Praktikum 1: Parallel Computing Hardware

M.Thaler, 2/2016, ZHAW

1 Einführung

Ziel dieses Praktikum ist es, sie vertraut zu machen mit:

- Informationen zur verwendeten Prozessor-Hardware
- Messung und Analyse von Laufzeitparamtern eines parallelen Programms
- Einflussfaktoren der Prozessor-Hardware auf die Laufzeit eines parallelen Programms

2 Die Prozessor-Hardware

Informationen zur Prozessorhardware wie Anzahl CPUs, Speicherorganisation, etc. können unter Linux mit dem Befehl **'lscpu'** abgefragt werden. Die Ausführung paralleler Programme kann mit dem Befehl **'htop'** dynamisch visualisiert werden.

2.1 `lscpu` (nur Linux)

Analysieren sie die Ausgabe von `lscpu` auf Ihrem Rechner und auf dem Praktikumsserver **DOVE**. Welcher Zusammenhang besteht zwischen der Anzahl CPUs, Cores und Threads[1]?

Parameter	ihr Rechner	DOVE
Architecture		
Cores		
Threads		
CPUs		
Instruction Cache Level 1		
Data Cache Level 1		
Cache Level 2		
Cache Level 3		

Für das Einlesen der Anzahl CPUs, Cores und Hardware-Threads aus einem Programm, stellen wir Ihnen das Header-File **cpuinfo.h** mit entsprechenden Funktionen zur Verfügung, eine einfache Anwendung finden sie im Verzeichnis `hardware/a1`.

2.2 `htop` (nur Linux)

Wichtige Informationen zur Ausführung von parallelen Programmen erhalten sie mit dem Befehl **'htop'**, der unter anderem die Auslastung der einzelnen CPUs angezeigt.

Starten sie **htop** und geben sie ein kleines `'h'` ein um die online Hilfe anzuzeigen. Hier wird erklärt, wie die Anzeigebalken farblich kodiert sind und welche Befehle zur Verfügung stehen. Wichtig ist der Befehl `'H'`, mit dem alle Software-Threads angezeigt werden können (im Normalfall wird nur der Prozess angezeigt). Der Aufruf **htop -u $USER -d 5** zeigt nur Ihre eigenen Programme und erneuert die Anzeige alle 0.5s.

[1] hier sind Hardware Threads gemeint (Hyperthreading)

3 Zeitmessung

Das Laufzeitverhalten von parallelen Programmen lässt sich mit Hilfe von Zeitmessungen analysieren. Zeitmessungen auf aktuellen Multicore-Systemen liefern jedoch Resultate mit eingeschränkter Genauigkeit:

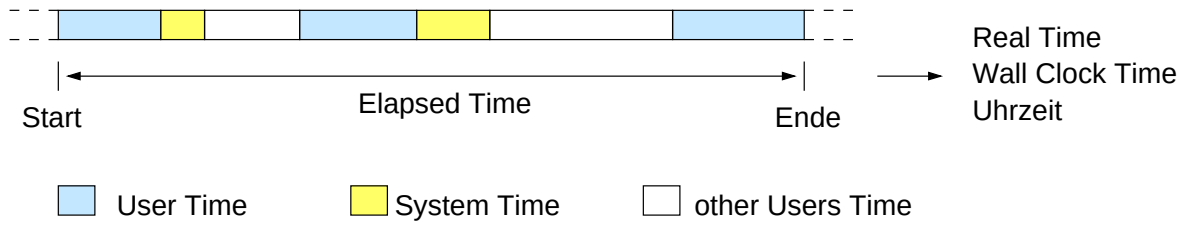
- die Zähler (Timer) auf verschiedenen Cores stimmen nicht unbedingt überein
- Threads können während der Laufzeit zwischen Cores migrieren
- Prozessoren und das BS passen die Taktrate dynamisch den Anforderungen an, etc..

Diskussionen dazu finden sie z.B. auf dem WEB unter dem Stichwort *Time Stamp Counter*.

Dennoch liefern Zeitmessungen hilfreiche Hinweise, wie sich das Laufzeitverhalten von parallelen Programmen abhängig von Anzahl Threads und verfügaren CPUs verhält.

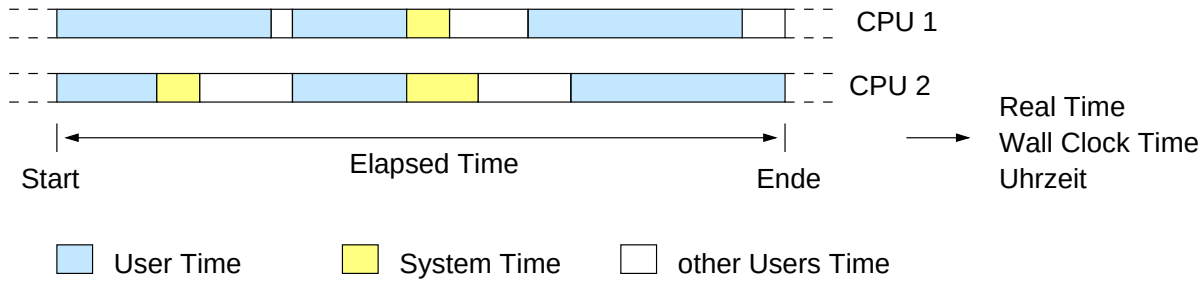
3.1 Welche Zeitgrössen lassen sich messen?

Auf einem Rechner mit Betriebssystem werden folgende Zeitgrössen unterschieden:



Die *Elapsed Time* ist die Zeit zwischen Start und Ende eines Programms, unabhängig von Anzahl CPUs und für wen der Prozessor (user, system) arbeitet. Die *Elapsed Time* einer Anwendung nimmt zu, wenn andere Benutzer den Rechner gleichzetig belasten.

Folgende Figur zeigt die ein Beispiel mit mehreren CPUs:



Wie gross ist die User Time im Verhaltnis zur Elapsed Time?

3.2 Funktionen (Macros) für die Zeitmessungen in MPC

Für Zeitmessungen steht das Header-File **`mtimer.h`** zur Verfügung, das drei verschiedene Unix Timer unterstützt. Für jeden der drei Timer sind ein Datentyp (Deklaration der benötigten Datenstrukturen) sowie die entsprechenden Timer-Funktionen (Macros) definiert. Unten stehende Tabelle zeigt die Funktionen und Eigenschaften der einzelnen Timer auf.

Timer	Datentyp	Timer-Funktionen	Was	Auflösung	System Funktion

gtimer	`gtimer_t`	`startGTimer(tdata)`
		`stopTGimer(tdata)`
		`printGTime(tdata)`
			Uhrzeit	1 μs	**`gettimeofday()`**

ttimer	`ttimer_t`	`startTTimer(tdata)`
		`stopTTimer(tdata)`
		`printTTime(tdata)`
			Uhrzeit
			System Time
			User Time
				10 ms	**`times()`**

ctimer
(Linux)
	`ctimer_t`	`initCTimer(tdata, mode)`
		`startCTimer(tdata)`
		`stopCTimer(tdata)`
		`printCTime(tdata)`
			Uhrzeit
			Prozess
			Thread
				1 ns	**`clock_gettime()`**

Die *Print-Funktionen* geben die Zeit in Sekunden aus.

Weitere Informationen zu den Timern finden sie in den Kommentaren im File `mtimer.h` sowie in den Manual Pages zu den entsprechenden Linux/Unix Funktionen.

3.3 Zeitmessung

Für Zeitmessungen stellen wir Ihnen im Verzeichnis `hardware/a3` Programme zu Verfügung, die jeweils mehrere Threads erzeugen und auf ihre Terminierung warten. Die Anzahl Threads kann als Parameter übergeben werden, Default ist 4.

Aufgaben:

a) Im Verzeichnis `hardware/a31` finden sie das Programm `main.c`. Analysieren sie, wie die Threads gestartet werden und wie in der Funktion `helloWorld()` den einzelnen Threads eine eindeutige ID übergeben wird. Hinweis: Der Typ **`long`** ist auf 32- und 64-Bit Systemen jeweils gleich lang wie ein Pointer.

b) Im Verzeichnis `hardware/a32` finden sie das Programm `main.c`, das 100'000 mal *leere* Threads erzeugt und auf ihre Terminierung wartet. Messen sie für die 100'000 Iterationen mit dem *ttimer* Timer die drei Zeitgrössen *Elapsed*, *System* und *User Time* wenn pro Iteration 1 Thread gestartet (`main.e 1`) wird und wenn gleich viele Threads wie Anzahl CPUs gestartet werden,. Wie lange dauert für beide Fälle im Mittel das Erzeugen und Terminieren eines einzelnen Threads (Messwerte/100'000)? Wie gross ist das Verhältnis von User zu System Time?

c) Im Verzeichnis `hardware/a33` finden sie ein Programm, das mehrere Threads erzeugt, die ihre Arbeit (Zähler inkremtieren) gleichmässig unter sich aufteilen. Notieren sie sich die Zeiten für [1, 2, ... CPUs] Threads mit Parameter `workload=0` und `workload=1`:

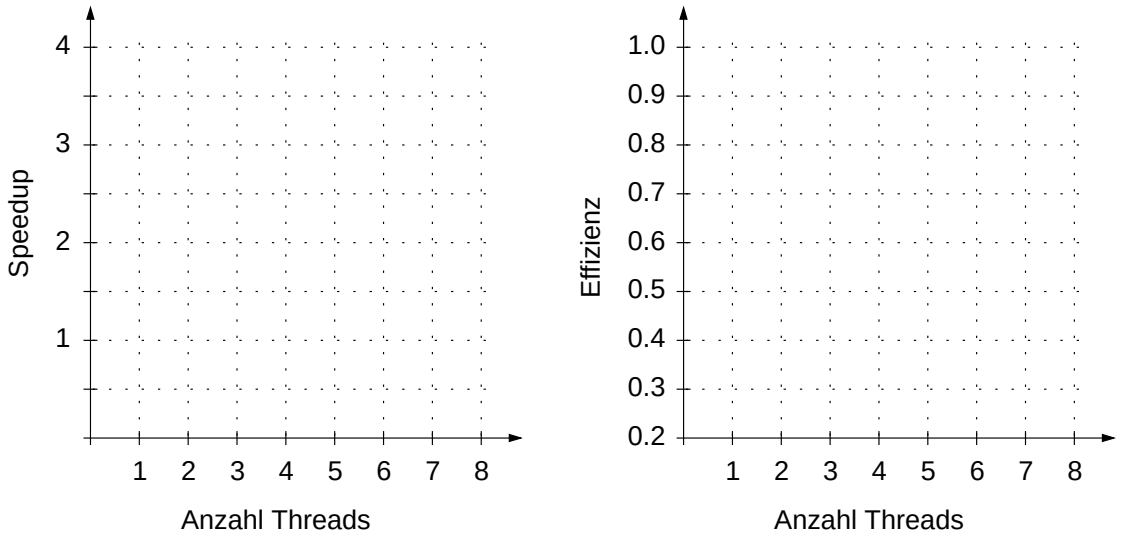
workload	Zeit	1	2	4	8
0	elapsed				
	user				
1	elapsed				
	user				

Was fällt auf? Diskutieren sie die Resultate für die Variante ohne und für die Variante mit Workload. Vergleichen sie die beiden Varianten: was kann aus dem Resultat geschlossen werden?

d) Für Vergleich und Evaluation von parallelen Programm werden Perfomance-Masse benötigt, die beiden wichtigsten sind der **Speedup** $S(P)$ und die Effizienz $E(P)$ mit P gleich Anzhal Prozessoren bzw. Threads. Speedup und Effizienz berechnen sich wie folgt:

$$S(P) = \frac{S(1)}{S(P)} \qquad E(P) = \frac{S(P)}{P}.$$

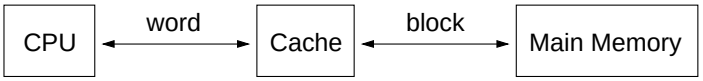
Tragen sie in die unten stehenden Diagramme den Speedup und die Effizienz der Messungen aus Aufgabe c) ein (elapsed time):



Diskutieren sie die resultierenden Kurven: was fällt auf?

4 False Sharing

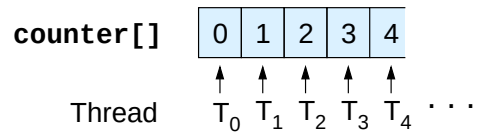
In der Speicherhierarchie eines Rechnersystems werden Datenzugriffe durch die CPU auf Blocktransfers zwischen Cache und Hauptspeicher abgebildet:



Der Datenblock wird auch `Cache Line` genannt und besteht aus mehreren Bytes. Ein Blocktransfer zwischen Cache und Speicher findet immer dann statt, wenn ein Datenwert geschrieben wird oder beim Lesen ein Datenwert. noch nicht im Cache steht.

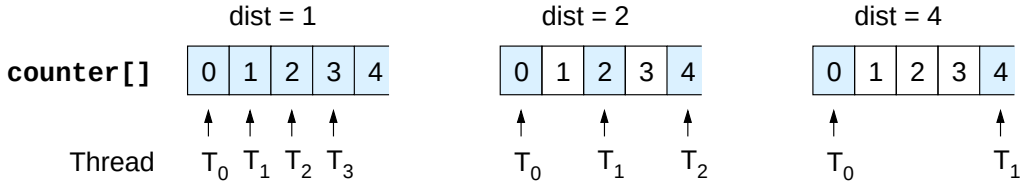
Greifen parallel arbeitende Threads auf Datenwerte zu, die in der gleichen Cache Line nebeneinander liegen, muss bei jedem Schreibzugriff die Cache Line in den Hauptspeicher zurückgeschrieben werden. Weil Threads diese Datenwerte nicht gemeinsam nutzen, sondern nur gemeinsam *handhaben*, spricht man von *False Sharing*.

Beispiel: ein globaler Array von Zählern, bei dem jeder Eintrag von einem anderen Thread inkrementiert wird:



Im Programm `main.c` (Verzeichnis `hardware/a4`) greifen die Threads mit *Array Padding* auf einen gemeinsamen Array zu. Die Array Indices der Threads referenzieren dabei nicht nur nebeneinanderliegende Zellen, sondern haben einen Abstand, der über die Thread ID (`id`) und die Distanz (`dist`) berechnet wird: `idx = id` $\times$ `dist`.

Mit den Distanzen `dist = 1, 2, 4, ...` referenzieren die Threads T_x die Zellen wie folgt:



Aufgaben:

a) Bestimmen sie die Rechenzeit des Programms, indem sie die Distanz beginnend bei 1 solange verdoppeln bis die Elapsed Time konstant bleibt. Die Distanz kann als Parameter beim Start des Programms übergeben werden, die Anzahl Threads wird gleich wie die Anzahl verfügbarer CPUs gewählt.

Distanz	1	2	4	8	16	32
Elapsed Time						

Aus wie vielen Bytes besteht eine Cache Line?

b) Wie könnte man im vorliegenden Fall False Sharing ohne *Array Padding* vermeiden? Testen sie Ihren Vorschlag?

Hinweis: das Hauptprogramm muss nach wie vor auf den Endstand der einzelnen Zähler zugreifen können.

5 Shared Data

Im Verzeichnis `hardware/a5` finden sie das Programm `main.c`, das mehrere Threads erzeugt, die solange in einer while-Schleife iterieren, bis die globale Variable **`spin`** durch das Hauptprogramm auf 0 gesetzt wird.

a) Übersetzen sie das Programm mit `make` und lassen sie es laufen.

b) Fügen sie im `makefile` bei `CFLGS` zusätzlich die Optionen `-O` ein und übersetzen sie das Programm mit **`make all`**. Starten sie `main.e` erneut: was geschieht?

c) Wie muss das Programm angepasst werden, damit dieser Effekt nicht mehr eintritt?

Hinweis: die Deklaration der Variablen `spin` muss erweitert werden?

6 Take Home

Was müssen sie bei der Programmierung von Parallelen Systemen beachten?

Notieren sie sich die wichtigsten Punkte, die sie im Zusammenhang mit diesem Praktikum kennengelernt haben.