Mini-Projekt: MPC

Sortieren auf Shared Memory Multi-Core

M. Thaler, ©ZHAW

1 Einführung

Sortieren ist eine häufige verwendete Operation in vielen Programmen, weil es das Handhaben von Daten stark vereinfachen kann. Viele parallele Verfahren wurden in der Vergangenheit untersucht und auf verschiedensten Architekturen implementiert.

In diesem Projekt werden Sie in 2er Gruppen ein paralleles Sort-Verfahren mit PThreads implementieren (siehe z.B. [1]).

2 Aufgabenstellung

In diesem Mini-Projekt soll für eine Shared Memory Multi-Core Architektur ein geeignetes Sortierverfahren gewählt und implementiert werden. Folgende Vorgaben müssen eingehalten werden:

a) Sortierfunktion: `parSort(sort_t *data, int len, int nThreads, int thresh)`
   - `sort_t *data`: Array, Länge `len`
   - `sort_t`: numerischer Skalartyp definiert in `parSort.h`
   - `int len`: Länge des Arrays, 100'000 $\leq$ `len` $\leq$ 1000'000.
   - `int nThreads`: maximale Anzahl Threads
     `nThreads` ≤ 0: Sortierverfahren wählt Anzahl Threads selbst
   - `int thresh`: Arraylänge, ab welcher sequentiell sortiert werden soll
     `thresh` ≤ 0: Sortierverfahren wählt `thresh` selbst
   - mindestens einer der beiden Parameter `nThreads` resp. `thresh` muss 0 sein
   - aufsteigend sortiert, Datenrückgabe *in-place* sortiert in `data`
   - Starten und Stoppen der Threads in der Sortierfunktion `parSort()`

b) Die Sortierfunktion muss aus einem beliebigen Programm aufrufbar sein und in `parSort.h` den Datentyp `sort_t` definieren können.

c) Optimieren Sie die Performance soweit als möglich (Performance-Evaluation: siehe unten)

d) Zeitmessungen mit dem *ctimer* (siehe Praktika), Start resp. Stop des Timers vor resp. nach dem Funktionsaufruf. Messgrösse: Elapsed Zeit (`MONOTONIC`) ohne zusätzliche Last.

2.1 Vorgehen

Erarbeiten Sie sich eine Übersicht zu möglichen Sortierverfahren, bewerten und wählen Sie anschliessend ein geeignetes Verfahren für die Implementierung. Verifizieren und optimieren Sie Ihren Code.

Hinweis: Im Kurs t.ADS haben Sie verschiedene sequentielle Sortierverfahren kennengelernt.

2.1.1 Performance-Evaluation

Der Speedup wird wie folgt gemessen (keine zusätzliche Rechnerlast):

$$S(n) = \frac{T_{elapsed}(1 \text{ Thread})}{T_{elapsed}(n \text{ Threads})}$$

Die Effizienz wird wie folgt gemessen (keine zusätzliche Rechnerlast):

$$E(n) = \frac{T_{elapsed}(1 \text{ Thread})}{n \cdot T_{elapsed}(n \text{ Threads})}$$

Die Messwerte sollen jeweils über 5 konsekutive Runs als Mittelwert ermittelt.

Zeichnen Sie Ihre Messwerte für einen Array der Länge 200'000 graphisch auf.

2.2 Bericht

Das Vorgehen und die Resultate müssen in einem Schlussbericht (max. 3 Seiten inkl. Graphiken, ohne Code) zusammengefasst und termingerecht abgegeben werden.

Der Bericht soll wie folgt aufgebaut sein:

1. **Einführung** → *was*
   Problemstellung, Ziel
2. **Vorgehen** →*wie*
   Vorgehen, Konzept, Rahmenbedingungen, Software-Design
3. **Resultate** → *was herausgekommen ist*
   Erreichte Resultate, Interpretation
4. **Reflexion** → *meine Schlussfolgerungen*
   (aufgetauchte) Probleme, was habe ich gelernt, Ausblick, etc.

2.3 Bewertung

Bewertet werden folgende Punkte (pro Projektgruppe eine Bewertung):

- termingerechte / korrekte Abgabe	1.0 Punkte
- Bericht	1.0 Punkte
- Funktionalität (Anzahl Threads, Arraylänge), Präsentation / Diskussion mit Dozent	3.0 Punkte

Das Sortieren sollte unter allen Randbedingungen korrekt arbeiten. Wenn Sie einschränkende Annahmen machen, bitte in Bericht dokumentieren.
Ihre Implementation wird mit einem Unit-Test automatisch verifiziert.

2.4 Termine

Abgabe:	Bericht als Papier-Kopie, Montag 30.4.2012, 12.00 Uhr Code der Sortierfunktion als Tarfile (tar -cvz) mit e-mail an tham@zhaw.ch Filename: `sortX.tgz`: `X` = Projektgruppen-Nummer