Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen
Michael Kuhn, Enno Zickler

Übungsblatt 1 zur Vorlesung
Hochleistungs-Ein-/Ausgabe
im SoSe 2018
Abgabe: 19.04.2018, 08:00

Bei jeglichen Problemen oder Nachfragen benutzen Sie bitte die folgende Mailingliste:

`hea-18@wr.informatik.uni-hamburg.de`

Sie können sich unter folgender Adresse in die Mailingliste eintragen:

`https://wr.informatik.uni-hamburg.de/listinfo/hea-18`

1 Cluster-Kennung

Zur Bearbeitung der Übungsaufgaben benötigen Sie generell kein Konto auf unserem Cluster. Die Korrektur der Blätter erfolgt allerdings auf dem Cluster, der somit auch als Referenzsystem gilt, auf dem Ihre Abgabe final funktionieren muss.

Um ein solches zu erhalten, tragen Sie sich zwingend mit korrekten und vollständigen Daten (Vorname, Nachname, E-Mail-Adresse) in die Mailingliste ein.

Weitere Informationen zum Cluster finden Sie auf unserer Webseite im Beginners' Guide:

`https://wr.informatik.uni-hamburg.de/teaching/ressourcen/beginners_guide`

2 C-Zeiger (60 Punkte)

In dieser Aufgabe sollen Sie sich mit dem grundlegenden Konzept der Zeiger in C vertraut machen. Dazu laden Sie sich die benötigten Materialien von der Webseite herunter. In der Datei `pointer.c` finden sie einige Funktionen. An einigen Stellen verrät die Ausgabe mittels `printf` das erwartete Ergebnis. An anderen Stellen verraten die Variablennamen oder Kommentare, was gemeint ist. Ihre Aufgabe ist es, die fehlenden Einträge zu vervollständigen, sodass die beschriebene Ausgabe korrekt erfolgt. Beachten Sie: Es darf nichts anderes am Programm geändert werden, außer die mit `TODO` gekennzeichneten Stellen. Das Programm muss am Ende fehler- und warnungsfrei kompilieren und eine semantisch korrekte Ausgabe produzieren.

Abgabe

Abzugeben ist der modifizierte Quelltext.

3 Debugging von C-Programmen (150 Punkte)

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` aus 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. Eine Liste von Links zu diesen beiden Programmen und wie diese auf dem Cluster verwendet werden können finden Sie auf unserer Webseite:

https://wr.informatik.uni-hamburg.de/teaching/ressourcen/debugging

Auf der Materialienseite befindet sich ein Archiv, in dem Zusatzmaterialien für die folgenden Aufgaben enthalten sind. Entpacken Sie dieses in Ihrem Home-Verzeichnis.

Erste Schritte

Im Verzeichnis `simple` ist ein primitives Programm enthalten, welches mit `make` kompiliert werden kann. Dieses Programm dient dazu, dass Sie sich ein wenig mit `gdb` und `valgrind` beschäftigen. Es enthält lediglich vier Funktionen, welche jeweils einen Zeiger auf eine Zahl oder ein Array mit einer Zahl enthalten und gibt diese dann in der `main`-Funktion aus. Leider enthält dieses Programm diverse Fehler.

- Führen Sie folgende kleinere Tests durch, um `gdb` kennen zu lernen.
  - Platzieren Sie einen Breakpoint auf der Funktion `mistake1`, starten Sie das Programm, geben Sie den Wert von `buf` und `buf[2]` aus. Gehen Sie zur nächsten Zeile und geben Sie beide Werte wieder aus. Von welchem Typ ist `buf`?
  - Platzieren Sie einen Breakpoint in der Funktion `mistake2`, setzen Sie den Programmlauf fort. Welchen Typ hat `buf`?
  - Setzen Sie den Programmlauf fort, welcher Text wird nun ausgegeben? Lassen Sie sich den Code um diese Stelle herum ausgeben. Welche Frames sind auf dem Stack? Wechseln Sie zu Frame 1. Geben Sie den Inhalt von `p` aus.
  - Rufen Sie im `gdb` die Funktion `mistake4` auf (schauen Sie nach, wie man in `gdb` Funktionen direkt aufrufen kann).
- Modifizieren Sie das Programm zunächst so, dass es nicht mehr abstürzt. Versuchen Sie die Modifikationen möglichst gering zu halten. Verwenden Sie zunächst `gdb`, um die Fehlerstellen aufzuspüren. Die Ausgabe soll dabei wie folgt aussehen:

  ```
  1 1
  2 2
  3 3
  4 4
  ```

- Nun läuft das Programm, leider enthält es jedoch noch weitere Speicherfehler, die je nach Umgebung (mehr oder weniger zufällig) auftreten können. Modifizieren Sie das Programm unter Zuhilfenahme von `valgrinds` `memcheck` so, dass jede Methode Speicher korrekt reserviert und dass am Ende der Programmlaufzeit der Speicher korrekt freigegeben wird. (Hinweis: Den Speicher einfach mit `static` zu allokieren ist **nicht** erlaubt. Verzichten Sie darüber hinaus auf globale Arrays und Variablen.)

Dokumentieren Sie die Fehler, die zu den Abstürzen und Speicherfehlern führen. Notieren Sie hierfür für jeden vorhandenen Fehler die Code-Zeile(n), welche fehlerhaft sind und den genauen Grund der Ursache (z.B. Speicher mehrfach freigegeben).

Abgabe

Abzugeben sind eine Textdatei mit Fehlerbeschreibungen (`simple-error.txt`) und der modifizierte Quelltext.

4 Checkpoints

In den folgenden Aufgaben sollen Sie sich mit den grundlegenden Lese- und Schreiboperationen in C vertraut machen. In den Materialien auf der Webseite finden Sie ein bereits vorgegebenes Programmgerüst `checkpoint.c`. Darin befindlich ist eine Funktion `calculate`, welche iterativ mathematische Operationen auf einer Matrix durchführt. Zum Ausführen des Programms müssen die Anzahl der Threads sowie der Iterationen übergeben werden.

In der Ausgabe können Sie die statistischen Auswertungen des Programmlaufs, wie die benötigte Zeit, den Durchsatz oder die IOPS nachvollziehen.

4.1 Checkpoint schreiben (90 Punkte)

Modifizieren Sie das gegebene Programm so, dass in jeder Iteration ein Checkpoint in die Datei `matrix.out` geschrieben wird; das Schreiben soll dabei parallel durch alle Threads erfolgen. Die Werte der Matrix sollen in jeder Iteration überschrieben werden. Sehen Sie sich dazu die Funktionen `write` und `pwrite` an und beachten Sie die parallele Umgebung (Tipp: Thread-Safety). Beschreiben Sie kurz die Funktionsweise beider Funktionen und begründen Sie Ihre Wahl für die gegebene Aufgabe. Integrieren Sie einen Header in die Ausgabedatei, der die Aufrufparameter des Programmlaufs – T (Threads), I (Iterationen) und I_c (die Nummer der zuletzt geschriebenen Iteration) – beinhaltet.

Implementieren Sie außerdem die Berechnung für den erreichten Durchsatz sowie die IOPS. Überlegen Sie sich hierbei auch, ob es sinnvoll ist, den Durchsatz und die IOPS auf Basis der Gesamtlaufzeit oder nur auf Basis der für die Ein-/Ausgabe benötigten Zeit zu berechnen.

4.2 Checkpoint lesen (60 Punkte)

Die Matrix wird derzeit statisch mit vorgegebenen Werten initialisiert. Implementieren Sie die vorgegebene Funktion `read_matrix`. Erweitern Sie hierzu die Eingabeparameter um den Pfad zu der vorhin geschriebenen Datei `matrix.out`. Sofern kein Pfad angegeben wird oder die Datei nicht existiert, soll die Matrix weiterhin statisch initialisiert werden; denken Sie an angemessene Fehlerbehandlung.

Es werden zwei hintereinander ausgeführte Programmläufe L_1 und L_2 mit den jeweiligen Aufrufparametern T_1, I_1, T_2 und I_2 betrachtet. Dabei wurde L_1 schon durchgeführt und hat einen Checkpoint mit den Werten für T_1, I_1 und I_c im Header geschrieben. Nun soll L_2 ausgeführt werden. Dabei soll der in L_1 geschriebene Checkpoint für L_2 weiterverwendet werden. Zunächst soll der Header analysiert und vier Szenarien betrachtet werden:

1. $T_1 \neq T_2$, wobei die Matrixgröße nicht von der Anzahl der Threads abhängt.
2. $I_c < I_1$
3. $I_c = I_1$ und $I_c > I_2$
4. $I_c = I_1$ und $I_1 < I_2$

Überlegen Sie sich und implementieren Sie sinnvolle und angemessene Vorgehensweisen für die ersten drei beschrieben Fälle. Begründen Sie Ihre Entscheidung.

Für den letzten Fall implementieren Sie das Einlesen der Werte aus der letzten geschriebenen Iteration. Mit diesen Werten soll im aktuellen Lauf (L_2) die Matrix initialisiert und ab der entsprechenden Iteration ($I_c + 1$) weiter berechnet werden.

4.3 Atomare Checkpoints (60 Punkte)

Implementieren Sie das atomare Schreiben des Checkpoints, d.h. selbst bei einem Programmabsturz während des Schreibens soll sich der Checkpoint trotzdem in einem konsistenten Zustand befinden. Konsistent bedeutet in diesem Zusammenhang, dass die geschriebenen Werte vollständig aus ein und derselben Iteration stammen und I_c den geschriebenen Werten zuzuordnen ist. Diskutieren Sie die Vor- und Nachteile von zwei möglichen Lösungsansätzen.

Abgabe

Abzugeben sind eine Textdatei mit den Antworten auf die Fragen (`antworten.txt`) und der modifizierte Quelltext.

5 Abgabe

Packen Sie ein komprimiertes Archiv (`.tar.gz`) aus dem sauberen Verzeichnis (ohne Binärdateien). Um das Archiv zum Versenden auf ihren Rechner zu kopieren, können sie `scp` verwenden. Näheres dazu finden Sie auf unserer Webseite.

Senden Sie das Archiv an `hea-abgabe@wr.informatik.uni-hamburg.de`.