Universität Hamburg
Department Informatik
Wissenschaftliches Rechnen
Dr. J. Kunkel, Hans Ole Hatzel

Übungsblatt 3 zum Praktikum
Paralleles Programmieren
im SoSe 2015
Abgabe: 13.05.2015

Auf diesem Aufgabenzettel finden Sie wieder einige Aufgaben. Sie müssen nicht alle bearbeiten, investieren Sie aber zwischen 5-10 Stunden um sich mit einigen davon zu beschäftigen. Vermutlich genügt es von jeder Sektion eine Aufgabe zu bearbeiten.

1 Speedup Diagramme (90 P, OBLIGATORISCH)

Testen Sie die Skalierbarkeit des zur Verfügung gestellten Beispiel Programmes. Das Beispielprogram wurde für das Cluster compiliert und sollte dort ausgeführt werden. Es genügt hierbei ein Knoten zu reservieren und dort den Speedup zu erfassen. Notieren Sie die erzielten Laufzeiten und Prozesszahlen (bitte mit 1-12 Prozessen testen) in einer Textdatei. Erstellen Sie zudem ein Speedup-Diagramm und ein Effizienz-Diagramm für das Beispielprogramm. (Tipp: `time` zur Messung, `gnuplot` für die Graphen.)
Beschreiben Sie die interessanten Aspekte der beiden Diagramme, gehen sie kurz darauf ein welche Effekte dahinter stehen könnten.

Abgabe:

speedup.pdf mit Ihren Grafiken sowie speedup.txt mit den Rohdaten und Beschreibungen.

2 Projektaufgabe (180 P, OBLIGATORISCH)

Machen Sie sich mit den Projektrichtlinien vertraut.

2.1 Projektrichtlinien

Zum erfolgreichen Abschluß des Praktikums entwickeln Sie im Team eine eigenständig parallele Anwendung. Im Folgenden ein paar Hinweise zum Aufbau der von Ihnen zu erstellenden Artefakte.
Zunächst stellen Sie in einer kurzen Präsentation Ihre gewählte Aufgabe vor:

1. Kurzbeschreibung des Problems
2. Lösungsansatz (sequentielles Programm)
3. Flexibles Parallelisierungsschema (wie teilen Sie die Problemaufgabe auf)

Bei der Programmierung und Auswertung Ihrer Lösung gehen Sie wie folgt vor:

1. Implementation eines sequentielles Programms (oder Sie passen ein bestehendes Programm an). Lassen Sie ihr Programm den Durchsatz an Nutzoperationen pro Sekunde ausgeben.
2. Leistungsanalyse des sequentiellen Programms mit `gprof` und `likwid`. Erreicht Ihr Programm die von Ihnen erwartete Leistung? Evtl. kleine Optimierungen vornehmen.
3. Parallelisierung des Programms mit MPI
4. Optional: Hybride Parallelisierung des MPI-Programms mit OpenMP
5. Erstellung eines Profils und/oder Spurdatenanalyse des Programms. Verbringt Ihr Programm die meiste Zeit in der Bereichnung der Lösung? Optimieren Sie evtl. das Kommunikationsmuster oder erhöhen Sie die Komplexität des sequentiellen Programms.
6. Skalierbarkeit – Erstellen Sie ein Speedup-Diagramm für eine feste Aufgabenstellung

Abschlußpräsentation und Ausarbeitung mit folgendem Inhalt:

1. Problemstellung
2. Lösungsansatz
3. Parallelisierungsschema
4. Laufzeitmessungen
5. Leistungsanalyse
6. Skalierbarkeit
7. Evtl. Grafiken der Programmausgabe

Für die Veröffentlichung Ihrer Ergebnisse benötigen wir:

1. Kurzbeschreibung für die Webseite
2. Source Code als tar.gz
3. Evtl. ein Video (bitte komprimiert)

2.2 Projektproblemstellung

Überlegen Sie sich welche Problemstellung Sie gerne bearbeiten würden. Dokumentieren sie Ihre Ideen. Ziel dieser Aufgabe ist nicht zwingend die finale Projektproblemstellung zu finden, vielmehr ist dies eine Chance erstes Feedback zur Aufgabenstellung zu erhalten und sich Zeit zur Ideenfindung zu nehmen.
Für ihr Projekt müssen Sie sich mit einem Partner zusammenfinden, suchen Sie sich deswegen gerne schon für diese Aufgabe Ihren Partner. Auch eine Abgabe zu dritt ist möglich, es werden dann entsprechend schwierigere und umfangreiche Projekte erwartet. Denken Sie aber daran das es mit steigender Anzahl an Personen zunehmend schwieriger wird effizient zusammenzuarbeiten.

Abgabe:

Eine Textdatei `Problemstellung.txt` mit ausformulierter Problemstellung. Sie dürfen dieses Dokument gerne gemeinsam mit ihrem Projektpartner abgeben.

3 MPI-PGAS

Partitioned Global Adress Space (PGAS), ist ein Model, bei dem jeder Prozess explizit auf Speicher aller anderen Prozesse zugreifen kann. Im Modell mit Nachtrichtenaustausch musste sowohl Sender als auch Empfänger teilnehmen. Bei PGAS kann ein Prozess Daten eines zweiten Prozesses lesen oder schreiben, ohne dass der zweite Prozess Kentniss von der Operation bekommt.
Schauen Sie folgende Operationen an: `mpi_put()`, `mpi_get()`, `mpi_win_create()`, `mpi_win_free()`, `mpi_win_fence()`, `MPI_Accumulate ()`.

3.1 Matrix-Matrix Multiplikation

Programmieren Sie die Matrix-Matrix Multiplikation mit einseitiger Kommunikation unter MPI-2.

3.2 Julia Mengen

Betrachten wir eine Iterationsvorschrift der Form $x_{n+1} = x_n + c$ im Raum der komplexen Zahlen. Wobei x_0 als Initalwert gegeben ist. Ebenfalls sei c gegeben.
Um Julia Mengen für ein gegebenes $c \in \mathbb{C}$ zu visualisieren, kann man auf der X-Achse den Realanteil und auf der Y-Achse den Imaginärteil der Startwerte (x_0) z.B. zwischen -0.5 und 0.5 nutzen. Die Farbwerte an einer Position geben an, wieviele Iterationen gebraucht werden, bis die Iterationsvorschrift divergiert (gegen ∞ geht). Es gibt daneben auch Startwerte, bei denen die Iterationsvorschrift unendlich oft angewendet werden, d.h. es divergiert niemals. Das Program wird daher nach einer Anzahl an Iterationen (maximale Iterationsanzahl) abbrechen und die maximale Iterationszahl für die Position setzen. Um festzustellen dass die Iteration divergiert kann man den Absolutbetrag verwenden. Sobald $|x_n| > 4.0$ geht, so kann man abbrechen, denn die Vorschrift wird divergieren. Die Anzahl der Iterationen werden dann in eine Matrix (entsprechend der Position) gespeichert.
Sequentiellen Beispielcode (in C) für die Julia-Mengen finden Sie auf unserer Webseite. Das Programm erstellt Grafiken (Bitmaps) entsprechend der Parameter.
Schauen Sie z.B. die Grafik an für: `./juliaMengen -0.39 0.6 test.bmp 0 0 0 1000` und `./juliaMengen -0.39 0.6 test.bmp 11 0 0 1000`.
Das Programm kann einerseits mehrfarbige Bilder erzeugen, wobei für jeden Punkt bis zu 2^{24} Iterationen durchgeführt werden müssen (wenn der Absolutwert nicht divergiert). In einem einfarbigen Bild werden nur 2^8 Iterationen durchgeführt.
Im Farbbild kann es erwartungsgemäß sehr lange dauern bis alle 2^{24} Iterationen abgeschlossen sind. Es gibt aber auch Positionen die nach wenigen Iterationen divergieren.
Ein Beispiellauf bei dem nicht alle Punkte divergieren: `./juliaMengen -0.39 0.5 test.bmp 0 0 0 10`.
Beispiele für schöne Parameter finden Sie unter `http://de.wikipedia.org/wiki/Julia-Menge`.

Parallelisierung mittels Gebietszerlegung und PGAS Parallelisieren Sie das Program mittels Gebietszerlegung. Ein Masterprozess sammelt die Iterationszahlen von allen Gebieten in einer Matrix. Nutzen Sie lediglich einseitige MPI-Aufrufe um die Ergebnisse in die Matrix zu schreiben.
Rechnen alle Prozesse immer gleich lange? Wodurch entsteht die Lastungleichheit?

Parallelisierung mit Lastausgleich Parallelisieren Sie das Program mit Lastausgleich. Da es Punkte gibt die erst spät divergieren werden für jeden Punkt bis zu 2^{24} Iterationen durchgeführt. Diese Lastungleichheit wollen wir nun beseitigen.
Verteilen Sie hierzu die zu berechnenden Punkte dynamisch auf die Prozesse. Dazu können Sie einen Masterprozess die Zuteilung übernehmen lassen, dieser Prozess könnte die zu berechnenden Werte Blockweise z.b. in Blöcken von 100 Werten an die Workerknoten übergeben. Sobald ein Prozess seine zugeteilten Werte bearbeitet hat und die Ergebnisse an den Master zurück schickt.
Verwenden Sie beliebige MPI-Aufrufe. Wenn Sie mit dem PGAS Modell weiter programmieren wollen, so benötigen Sie vermutlich `MPI_Win_lock()`.