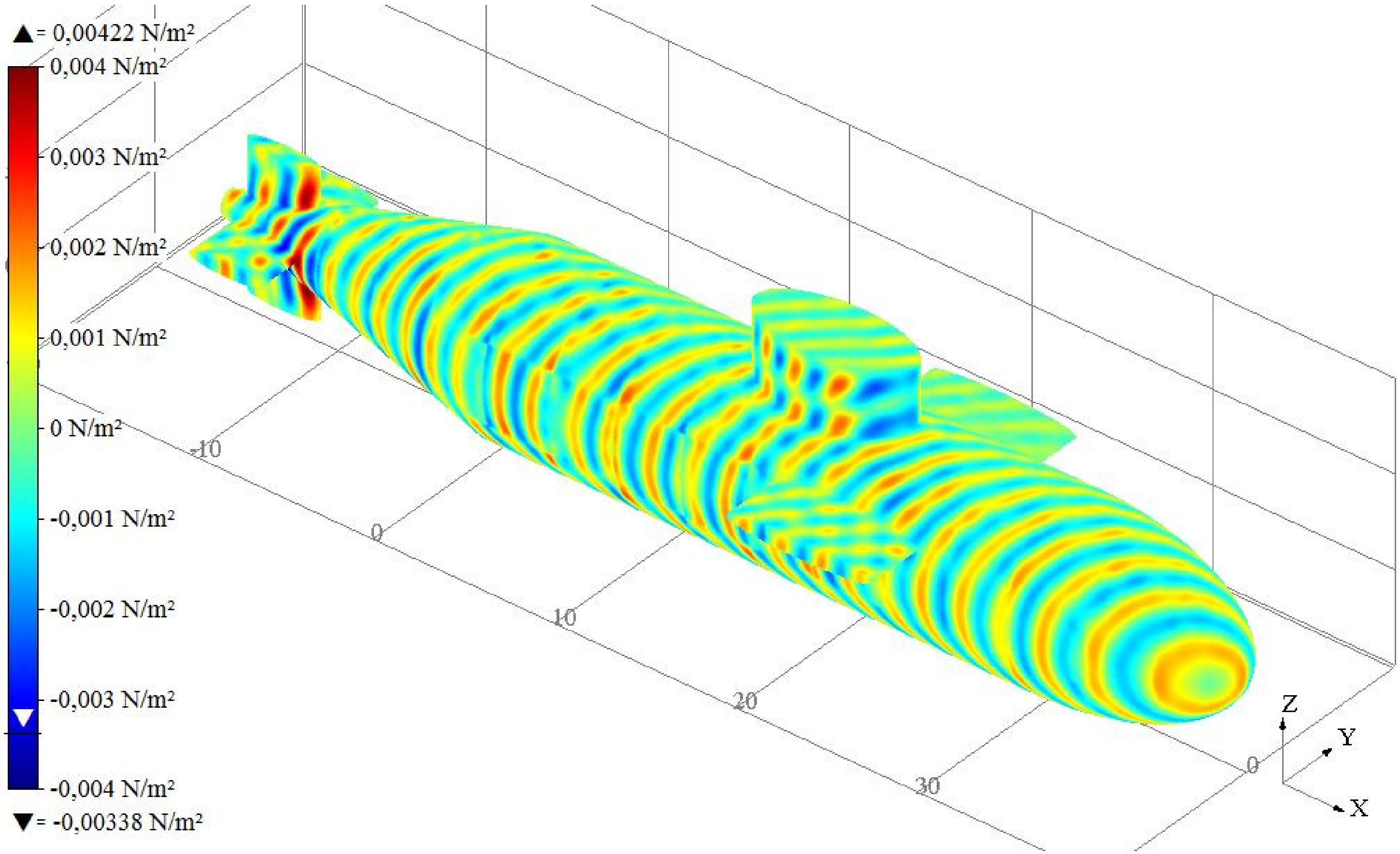


Abb. 9a: Modell eines U-Boots, $\mathrm{Re}(p_{surf})$, N_{elem} = 175.412, f = 1.000 Hz

Ergebnisse – Ellipsoid

Ein weiteres Problem bei Verwendung des Algorithmus mit fester Ordnung besteht darin, dass die Lösung aufgrund des Fehlers in der Matrix-Vektor-Produktbildung nicht mit der erwarteten Lösung übereinstimmt.

Um dieses Problem zu verdeutlichen, wird ein schallhartes Ellipsoid einer Größe von 2×4×1 m (N_{elem} = 17.300) in Wasser durch eine ebene Welle mit einem Einfallswinkel von 30 ° bei einer Frequenz von 2,5 kHz angeregt.

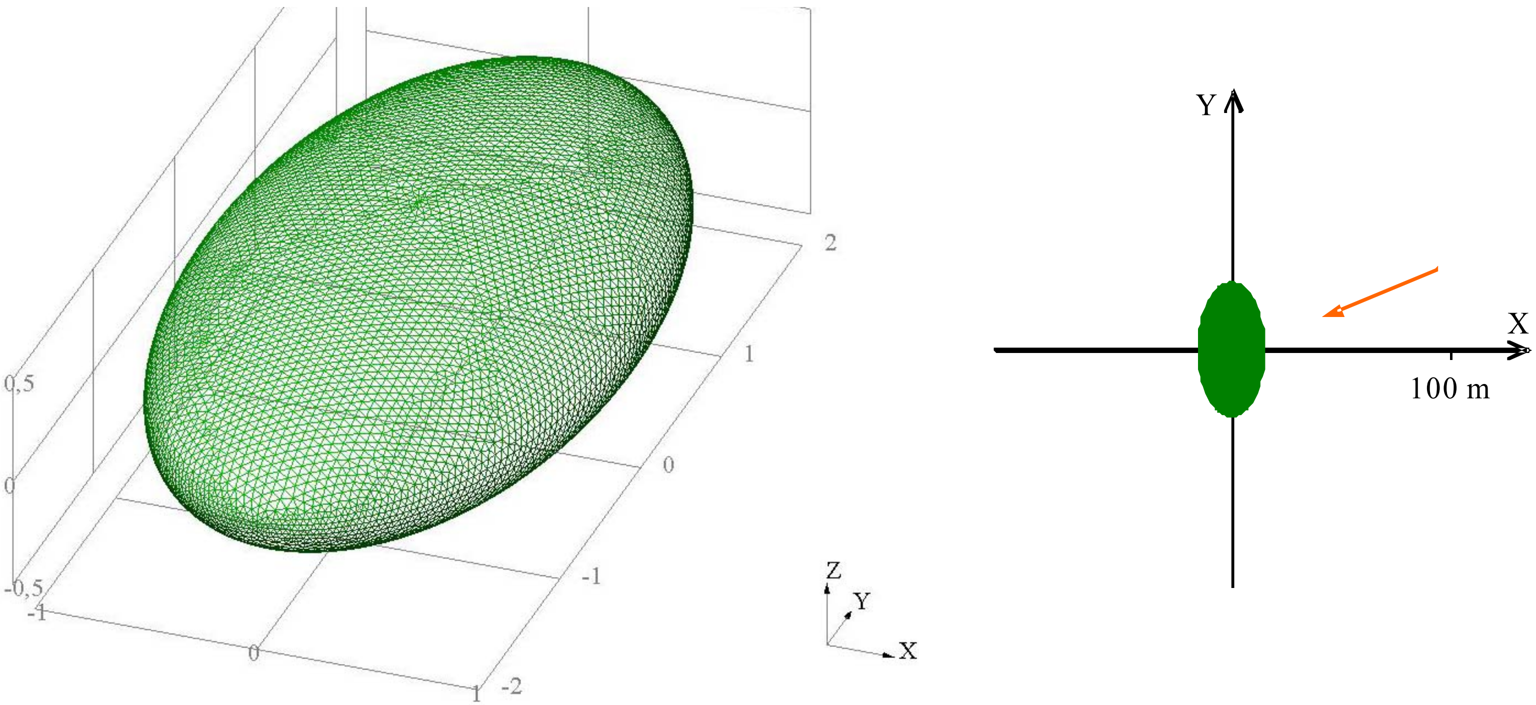


Abb. 9: Ellipsoid (Anordnung)

Ergebnisse – Ellipsoid

Ein weiteres Problem bei Verwendung des Algorithmus mit fester Ordnung besteht darin, dass die Lösung aufgrund des Fehlers in der Matrix-Vektor-Produktbildung nicht mit der erwarteten Lösung übereinstimmt.

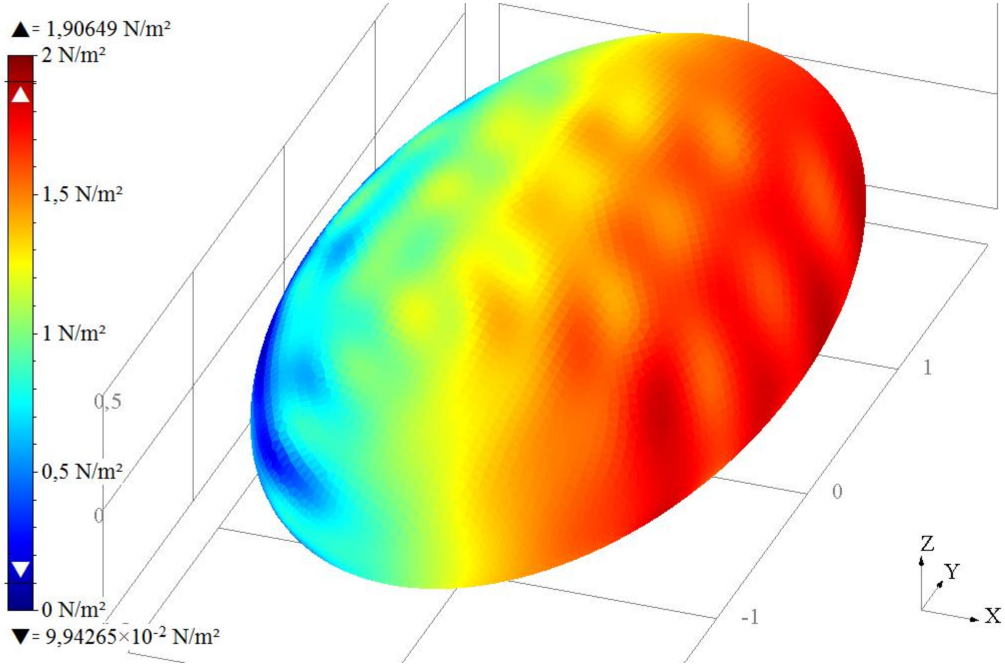


Abb. 10: Ellipsoid, abs(p_{surf}), konventionelle BEM, matrix-basiert

GMRES, matrix-basiert

N_{iter}:	70	
e_{iter}:	$\leq 10^{-6}$	
Δt_{solve}:	37,47	s (Windows)
	30,90	s (LINUX)

direct solver, matrix-basiert

Δt_{solve}:	139,50	s (Windows)
Δt_{solve}:	131,39	s (LINUX)

Ergebnisse – Ellipsoid

Ein weiteres Problem bei Verwendung des Algorithmus mit fester Ordnung besteht darin, dass die Lösung aufgrund des Fehlers in der Matrix-Vektor-Produktbildung nicht mit der erwarteten Lösung übereinstimmt.

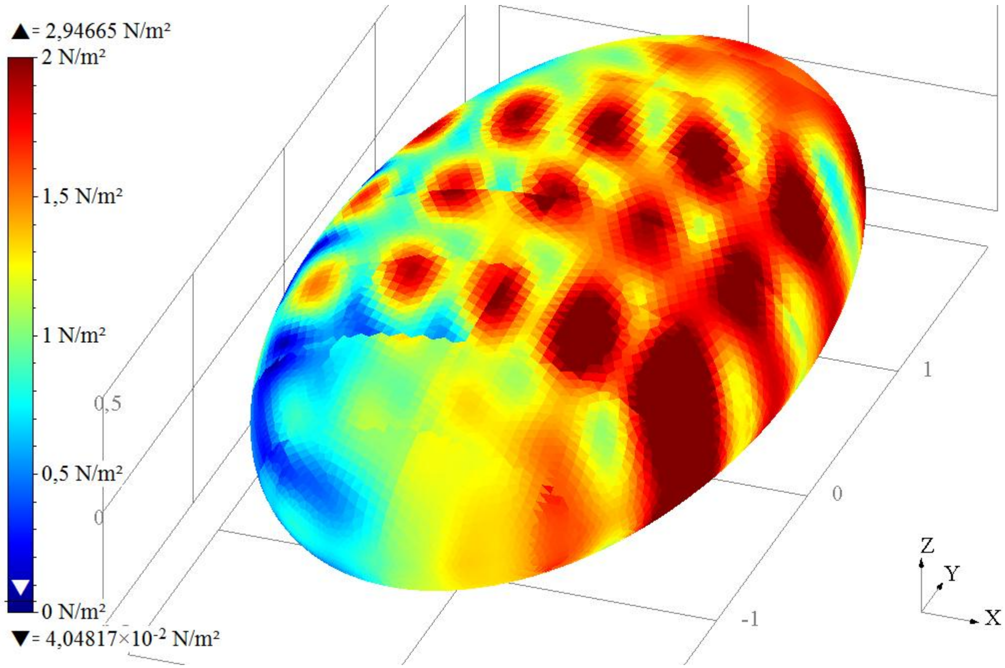


Abb. 11: Ellipsoid, abs(p_{surf}), MLFMM, feste Ordnung $O_{mp} = 6$

GMRES mit MLFMM
feste Multipol-Ordnung

N_{iter}:	67
e_{iter}:	$\leq 10^{-6}$
O_{mp}:	6
Δt_{solve}:	11,06 s (Windows)
	8,67 s (LINUX)

Grosse Differenzen, nicht verwendbar!

Ergebnisse – Ellipsoid

Ein weiteres Problem bei Verwendung des Algorithmus mit fester Ordnung besteht darin, dass die Lösung aufgrund des Fehlers in der Matrix-Vektor-Produktbildung nicht mit der erwarteten Lösung übereinstimmt.

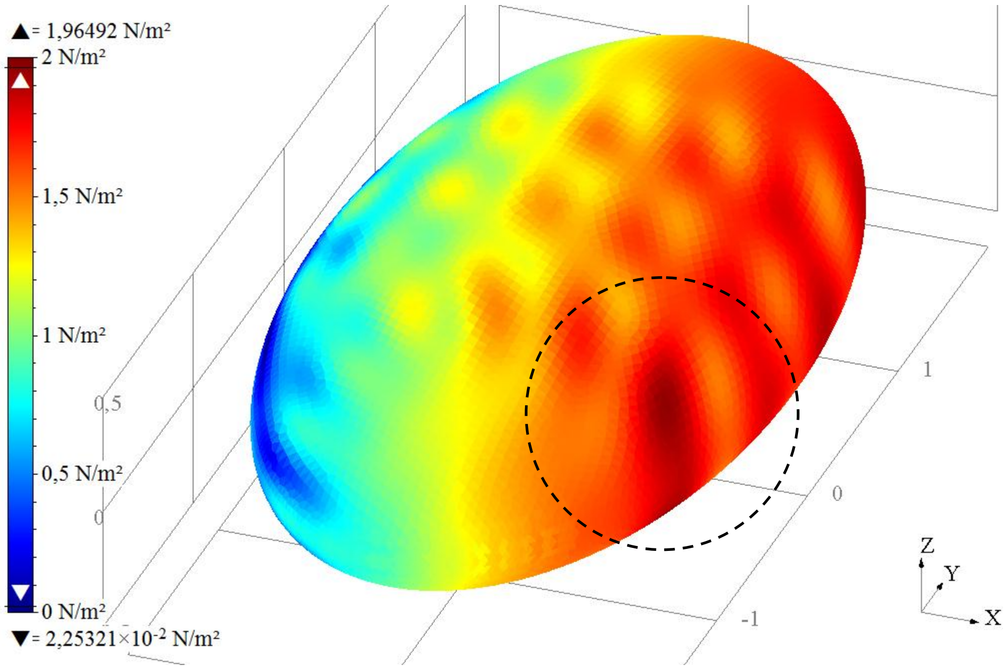


Abb. 12: Ellipsoid, abs(p_{surf}), MLFMM, feste Ordnung $O_{mp} = 15$

GMRES mit MLFMM
feste Multipol-Ordnung

N_{iter}:	71
e_{iter}:	$\leq 10^{-6}$
O_{mp}:	15
Δt_{solve}:	36,93 s (Windows)
	27,56 s (LINUX)

Noch sichtbare Differenzen

Ergebnisse – Ellipsoid

Ein weiteres Problem bei Verwendung des Algorithmus mit fester Ordnung besteht darin, dass die Lösung aufgrund des Fehlers in der Matrix-Vektor-Produktbildung nicht mit der erwarteten Lösung übereinstimmt.

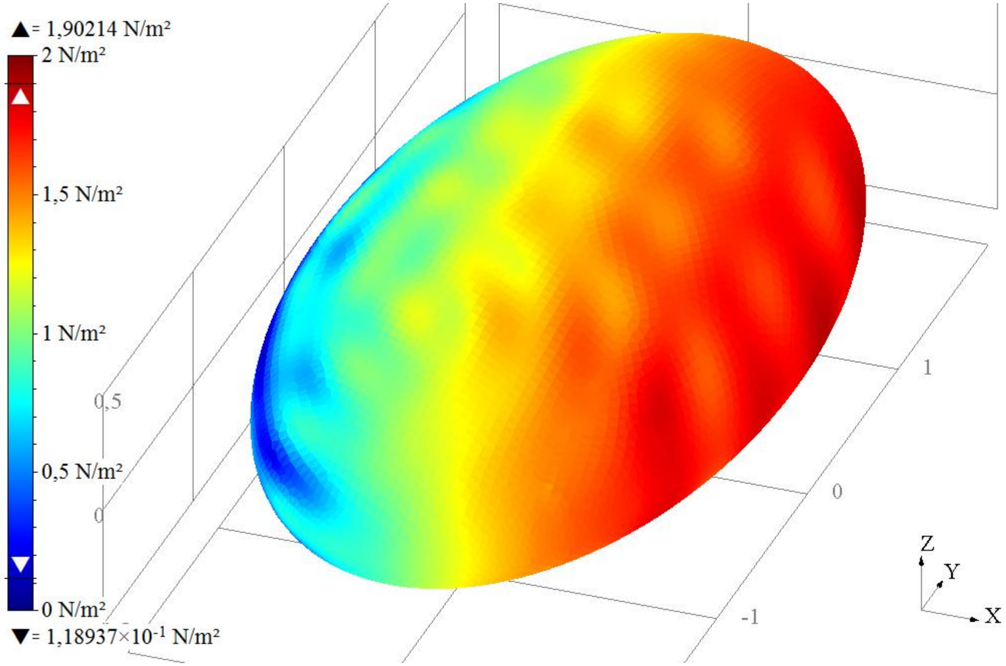


Abb. 12: Ellipsoid, abs(p_{surf}), MLFMM, feste Ordnung $O_{mp} = 15$

GMRES mit MLFMM
adaptive Multipol-Ordnung

N_{iter}:	70
e_{iter}:	$\leq 10^{-6}$
O_{mp}:	6…55
Δt_{solve}:	33,45 s (Windows)
	28,17 s (LINUX)

Sehr gute qualitative und quantitative Übereinstimmung!

Zusammenfassung

- Die vorgestellten Ergebnisse zeigen, dass die adaptive und performance-optimierte Version des MLFMM-Algorithmus eine bessere Qualität und schnellere Lösungen liefert.
- Bei höheren Frequenzen zeigt die MLFMM erhebliche qualitative Unterschiede, da sich der Fehler in der methodenbasierten Matrix-Vektor-Produktbildung verstärkt auswirkt. Hier sind weitere Untersuchungen zur Optimierung des Codes für höhere Multipol-Entwicklungsordnungen notwendig.

Ausblick

- Es sollen geeignete Vorkonditionierungverfahren getestet werden, um eine bessere Konvergenz des iterativen Verfahrens zu erreichen.
  Erste Ergebnisse wurden in **[3]** veröffentlicht, aber diese waren nicht wirklich erfolg-versprechend, sodass hier weitere Arbeiten geplant sind.
- Erste aktuelle Ergebnisse der Kombination des adaptiven MLFMM-Algorithmus mit der Burton-Miller-Methode zeigen aufgrund der besseren Kondition der Systemmatrix eine deutliche Reduktion der Anzahl an Iterationen, sodass hier ein weiterer Schwerpunkt zu setzen ist.

Vielen Dank für Ihre Aufmerksamkeit! ☺

Literaturverweise / empfehlenswerte Links:

[1] R. Burgschweiger, I. Schäfer and M. Ochmann: ”A Multi-Level Fast Multipole Algorithm (MLFMM) for calculating the Sound scattered from Objects within Fluids”, Proceedings of the ICA, 2010, Sydney, Australia

[2] R. Burgschweiger and M. Ochmann: ”A Multi-Level Fast Multipole BEM-Method for computing the sound field in rooms”, Proceedings of the Forum Acusticum, 2011, Aalborg, Denmark

[3] M. Ochmann, R. Burgschweiger and C. Steuck: ”Numerical experiments for testing the convergence of the acoustical Fast Multipole Method”, Proceedings of the 1st EAA Congress on Sound & Vibration (EuroRegio), 2010, Ljubljana

[4] T. Sakuma, S. Schneider and Y. Yasuda: “Fast Solution Methods”, chapter 12, in ”Computational Acoustics of Noise Propagation in Fluids”, 2008, Springer Verlag, S. 333ff

[5] H. Cheng, L. Greengard and V. Rokhlin: “A Fast Adaptive Multipole Algorithm in Three Dimensions”, Journal of Computational Physics, Vol. 153, 1999

[6] T. Sakuma and Y. Yasuda: “Fast Multipole Boundary Element Method for Large-Scale Steady-State Sound Field Analysis, Part I: Setup and Validation”, Acta Acustica united with Acustica, 2002, Vol. 88

[7] Y. Yasuda and T. Sakuma: “Fast Multipole Boundary Element Method for Large-Scale Steady-State Sound Field Analysis, Part II: Examination of Numerical Items”, Acta Acustica, 2003, Vol. 89

[8] H. Cheng, W. Crutchfield, Z. Gimbutas, L. Greengard, J. Ethridge, J. Huang, V. Rokhlin, N. Yarvin and J. Zhao: “A wideband fast multipole method for the Helmholtz equation in three dimensions”, Journal of Computational Physics, Vol. 216, 2006, S. 300-325