worksheet 4

by ***Konstantin Devyatov***

Part I. Problem 1.1

See folder **../1**

Part II. Problem 1.2

See folder **../2**

Part III. Problem 1.3

Measurements

number of threads / implementation	sequential (in s)	parallel (creation of 100K threads) (in s)	parallel (thread/100k iterations) (in s)
1	0.313,921	2.586,987	0.326,875
2	0.308,483	2.422,732	0.167,585
5	0.309,331	6.429,677	0.169,293
10	0.308,600	16.442,160	0.169,071
15	0.308,405	25.130,958	0.176,048
20	0.308,325	33.661,502	0.172,082
25	0.309,178	42.241,863	0.166,965
30	0.309,337	51.041,486	0.169,510

All tests have been performed on linux18 lab machine: centOS 6.3 x64, 3.7 GiB of RAM, dual-core 3 GHz CPU.

Observations

From the measurements i was able to observe that the technology behind pthread_create does not reap time-efficiency benefits on its own. Which means that programming ingenuity is still highly valued, especially so with the market approachign limitations of the processor clock speed.

Specifically, i saw that 2 threads provide the most time-efficient solution to this trivial problem. That is because the of two reasons:

- with the number of threads resource overhead increases and when n>2, the overhead starts to overtake the benefit gained by parallelisation
- once a thread is finished it need to wait for other threads, in arbitrarily selected order, to finish, before it could terminate; this also creates workload bubbles, decreasing overall time-efficiency

Part IV. problem 1.4

Measurements

number of threads / implementation	sequential (in s)	parallel (thread_join) (in s)	parallel (using thread barrier) (in s)
1	0.314,818	0.326,875	0.352,491
2	0.312,098	0.167,585	1.297,994
5	0.318,679	0.169,293	4.411,378
10	0.317,729	0.169,071	8.804,457
15	0.313,806	0.176,048	12.240,947
20	0.313,893	0.172,082	16.227,633
25	0.322,772	0.166,965	20.788,779
30	0.318,315	0.169,510	24.481,747

Observations

Introducing pthread_barrier provides data safety: a new iteration starts only after all threads of the previous iteration have finished execution.

However, this also increses execution time dramatically. Although, not as much as thread joining.