PCA Exercise 04

Soheb Abdi, Raphael Lubaschewski

4.1.2 results

1 thread

n	result	measured t_compute (in microseconds)	measured t_wall (in microseconds)	absolute error
2	3.160417031779	347	578	0.018824378189347
3	3.143555466911	312	494	0.001962813321328
4	3.1417914776113	733	904	0.0001988240216168
5	3.1416126164019	5644	5846	0.0000199628122565
6	3.1415946524138	48551	48738	0.0000019988241208

2 threads

n	result	measured t_compute (in microseconds)	measured t_wall (in microseconds)	absolute error
2	3.160417031779	239	409	0.01882437818934
3	3.143555466911	331	531	0.001962813321326
4	3.1417914776113	500	664	0.0001988240216186
5	3.141612616402	2810	3016	0.000019962812302
6	3.1415946524138	38133	38412	0.0000019988241604

4 threads

n	result	measured t_compute (in microseconds)	measured t_wall (in microseconds)	absolute error
2	3.160417031779	292	464	0.01882437818934
3	3.143555466911	477	694	0.001962813321326
4	3.1417914776113	489	688	0.0001988240216186
5	3.1416126164019	2075	2245	0.000019962812302
6	3.1415946524137	11129	11303	0.0000019988241604

8 threads

n	result	measured t_compute (in microseconds)	measured t_wall (in microseconds)	absolute error
2	3.160417031779	582	808	0.01882437818934
3	3.143555466911	393	587	0.001962813321326
4	3.1417914776113	452	609	0.0001988240216186
5	3.1416126164019	1327	1530	0.000019962812302
6	3.1415946524137	5519	5671	0.0000019988241604

16 threads

n	result	measured t_compute (in microseconds)	measured t_wall (in microseconds)	absolute error
2	3.160417031779	863	985	0.01882437818934
3	3.143555466911	754	940	0.001962813321326
4	3.1417914776113	1450	1615	0.0001988240216186
5	3.1416126164019	1240	1404	0.000019962812302
6	3.1415946524137	7305	7533	0.0000019988241604

32 threads

n	result	measured t_compute (in microseconds)	measured t_wall (in microseconds)	absolute error
2	3.160417031779	2077	2275	0.01882437818934
3	3.143555466911	1624	1780	0.001962813321326
4	3.1417914776113	1656	1825	0.0001988240216186
5	3.1416126164019	2108	2303	0.000019962812302
6	3.1415946524137	7122	7308	0.0000019988241604

The best speed up can be achieved with 8 threads (roughly 7x faster as the sequential version). Spawning more threads introduces an overhead, which is noticeable when looking at smaller overall iteration sizes. This overhead is expected because the action of spawning a thread takes time, which is not needed in the sequential version of the program. Furthermore the mutex operations perform a system call implicitly, which is relatively time consuming. For n greater than 4, all multi threaded solutions achieve a noticeable speed up. For smaller overall iterations, the sequential version might be preferable because of the overhead.

4.2

a)

Der gravierende Unterschied zwischen den Threads und den Prozessen besteht darin, dass Threads unabhängige Befehlsfolgen innerhalb eines Prozesses sind. Threads sind in einem (oder mehreren) Prozessen gefangen oder verkapselt.

b)

Einen neuen Prozess zu starten kann sehr viel Zeit in Anspruch nehmen und sich negativ auf die Ausführung auswirken. Threads hingegen können erstellt werden ohne ganze Prozesse neu zu erstellen. Dadurch erspart man sich die Kommunikation und Synchronisation der Address-Speicher.

4.3

a)

MISD = multiple instruction single data stream

Weil mehrere (unterschiedliche) Kommandos auf das selbe Data-set (stream) ausgeführt werden und das für viele Anwendungen nicht geeignet ist. Aus diesem Grund ist die MISD Klassifikation von untergeordneter Bedeutung.

b)

Feldrechner bestehen aus n Rechenelementen, die alle auf dem gleichen Instruktionsstrom arbeiten, aber auf verschiedene Daten zugreifen.

Vektorrechner hingegen arbeiten nach dem „Pipeline-Prinzip“. Das Prinzip ist anwendbar, wenn komplexe Operationen aus mehreren aufteilbaren Rechenschritten bestehen. Dafür wird die Operation in möglichst gleichlange Teiloperationen zerlegt, die dann wie in einer Pipeline zeitsequentiell hintereinander in den verschiedenen Stufen bearbeitet werden