Universität Augsburg
University

Vertiefte Multicore-Programmierung

Universität Augsburg
Institut für Informatik
Prof. Dr. Theo Ungerer
Dr. Sebastian Weis

20. April 2016

Aufgabenblatt 1

Allgemeine Hinweise

Unter *https://sik-svn.informatik.uni-augsburg.de/svn/multi16_{RZ-Kennung}* wurde für jeden Teilnehmer ein Subversion-Repository angelegt. Sichern Sie Ihre Lösungen bzw. alle Abgaben in diesem Repository. Das jeweilige Übungsblatt, allgemeine Dokumente und Quellcodevorlagen finden Sie in ihrem Home-Verzeichnis unter *share/multicore/Blattxx* und in Digicampus.
Das Erscheinen zum wöchentlichen Übungsstermin ist verpflichtend. Der Praktikumsraum kann aber auch außerhalb der Übungszeit, sofern dort keine anderen Veranstaltungen stattfinden, genutzt werden.
Sorgen Sie außerdem dafür, dass Ihre Abgaben immer (sinnvoll) kommentiert und für Dritte nachvollziehbar sind.

Hinweis

Verwenden Sie die vorgegeben Vorlagen für die jeweilige Aufgabe und kompilieren Sie den Quelltext mit dem vorgegebenen Makefile. Der Standard für alle Aufgaben ist C++11 (`-std=c++11`).
Rufen Sie zu Beginn der `main`-Funktion `mcp_init(argc, argv)` auf. Die Funktionen `get_num_threads()` und `get_num_elements()` geben Ihnen die Anzahl der Threads, bzw. die Eingabegröße zurück. Rufen Sie Ihr Programm mit: `./<name> -n <NUM> -t <THREADS>` auf. `<NUM>` gibt die Eingabegröße an, `<THREADS>` die Anzahl der Threads.
Um den Quelltext zu kompilieren, rufen Sie `make` auf. Falls Sie *Eclipse* verwenden sollten, erstellen Sie ein neues Projekt (`File - New - Makefile project with existing code`) und importieren Sie den Ordner der entsprechenden Aufgabe.

1. Aufgabe (1 Punkt)

Implementieren Sie eine C++11-Klasse, die einen Zähler inkrementiert und von mehreren Threads aufgerufen werden kann. Das Inkrementieren soll durch Verwendung des `++`-Operators auf dem Objekt realisiert werden, dieser Operator muss dazu überladen werden. Vergleichen Sie den Wert des Zählers bei Programmende mit und ohne Schutz vor gleichzeitigem Zugriff.

2. Aufgabe (2 + 4 Punkte)

Implementieren Sie eine parallele Berechnung der Standardabweichung.

a) Erstellen Sie eine Klasse *Barrier*, die mit Hilfe eines Mutex und einer Conditional-Variable eine Barriere implementiert. Zur Verwendung der Barriere in mehreren Objekten kann diese statisch instantiiert werden (`static Barrier b(n);` für `n` Threads). Das Warten an der Barriere wird durch `void Barrier::wait()` realisiert.

b) Verwenden Sie oben erstellte Barriere in einem mehrfädigen Programm zur Berechnung der Standardabweichung. In einem ersten Schritt soll der Durchschnitt aller Elemente eines Vektors berechnet werden. Jeder Thread berechnet dazu in einem ihm zugewiesenen Intervall lokal den Durchschnitt. Dieser wird dann mit den Durchschnittswerten der anderen Threads zu einem globalen Durchschnitt verrechnet. Mit diesem kann nun die Abweichung jedes Werts vom Mittelwert bestimmt werden. Jeder Thread soll in seinem Intervall der Eingabewerte die Varianz berechnen. Ein Thread akkumuliert die lokal berechneten Varianzen und berechnet daraus die Standardabweichung über alle Eingabewerte.

3. Aufgabe (3 Punkte)

Entwickeln Sie einen Sudoku-Löser mit Hilfe eines selbst entwickelten Thread-Pools. Dazu werden noch nicht vollständig gelöste Sudokus in einem Stack verwaltet. Ein Thread nimmt dabei jeweils das oberste Rätsel vom Speicher, berechnet alle möglichen Werte für ein noch nicht ausgefülltes Feld und legt Kopien der neuen Teillösungen auf den Stack zurück. Initial befindet sich nur das zu lösende Sudoku auf dem Stack. Der Lösungsalgorithmus soll dabei wie im abgebildeten Pseudocode vorgehen.
Implementieren Sie den Sudoku-Löser für Sudokus der Größe 9x9. Stellen Sie sicher, dass das Programm alle möglichen Lösungen findet und danach terminiert. Der Thread-Pool soll eine einstellbare Anzahl von Workern (`get_num_threads()`, Parameter `-n`) besitzen und überschüssige Threads in einer Warteschlange verwalten. Testen Sie ihr Programm mit verschiedenen Eingabe-Sudokus. Messen Sie die Laufzeit ihres Programms mit den angegeben Funktionen `time_start()`, `time_stop()` und `time_print()`, und untersuchen Sie die Skalierbarkeit ihrer Lösung für 1-32 Worker.

Eingabe : Teilweise ausgefülltes Sudoku Feld S

Lege S auf den Stack
for *alle Sudoku Felder S' im Stack* **do**
  Finde die Koordinaten x,y eines unausgefüllten Feldes in S'
  for *alle möglichen Kandidaten k der Koordinate x,y* **do**
    erstelle eine Kopie S' von S
    füge den Wert k an die Stelle x,y ein
    platziere S' zur weiteren Bearbeitung auf dem Stack
  end
end

Linksammlung

- C++ Tutorial: `http://www.tutorialspoint.com/cplusplus/`
- C++ Tutorial: `http://www.cplusplus.com/doc/tutorial/`
- C++ Referenz: `http://www.cplusplus.com/reference/`
- C++ Referenz: `http://en.cppreference.com/w/cpp`