Universität Hamburg
Department Informatik
Wissenschaftliches Rechnen
Dr. J. Kunkel, O. Pola

Übungsblatt 2 zum Praktikum
Paralleles Programmieren
im SoSe 2017
Abgabe: 02.05.2017

1 Vertiefung von C-Kenntnissen (180 P)

Downloaden Sie auf dem Cluster den C-Workshop Code von
`http://wr.informatik.uni-hamburg.de/teaching/wintersemester_2012_2013/c-workshop`
und entpacken sie die Datei (Tipp: `wget` und `unzip`). Versuchen Sie Lektionen 10-*.c bis 24-*.c durchzuarbeiten und zu verstehen. Es ist nicht wichtig, dass Sie alles auf Anhieb nachprogrammieren können, sondern dass Sie sich ein wenig mit der Programmiersprache C befassen. Notieren Sie sich die dabei entstehenden Fragen.

Online Kurs. Zusätzlich zum Workshop bieten wir den Zugang auf die selbst-entwickelte Plattform ICP[1] an. Der Kurs ist zugreifbar unter `https://oer.wr.informatik.uni-hamburg.de` und erlaubt das Starten von Code im Browser und beinhaltet eine Sammlung von Tests für die einzelnen Aufgabe. In der Plattform werden wir nach und nach Programmieraufgaben bereit stellen. Erstellen Sie sich einen Account und probieren Sie die Plattform aus.

Abgabe:

`1-fragen.txt` Ihre Fragen zum Workshop.

2 Debugging (70 P)

Um später effizient programmieren zu können, wollen wir uns ein wenig mit der Fehlersuche in (parallelen) Programmen beschäftigen. Hierzu schauen wir uns den GNU Debugger `gdb` und den Speicherprüfer `memcheck` von der `valgrind`-Tool-Suite an. Diese können sowohl für sequentielle als auch für parallele Programme genutzt werden. In C ist die Speicherallokation sehr fehlerträchtig, `valgrind` hilft hierbei typische Fehler aufzuspüren.
`http://wr.informatik.uni-hamburg.de/teaching/ressourcen/debugging`

2.1 GNU Debugger

Experimentieren Sie etwas mit GDB herum um den Debugger kennen zu lernen. Im folgenden Vorschläge um GDB kennen zu lernen.
Erstellen Sie ein einfaches Programm mit einer rekursiven Funktion z. B. Fakultät.
Kompilieren Sie das Programm mit Debug-Symbolen und starten Sie es mit GDB.

- Schauen sie sich den Befehl (help) an.
- Erstellen Sie einen Breakpoint auf die rekursive Funktion (break).
- Starten Sie das Programm (run).
- Geben Sie sich den Stack aus.
- Wechseln Sie den Stackframe auf die Main-Routine.
- Zeigen Sie sich den Code an.
- Wechseln Sie zur Funktion zurück.
- Schauen Sie sich einzelne Variablenwerte an (print).
- Welchen Typ haben die Variablen?

[1] Mehr Informationen zum Projekt: `http://www.hoou.de/p/2015/11/26/interaktiver-c-programmierkurs-icp/`

2.2 Valgrind

Schreiben Sie ein einfaches Programm welches ein Array aus Integern verwendet (z. B. 5 Elemente). Lassen Sie jeweils einen Wert des Arrays ausgeben und spielen Sie mit folgenden Konfigurationen:

- Der Wert wurde nicht initalisiert.
- Greifen Sie auf ein ungültiges Element zu z.b. das 6. Element (obwohl der Array nur aus 5 Elementen besteht).
- Schreiben Sie eine Funktion welche den Array (statisch) deklariert, initalisiert und dann den Array zurück gibt. Geben Sie dann in der Main-Funktion einen Wert des zurückgebenen Arrays aus.
- Passen Sie die Funktion an: Allokieren Sie Speicher mit `malloc` (in C) bzw. `allocate` (in Fortran). Starten Sie hier Valgrind mit "–leak-check".

Starten Sie die Programme mit Valgrind, was gibt ihr Programm aus, welchen Fehler meldet Valgrind?

2.3 Paralleles Debugging mit DDT

In dieser Aufgabe sollen Sie sich mit dem parallelen Debugger DDT vertraut machen.
Unter folgendem Link finden Sie eine kurze Einführung in DDT:

`http://wr.informatik.uni-hamburg.de/teaching/ressourcen/debugging#ddt`

Nutzen Sie die DDT-Installation auf dem Cluster in dem Sie von der Kommandozeile aus `ddt` starten. (Hinweis: Das Starten von DDT ist nur auf dem Login Knoten möglich. Das Programm was analysiert wird kann jedoch auf mehreren Knoten laufen)
Wenn es dabei Probleme geben sollte, stellen Sie sicher das Sie X-Fowarding für Ihre Session angestellt haben (`ssh -X`).
Je nach lokaler Internetverbindung kann es passieren das die Kommunikation über X-Fowarding sehr langsam ist. Falls dies für Sie der Fall ist können Sie DDT von der Website des Herstellers herunterladen, auf ihrem lokalen Rechner installieren, und die `Remote Launch` Funktion verwenden. `http://www.allinea.com/products/downloads/clients`
Sehen Sie sich Ihr Open-MPI Programm von letzter Woche oder aus der nächsten Aufgabe in DDT an und machen Sie sich mit dem Debugger vertraut. Dokumentieren sie kurz die folgenden Punkte:

- Setzen Sie in einer Zeile einen Breakpoint. Welche Step-Möglichkeiten gibt es und wie unterscheiden sich diese?
- Schauen Sie sich die Werte einzelner Variablen an. Wie kann man diese über mehrere Prozesse hinweg vergleichen?
- Welche Funktionalität bietet das Evaluate-Fenster in der rechten unteren Ecke.
- Wie wechselt man zu einem anderen Prozess?
- Wie lassen sich Datenstrukturen visualisieren. Fügen Sie hierfür ein Array, initialisiert mit beliebigen Werten zu ihrem Programm hinzu.

Abgabe:

`2-debugging.txt`	Kurze Beschreibung des Vorgehens sowie Antworten auf die Fragen.
`2-rekursion.c`	Ihr Programm das Sie zum Testen mit GDB verwendet haben.
`2-array.c`	Ihr Programm das Sie zum Testen mit Valgrind verwendet haben.

3 Einfache MPI-Befehle (60 P)

3.1 Pre/Post-Processing

Oftmals wird zu Beginn einer Anwendung eine Vorverarbeitung durchgeführt, diese Information wird von einem Master-Prozess (Rang 0) an alle Prozesse weitergeben. Am Ende des Programmes werden die Informationen wieder von dem Master eingesammelt und ausgegeben.
Schreiben Sie ein Programm, welches von der Kommandozeile beliebig viele Parameter im Zahlenformat annimmt. Diese Zahlen sollen fair auf die vorhanden Prozesse aufgeteilt werden (kein Prozess muss 2 Zahlen mehr berechnen als ein anderer, maximal 1 Zahl). Die Prozesse addieren dann ihre zugeteilten Zahlen in einer Schleife. Der Master-Prozess sammelt dann die Zahlen ein, gibt die einzelnen Summen aus, summiert alle Zahlen und gibt diese Gesamtsumme aus. Verwenden Sie zunächst nur die MPI Funktionen "Send" und "Recv" (und natürlich "Init" bzw. "Finalize").

3.2 Kollektive Operationen

Ersetzen Sie im vorangegangen Beispiel (Pre/Post-Processing) alle Sende und Empfangsoperationen durch die MPI Funktionen "Scatterv" und "Gather". Nun sollten Sie keine "Send" und "Recv" Operationen mehr benötigen!

3.3 Aggregationen

Ersetzen Sie im vorangegangen Beispiel (Pre/Post-Processing) das Aufsammeln aller Zahlen im Master-Prozess durch die "Reduce" Funktion. Welche Vereinfachung könnte man durch das Nutzen der "Reduce" Funktion noch vornehmen?

3.4 Datenaustausch

Tauschen Sie zwischen zwei Prozessen gegenseitig mindestens 1 MByte an Daten aus (erstellen Sie z. B. einen Array mit 1 Million Integern). Jeder Prozess sollte nach dem `MPI_Finalize` noch eine Nachricht ausgeben, dass er beendet hat.
Der erste Prozess soll die Daten an den zweiten Prozess schicken und umgekehrt. Verwenden Sie nur "Send" und "Recv" (und natürlich "Init" bzw. "Finalize"). Drehen Sie die Reihenfolge der Send und Recv der einzelnen Prozesse um. Welches Problem kann auftreten. "WARNUNG": Verwenden Sie eine niedrige Wallclock Zeit im Jobskript oder starten Sie das Programm interaktiv.

Abgabe:

`3-mpi-befehle.zip` Der gezippte Ordner der für jede Teilaufgabe eine C-Datei enthält und ein Makefile zum Erstellen aller Programme.

4 Parallelisierungsschema für MPI (90 P)

Überlegen Sie sich eine Aufgabe, die sie parallel abarbeiten möchten, bspw. Kassen in einem Supermarkt oder die Anzahl aller Möglichen Mühle-Stellungen berechnen. Tipp: Möglicherweise denken Sie bereits an etwas, dass sie später möglicherweise bearbeiten wollen.
Überlegen Sie, wie sich Ihr Problem auf mehrere Prozesse bzw. Threads aufteilen lässt. Wie werden Daten bzw. Code abgearbeitet, wie wird die Eingabe auf die Prozesse verteilt bzw. die Ausgabe eingesammelt? Skizzieren Sie als Pseudo-Code ihr Schema in einer Datei.

Beispiel für die verteilte Berechnung einzelner Elemente einer 2D-Matrix:

```
// matrix der Größe N
int m[N][N]
```

```
berechne start aus size, rank
berechne ende aus size, rank

for (i = start ; i < ende; i++){
   for (j = 0; j < N; j++){
      berechne(m[i][j])
   }
}

if (rank != 0){
   send(lokale Submatrix an Rank 0)
}
if (rank == 0){
   for (i = 1; i < size; i++){
      receive(submatrix von i)
   }
   schreibe submatrix in Datei
}
```

Bitte suchen sie sich ein nicht ganz triviales Beispiel aus.

4.1 OpenMP

Wie könnte man Ihr Problem mittels OpenMP angehen?

Das triviale Beispiel der Berechnung von Matrixelementen

```
int m[N][N]

// es gibt keine Datenabhängigkeit bei der Berechnung einzelner Elemente.
#pragma omp parallel for
for (i = 0; i < N; i++){
   for (j = 0; j < N; j++){
      berechne(m[i][j])
   }
}
```

Abgabe:

- `4-mpi.c` Ihr MPI Code mit der von Ihnen gewählten Problemstellung und (leicht) kommentiertem Pseudocode.
- `4-omp.c` Ihr OpenMP Code mit der von Ihnen gewählten Problemstellung und (leicht) kommentiertem Pseudocode.

5 MPI Details (Optional)

5.1 Datentypen

Erstellen Sie einen abgeleiteten Datentyp für eine Struktur bestehend aus 3 Integern und einem String aus 20 Zeichen. Initalisieren Sie die Struktur mit Werten und Senden Sie die Werte zu einem zweiten Prozess. Lassen Sie sich auf beiden Prozessen die Werte zur Sicherheit ausgeben.

5.2 Matrix-Multiplikation

In dieser Aufgabe wollen wir eine Matrix Multiplikation mit einem Skalar durchführen.

Schreiben Sie ein Programm, welches von einer Textdatei eine Matrix einliest. Von einer zweiten Textdatei soll ein Skalar eingelesen werden. Die Dateien sollen per Kommandozeile übergeben werden.
Die Matrix wird auf die einzelnen Prozesse aufgeteilt und jeder Prozess führt die Multiplikationen durch.
Die Ausgabe soll in eine dritte Datei geschrieben werden.
Vielleicht ist der Speicher eines Knotens nicht groß genug die gesamte Matrix zu halten, daher stellen Sie sicher dass jeder Prozess nur die benötigten Daten im Speicher hält.
Verwenden Sie POSIX-Funktionen ("open()", "read()", "write()") um die Datei zu lesen bzw. zu schreiben.
Prüfen Sie die Korrektheit ihrer Lösung!
Beispiele für die Eingabedateien finden Sie auf der Webseite. Testen Sie verschiedene Prozesszahlen z. B. "1, 2, 4, 8" mit den bereitgestellten Eingabedateien und lassen Sie sich die maximale (und minimale) Laufzeiten über alle Prozesse ausgeben (messen Sie zwischen "Init" und "Finalize"). Verwenden Sie hierfür die Funktion `clock_gettime` (verwenden Sie `man clock_gettime()` bzw. in Fortran (ab 95) `cpu_time()`). Um die maximale (und minimale) Laufzeiten eines Prozesses zu ermitteln bietet sich ein MPI Befehl an, welchen Sie bereits kennen gelernt haben. Wie erreichen Sie dass Sie nur einen MPI Aufruf benötigen um die minimale und maximale Zeit zu erhalten? Wie verhalten sich hier die Laufzeiten in Abhänigkeit zur Prozessanzahl?

5.3 Matrix-Matrix-Multiplikation

Erweitern Sie das Programm mit der Skalaren Multiplikation, so dass eine Matrix-Matrix Multiplikation erfolgt. In dem Beispiel sollten beide Matrizen gleichmäßig auf beide Prozesse aufgeteilt werden. Versuchen Sie die Kommunikation zwischen den Prozessen gering zu halten! Achten Sie auch auf den Speicherbedarf (kein Prozess sollte eine Matrix ganz im Speicher halten).
Dokumentieren Sie auch hier die Zeiten die Sie mit den bereitgestellten Eingabedateien erzielen. Wie verhalten sich hier die Laufzeiten in Abhänigkeit zur Prozessanzahl?

5.4 Ergebnisse vergleichen

Notieren Sie die erzielten Laufzeiten und Prozesszahlen in Sekunden, mit 3 Nachkommastellen in einer Textdatei.

Abgabe:

`5-mpi-details.zip` Ihre Ausarbeitungen zu den Teilaufgaben.

6 MPI-I/O (Optional)

MPI-2 definiert eine Schnittstelle zur Ein/Ausgabe von Dateien. Diese erlaubt portabel Dateien auszutauschen und den parallelen Zugriff auf die Dateien.

6.1 Matrix-Matrix-Multiplikation

Erweitern Sie ihr Matrix-Matrix Programm, so dass dieses MPI-I/O zur Eingabe bzw. Ausgabe der Daten nutzt. Erzeugen Sie eine Dateisicht um alle Daten der Matrix auf einmal einlesen zu können.
Sollten im Beispiel kollektive- oder individuelle MPI-Aufrufe verwendet werden?

Abgabe:

`6-mpi-io.zip` Ihr erweiterter Programmcode.