Praktikum Algorithm Enginnering:
Multicore-Programmierung mit der (MC)STL

Peter Sanders, Johannes Singler SS 2007

http://algo2.iti.uni-karlsruhe.de/mcstl-praktikum.php

Arbeitsblatt 1

Abgabe: 3. Mai 2007

Aufgabe 1: Einarbeiten

- Lesen Sie das Paper zur Multi-Core Standard Template Library [3] und schauen Sie sich (noch einmal) die Folien zur zugehörigen Vorlesung [4] an.
- Machen Sie sich mit den STL-Algorithmen vertraut [1].
- Lesen Sie sich in C++ ein [6, 2].
- Probieren Sie ihren Account aus und testen Sie, ob die Compiler funktionieren: `icpc` und `g++-4.2.0`.
- Besuchen Sie die MCSTL-Website [5], laden Sie sich die aktuelle Version der Bibliothek herunter, installieren Sie sie in Ihrem Home-Verzeichnis und lesen Sie die Dokumentations-Seite.

Aufgabe 2: Kennenlernen der Konzepte Iterator und Funktor aus der STL

Schreiben Sie ein Programm, das

- einen `vector` von n `double`-Variablen erzeugt, und diese mittels `generate` mit den Werten $\sin(i), i = 0..n-1$ füllt. Dazu benötigen Sie einen *Funktor*.
- aus dem obigen Vektor die Elemente an den Positionen $i \in \mathbb{N}_0 : 2^i < n$ ausgibt.

Probieren Sie das gleiche mit einer Liste (`list`). Was gibt es jetzt für ein Problem?

Aufgabe 3: Benutzung der MCSTL und Messung der Beschleunigung (Abgabe)

Schreiben Sie ein Programm, das einen STL `vector` mit 1.000.000 zufälligen Integern füllt und ihn dann mittels `sort` sortiert.

- Die Laufzeit soll mittels `omp_get_wtime` gemessen werden. Dazu muss `omp.h` inkludiert werden und das Programm mit OpenMP-Unterstützung kompiliert werden: `g++-4.2.0 -O3 -g0 -fopenmp -o parsort parsort.cpp`
- Kompilieren Sie das Programm jetzt mit der MCSTL, fügen Sie also `-I$(MCSTL_ROOT)/c++ -I$(MCSTL_ROOT)/originals/_software_gcc_include_c++_4.2.0` in das Kommando ein, wobei `-I$(MCSTL_ROOT)` für den Pfad zu Ihrem MCSTL-Verzeichnis ist. Führen Sie Laufzeitmessungen für verschiedene Anzahlen von Threads durch. Wie groß ist der Beschleunigungsfaktor (Speedup) auf einem der Multicore-Rechner? Experimentieren Sie mit anderen Eingabegrößen, um festzustellen, wo der Break-Even ist, d. h. ab welchem Wert sich ein Speedup größer 1 ergibt.

Aufgabe 4: Vereinigung von Mengen (Abgabe)

Die STL enthält vier Algorithmen für Mengen, nämlich `set_union` (Vereinigung), `set_intersection` (Schnitt), `set_difference` (Differenz) und `set_symmetric_difference` (symmetrische Differenz, $(A \cup B) \setminus (A \cap B)$). Gemäß der Spezifikation der Mengen-Algorithmen liegen die Eingabe-Mengen als sortierte Sequenzen vor, mittels deren `begin`- und `end`-Iteratoren. Zusätzlich können Sie annehmen, dass die Iteratoren wahlfreien Zugriff erlauben (Random Access).

1. Implementieren Sie zunächst eine parallele Variante von `unique_copy` mittels OpenMP. Statische Lastverteilung reicht aus, d. h. jeder Thread bekommt die gleiche Anzahl (plus minus 1) Eingabewerte zugeteilt. Welche Probleme ergeben sich? Welchen Spezialfall müssen Sie berücksichtigen? Sie dürften zusätzlichen Speicherplatz benutzen, um zu einer effizienten Lösung zu gelangen.

2. Implementieren Sie `set_union`, ohne expliziten Parallelismus zu nutzen, sondern nur mit Hilfe der bereits parallelisierten Algorithmen `merge` und `unique_copy`, sowie `copy`.

3. Schreiben Sie ein Programm, das die Funktion für beliebige Sequenzen von 32-Bit-Integern und Anzahl von Threads testet und dabei die Laufzeit misst. Die Eingabedaten sollen zufällig erzeugt werden, müssen aber natürlich vorsortiert werden. Die Korrektheit des Ergebnisses muss automatisch überprüft werden, d. h. mit dem Ergebnis des sequentiellen Algorithmus verglichen werden. Dokumentieren Sie die Speedups für bis zu 4 Threads auf einem entsprechenden Rechner, für Eingabegrößen von 1 bis 10.000.000. Experimentieren Sie auch mit dem Fall, dass beide Eingabemengen nicht gleich groß sind. Fassen Sie die Ergebnisse sinnvoll zusammen; erstellen Sie mindestens ein Schaubild, das den Speedup über der Eingabegröße zeigt.

4. An welchen Stellen wäre ein explizit parallelisierter Algorithmus möglicherweise effizienter?

Hinweise:

- Führen Sie immer mehrere (mindestens 10) Messungen aus, und mitteln Sie die Zeiten darüber. Kompilieren Sie ihr Programm für Laufzeittests mit Optimierungen (`-O3`).
- Überprüfen Sie die Korrektheit Ihrer Implementierung auch mit Spezialfällen, z. B. viele gleiche Elemente, Sequenzen kürzer als die Anzahl Threads.
- Bitte verwenden Sie sprechende, englische Bezeichner.
- Kommentieren Sie Abschnitte des Quellcodes, die nicht offensichtlich sind, auf hohem Level. Aussagekräftige Bezeichner machen Kommentare oft überflüssig.
- Um nicht mit der bereits existierenden Version zu kollidieren, können Sie Ihren Implementierung von `unique_copy` leicht anders benennen.

Dokumentieren Sie ihre Lösung sowie die Messergebnisse schriftlich. Geben Sie dieses Dokument ausgedruckt ab, und schicken Sie dem Betreuer dieses als Datei sowie die Quelltexte.

Literatur

[1] Index: The standard template library. `http://www.sgi.com/tech/stl/stl_index.html`.

[2] S. B. Lippman, J. Lajoie, and B. E. Moo. *C++ Primer*. Addison-Wesley, 2005.

[3] F. Putze, P. Sanders, and J. Singler. The multi-core standard template library. `http://algo2.iti.uni-karlsruhe.de/singler/mcstl/multicorestl1.pdf`, January 2007.

[4] J. Singler. MCSTL: Practical implementation of parallel algorithms for shared-memory systems. `http://algo2.iti.uni-karlsruhe.de/sanders/courses/paralg06/singler.pdf`, December 2006.

[5] J. Singler. The MCSTL website, June 2006. `http://algo2.iti.uni-karlsruhe.de/singler/mcstl/`.

[6] B. Stroustroup. *Die C++ Programmiersprache*. Addison-Wesley, 2000.