lea.schoenberger [☺] tu-dortmund.de
kuan-hsun.chen [☺] tu-dortmund.de
nils.hoelscher [☺] tu-dortmund.de

Übung zur Vorlesung
Rechnerarchitektur
Sommersemester 2019

Übungsblatt 3

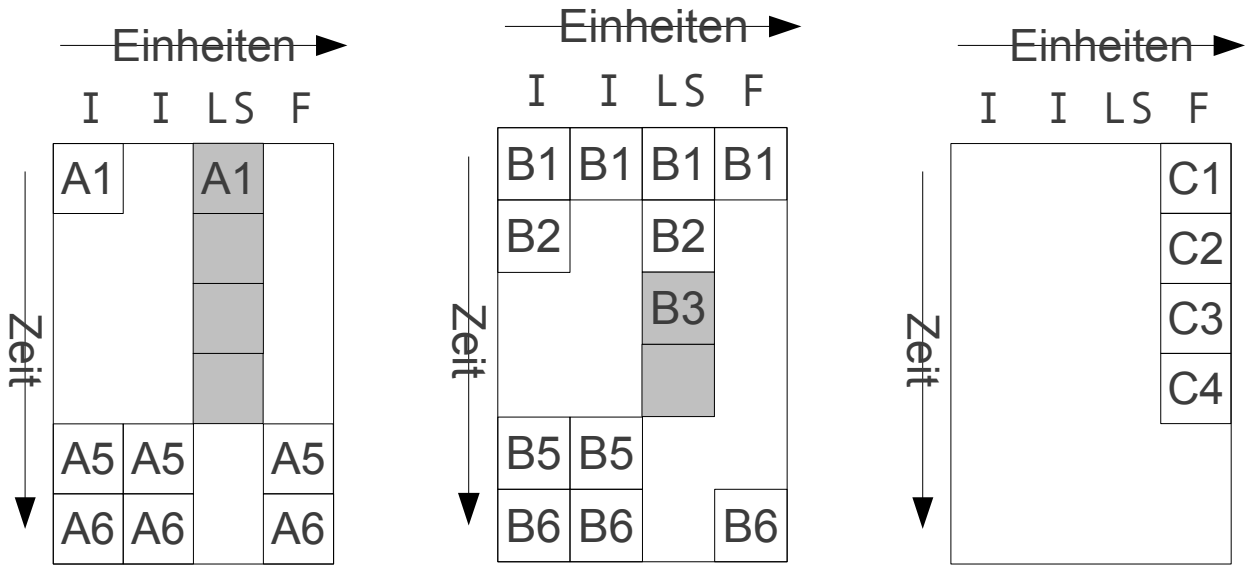
Bearbeitung ab Mittwoch, 8. Mai 2019

3.1 Multithreading

Gegeben sei dazu ein System mit einem Prozessor mit dynamischem Scheduling, der zwei Integer-Ausführungseinheiten (Latenz: 1 Takt), eine Load/Store-Einheit (Latenz: 1-4 Takte, je nach Cacheverhalten) und eine Floating-Point-Einheit (Latenz: 1 Takt) besitzt. Zudem sind ein L1-Cache und ein L2-Cache mit den folgenden Zugriffslatenzen vorhanden:

L1-Zugriff	L2-Zugriff	Latenz (Takte)
Hit	-	1
Miss	Hit	2
Miss	Miss	4

Beide Caches unterstützen *Hit-Under-1-Miss*, d.h., während ein Miss bearbeitet wird, können weitere Hit-Zugriffe abgewickelt werden. Ein Zugriff, der einen weiteren Miss produzieren würde, wird abgebrochen und muss vom Prozessor wiederholt werden.



(a) Thread A (Priorität 0) (b) Thread B (Priorität 1) (c) Thread C (Priorität 2)

Abbildung 1: Ausführungsverlauf dreier Threads auf der gegebenen Plattform

Für dieses System sind die Ausführungsverläufe dreier Programme A-C in Abbildung 1 skizziert, wie sie sich bei isolierter Ausführung auf dem System ergeben. Operationen, die mehr als einen Takt benötigen, sind hellgrau markiert.

Außerdem ist die Priorität jedes Threads angegeben (nur bei CGMT und SMT nötig), wobei niedrigere Zahlen einer höheren Priorität entsprechen.

a) Zeichnen Sie für eine Ausführung mit *Coarse-Grained Multithreading (CGMT)*, *Fine-Grained Multithreading (FGMT)* und *Simultaneous Multithreading (SMT)* jeweils den zeitlichen Verlauf der Abarbeitung auf dem gegebenen System. Richten Sie sich bei der Notation nach Abbildung 1, d.h., tragen Sie die Verarbeitungseinheiten horizontal und die Zeit vertikal auf. Der *Context-Switch-Overhead* sei ein Takt für CGMT und null Takte für die anderen Modelle.
Sie können davon ausgehen, dass die Speicherzugriffe der Threads auf disjunkte Cache-Bereiche abgebildet werden, d.h., dass diese Zugriffe interferenzfrei sind. Außerdem sei der verwendete Scheduler ein ASAP-Scheduler (As Soon As Possible). Falls nicht alle Instruktionen aus einem Zeitschrit aus den Abbildungen 1(a) bis 1(c) gleichzeitig gestartet werden können, führt der Scheduler die verbleibenden so bald wie möglich danach aus. Nehmen Sie in diesem Fall außerdem an, dass erst *alle* Instruktionen eines Zeitschritts i aus Abbildung 1 ausgeführt worden sein müssen, bevor Instruktionen aus Schritt $i+1$ desselben Threads ausgeführt werden können.

b) Was sind die hardwareseitigen Voraussetzungen für die einzelnen Modelle?

3.2 Grobgranulares Multithreading

Für die Ausführungszeiten der Instruktionen aller Threads einer Anwendung gilt diese Verteilung:

- 40% der Taktzyklen entfallen auf die Ausführung der Instruktionen im Prozessor.
- 30% der Taktzyklen sind Wartezyklen, die durch L1-Cache-Misses (aber L2-Cache-Hits) entstehen. Die L1-Miss-Bearbeitung benötigt 10 Taktzyklen.
- 30% der Taktzyklen sind Wartezyklen, die durch L2-Cache-Misses hervorgerufen werden (jeweils 30 Taktzyklen).

Im Durchschnitt tritt pro Thread nach jeweils 10 Rechentakten ein Cache-Miss auf. Beantworten Sie folgende Fragen unter der Annahme, dass ausreichend viele Threads sowohl von der Hardware als auch durch die Anwendung unterstützt werden.

a) Wie hoch ist die maximale Auslastung, wenn ein Threadwechsel 5 Taktzyklen dauert?

b) Wie lange darf ein Threadwechsel höchstens dauern, sodass die Auslastung mindestens 50% beträgt?

3.3 Parallelisierte Textverarbeitung mit OpenMP

a) Schreiben Sie ein C++-Programm, das 10MB zufälligen Text generiert. Nutzen Sie für die Generierung des Textes folgende Funktion:

```
void initText( string &s ) {
static const char alpha[] =
"abcdefghijklmnopqrstuvwxyz";

for ( int i = 0; i < s.length(); i++ ) {
s[i] = alpha[rand() % (sizeof(alpha) - 1)];
}
}
```

b) Implementieren Sie eine einfache Textsuche, die einen gegebenen, festen Text S in dem zufällig generiertem Text T sucht, indem sie S an jeder möglichen Stelle an T "anlegt" und die Zeichen von S der Reihe nach mit denen von T an der entsprechenden Position vergleicht. Implementieren Sie eine Zeitmessung für die Suchfunktion mithilfe der C-Funktion clock. Achten Sie hierbei darauf, dass bei der Suche *keine* I/O-Operationen durchgeführt werden (wie z.B. das Ausgeben von Nachrichten auf `cout` oder Aufrufe von `printf`), da diese die Zeitmessung verfälschen.

c) Testen Sie verschiedene Möglichkeiten, Ihre Implementierung aus Teil b) mithilfe der OpenMP-Pragmas zu parallelisieren. Messen Sie auch hier jeweils die Zeit und geben Sie eine Parallelisierung an, die auf dem von Ihnen benutzten Rechner gut funktioniert.

d) Ergänzen Sie Ihre Lösung aus c) so, dass die Anzahl Treffer bei der Suche ermittelt wird. Die Ermittlung soll hier online während der Suche geschehen und nicht erst, nachdem der komplette Text T durchsucht wurde und damit alle Treffer gefunden wurden. Nutzen Sie die Synchronisierungmechanismen von OpenMP, um die Korrektheit des Ergebnisses sicherzustellen.

e) Können Sie die Korrektheit in Teil d) auch mithilfe einer *Barriere* sicher stellen? Wenn ja, wie?

Allgemeine Hinweise: Informationen zur Veranstaltung finden Sie unter https://ls12-www.cs.tu-dortmund.de/daes/lehre/lehrveranstaltungen/sommersemester-2019/rechnerarchitektur-ss19.html. Eine Abgabe der Übungsblätter ist nicht erforderlich.