Qualitätsprüfung des Parallelisierten PDE-Solvers

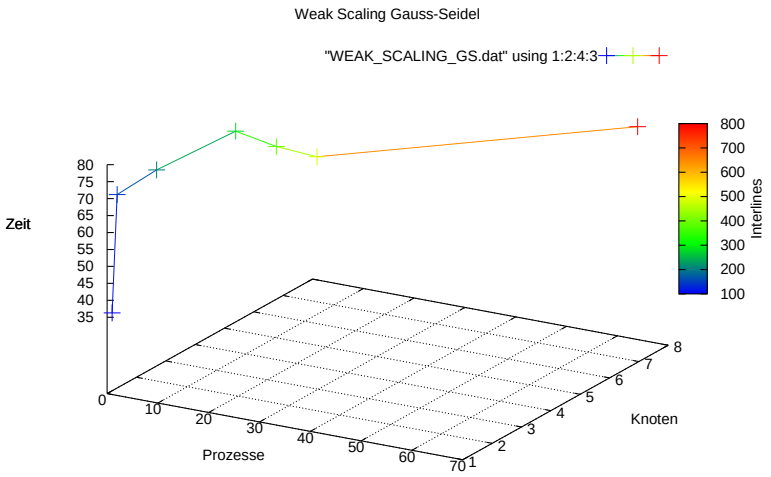
Johannes Timm Guannan Hu

17. Januar 2015

1 Qualitätsprüfung der Implementierung des Gauss-Seidel und Jacobi Iterationsverfahrens

1.1 Weak Scaling

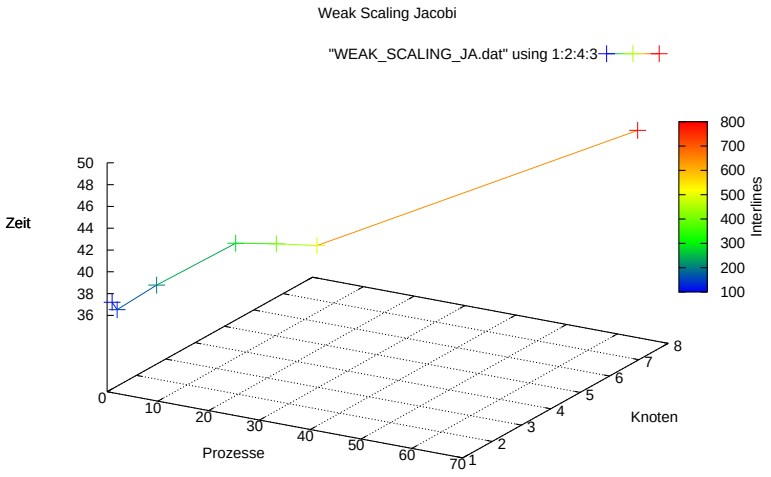
1.1.1 Gauss-Seidel




```
# NPROCS NNODES ILINES TIME
1 1 100 36.5985
2 1 141 71.7535
4 2 200 74.7571
8 4 282 77.6514
16 4 400 75.3382
24 4 490 74.5451
64 8 800 75.2673
```

Diesr Graph zeigt die Anzahl der Prozesse, Knoten, Interlines und der Laufzeit. Diese Tatsache macht eine Interpretation schwierig. Allgemein lässt sich aber beobachten, dass die Schwankungen der Laufzeit, mit Außnahme der Konfiguration (1,1,100), klein sind. Sie liegen im Bereich von 71-77 Sekunden. Die Überragende Geschwindigkeit bei einem Prozess wird durch das Fehlen der Kommunikation (und dem damit verbundenen Overhead) sowie dem Ausnutzen des CPU-Caches zugeschrieben.

1.1.2 Jacobi




```
# NPROCS NNODES ILINES TIME
1 1 100 37.3019
2 1 141 36.7130
4 2 200 37.6343
8 4 282 38.8211
16 4 400 39.4565
24 4 490 39.9848
64 8 800 48.0245
```

Das Jacobi-Verfahren zeigt ein Konträres Verfahren im Vergleich zum Gauss-Seidel-Verfahren. Bis auf die letzte Konfiguration liegen die Ausführungszeiten in einer Größenordung zwischen 36 und 40 Sekunden. Die Letzte Berechnung braucht jedoch 48 Sekunden.

1.1.3 Wahl der Interlines

Prozesse	Matrix Größe	Matrix/Prozess	Interlines
1	809	809	100
2	1137	568,5	141
4	1609	402,25	200
8	2257	282,125	281
16	3209	200,5625	400

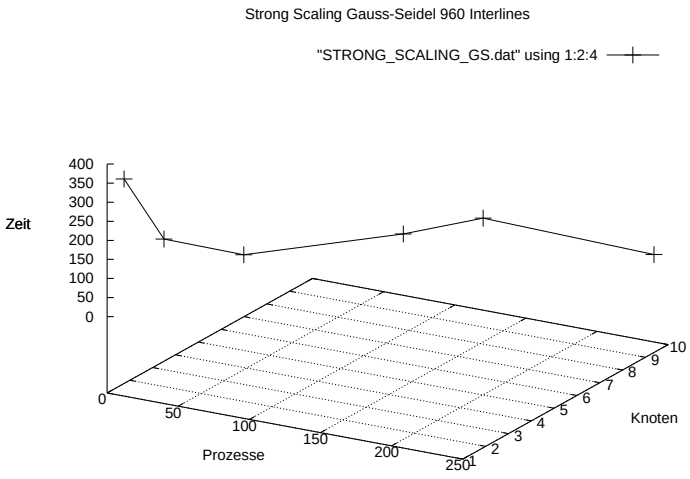
Prozesse	Matrix Größe	Matrix/Prozess	Interlines
24	3929	163,7083333333	490
64	6409	100,140625	800

Tabelle 1: Beziehung zwischen Prozessen und Matrixgröße

Die Wahl der Interlines wurde so getroffen, dass die Matrixgröße mit Anzahl der Prozesse zunimmt, aber die Größe der Matrix pro Prozess abnimmt. Die schnellere Berechnung in jedem Prozess sollte dann für das mehr an Kommunikation kompensieren. Leider sind die Interlines so gewählt, dass eine Lastungleichheit entsteht, da mindestens 1 Prozess eine Zeile mehr oder weniger berechnen braucht als die anderen.

1.2 Strong Scaling

1.2.1 Gauss-Seidel

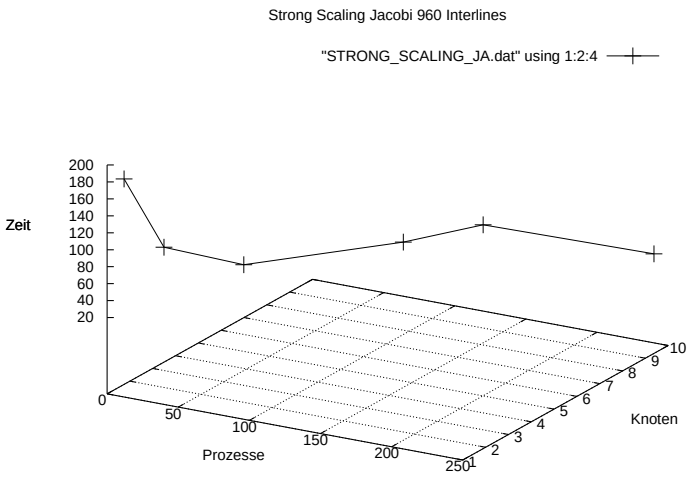



```
# NPROCS NNODES ILINES TIME
12 1 960 369.3752
24 2 960 186.9325
48 4 960 95.6740
96 8 960 50.1567
120 10 960 41.5601
240 10 960 29.3565
```

Diser Graph zeigt Prozesse, Knoten und benötigte Zeit. Es lässt sich beobachten, dass der Speedup im Allgemeinen zwar vorhanden ist, aber selbst mit einer hohen Anzahl an Prozessen schlecht ausfällt.

Auch hier ist eine Interpretation wieder schwierig.

1.2.2 Jacobi

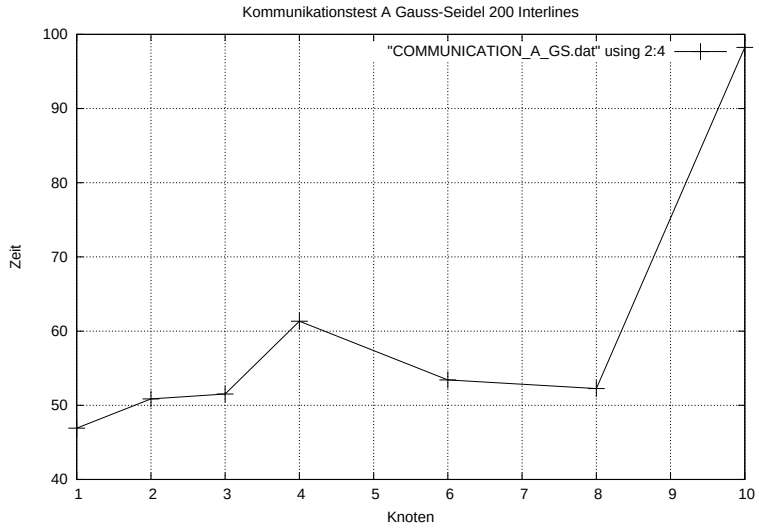



```
# NPROCS NNODES ILINES TIME
12 1 960 187.3134
24 2 960 95.5056
48 4 960 52.3951
96 8 960 33.9367
120 10 960 31.8694
240 10 960 35.1031
```

Dieser Graph stellt die gleichen Daten dar wie der vorherige. Es ist zu beobachten, dass Gauss-Seidel im Vergleich bei einer Hohen Anzahl an Knoten schneller ist als Jacobi. Zudem profitiert Jacobi offenbar nicht so stark von einer erhöhten Anzahl an Prozessen.

1.3 Communication

1.3.1 Gauss-Seidel



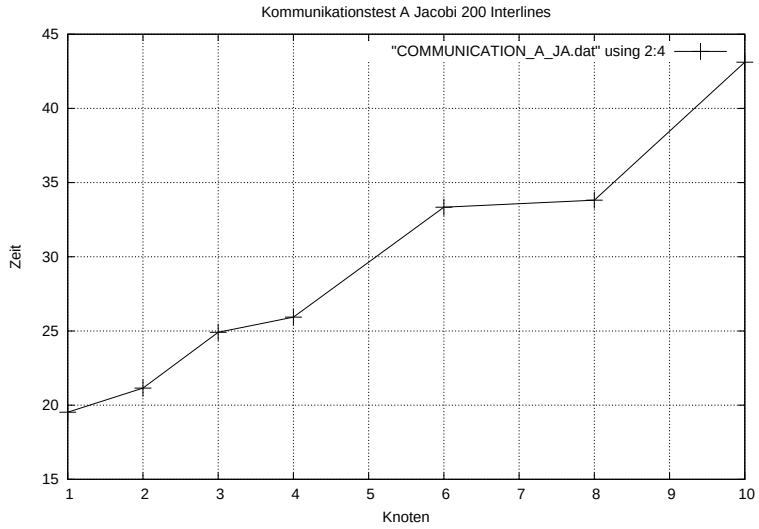

```
# NPROCS NNODES ILINES TIME
10 1 200 46.9201
10 2 200 50.8605
10 3 200 51.5227
10 4 200 61.3365
10 6 200 53.4207
10 8 200 52.2520
10 10 200 98.2490
```

In diesen Graphen sieht die Beziehung zwischen benötigter Zeit und Anzahl der Knoten mit insgesamt 10 Prozessen, auf dem das Programm gleichzeitig ausgeführt wurde. Die benötigte Zeit nimmt tendenziell zu. Abbruch war nach dem Erreichen einer Genauigkeit von $< 3.3504 * 10-5$.

Bei 4 Knoten lässt sich ein lokales Maximum beobachten, bei 6 Knoten ein lokales Minimum und bei 10 Knoten ein globales Maximum.

Der Anstieg der Zeit lässt sich dadurch erklären, dass bei mehr Nodes der Aufwand der Kommunkation im Sinne von nötiger (langsamerer) Hardware wächst. Die Kommunikation innerhalb eines Knotens ist wesentlich schneller, als wenn über Knotengrenzen kommuniziert wird. Dies gilt auch für die Latenz.

1.3.2 Jacobi




```
# NPROCS NNODES ILINES TIME
10 1 200 19.5214
10 2 200 21.1532
10 3 200 24.9100
10 4 200 25.9347
10 6 200 33.3387
10 8 200 33.8153
10 10 200 43.1181
```

Dieser Graph zeigt die gleichen Parameter wie der vorherige bei gleichen Vorraussetungen, abgesehen davon, dass jetzt das Jacobi Verfahren verwendet wird.

Das Verhalten ändert sich maßgeblich. Es gibt jetzt keine klar erkennbaren Minima , sondern nur ein globales Maximum bei 10 Knoten. Dieses Maximum ist aber nur halb sogroß wie das vom Gauss-Seidel Verfahren. Offenbar ist das Jacobi Verfahren nicht ganz so anfällig für Performance Probleme, wie das Gauss-Seidel-Verfahren, wenn es um die Latenz der Kommuikation geht.

1.4 Diskussion

Es zeigt sich im direkten Vergleich, dass das Gauss-Seidel-Verfahren langsamerer ist als das Jacobi- Verfahren. Dies gilt beim Abbruch nach Genauigkeit. Eigentlich sollte dies nicht so sein, da das Gauss-Seidel-Verfahren mathematisch schneller konvergiert. Offenbar ist die Implementierung des Gauss-Seidel-Verfahrens nicht genug getunt, bzw. die vorhandene Optimierung ist für die verwendete Rechner-Architektur nicht optimal. Der Abstand zwischen dem Gauss-Seidel-Verfahren und der hybriden Implementierung des Jacobi-Verfahrens ist erwartbar noch größer. Um diese Performance Lücke zu schließen wird empfohlen, dass Gauss-Seidel-Verfahren zu hybridisieren und dies unter Berücksichtigung der NUMA -Architketur des Rechners. Auch das verbessern der

Cache-Nutzung scheint empfehlenswert. Es ist desweiteren zu prüfen, welche zusätzlichen Compiler-Optionen und Optimierungen hier einen Vorteil bringen. Die Aktivierung von AVX2 und SSE3 Befehlen sollte diesen Programmen Performance Verbesserungen verschaffen. Zur Zeit geht dies jedoch mit Probelmen bei der Nutzung von Debuggern/Memory-Checkern einher. Inwieweit die -O3 Optimierungen einen Vorteil bringen und wie sich die Ergebnisse ändern wurde nicht überprüft. Als erster Schritt sollten unnötige Instruktionen aus dem Gauss-Seidel Code entfernt werden, da der Compiler diese möglicherweise nicht selbst erkennt.