Quan Yuan

Result:

Note: time is calculated in seconds and real time is used for comparison, and all performance gain is compared with the basic serial one.

transactions	test 1	test 2	test 3	test 4	test 5	test 6	test 7	test 8	test 9	test 10	AVERAGE TIME
10,000	0.043	0.043	0.042	0.041	0.041	0.043	0.044	0.042	0.041	0.043	0.0423
100,000	0.401	0.414	0.402	0.402	0.413	0.410	0.410	0.413	0.409	0.406	0.408
1,000,000	3.910	3.905	3.957	3.981	3.922	3.959	3.902	3.917	3.975	4.012	3.944

Serial:

Parallel without synchronized

Two threads:

transactions	test 1	test 2	test 3	test 4	test 5	test 6	test 7	test 8	test 9	test 10	AVERAGE TIME
10,000	0.029	0.025	0.025	0.025	0.025	0.025	0.025	0.025	0.025	0.025	0.0254
100,000	0.222	0.220	0.222	0.221	0.221	0.221	0.221	0.222	0.222	0.220	0.2212
1,000,000	2.124	2.141	2.139	2.154	2.128	2.128	2.136	2.128	2.147	2.127	2.1352

Performance gain:

Transactions 10,000: (0.0423-0.0254)÷0.0423×100%= 39.95%

Transactions 100,000: (0.408-0.2212)÷0.4358×100%= 42.86%

Transactions 1,000,000: (3.944-2.1352)÷3.944×100%= 45.86%

Three threads:

transactions	test 1	test 2	test 3	test 4	test 5	test 6	test 7	test 8	test 9	test 10	AVERAGE TIME
10,000	0.018	0.018	0.018	0.018	0.017	0.017	0.018	0.018	0.018	0.018	0.0178
100,000	0.155	0.156	0.155	0.157	0.159	0.156	0.156	0.155	0.196	0.206	0.1651
1,000,000	1.567	1.497	1.481	1.518	1.563	1.483	1.490	1.523	1.480	1.485	1.5087

Performance gain:

Transactions 10,000: (0.0423-0.0178)÷0.0423×100%= 57.92%

Transactions 100,000: (0.408-0.1651)÷0.4358×100%= 55.74%

Transactions 1,000,000: (3.944-1.5087)÷3.944×100%= 61.75%

Four threads:

transactions	test 1	test 2	test 3	test 4	test 5	test 6	test 7	test 8	test 9	test 10	AVERAGE TIME
10,000	0.017	0.017	0.013	0.017	0.013	0.018	0.014	0.017	0.014	0.014	0.0154
100,000	0.141	0.160	0.121	0.121	0.121	0.121	0.120	0.122	0.123	0.121	0.1271
1,000,000	1.190	1.179	1.169	1.191	1.168	1.172	1.245	1.261	1.448	1.253	1.2276

Performance gain:

Transactions 10,000: (0.0423-0.0154)÷0.0423×100%= 63.59%

Transactions 100,000: (0.408-0.1271)÷0.4358×100%= 64.46%

Transactions 1,000,000: (3.944-1.2276)÷3.944×100%= 68.87%

Analysis:

There is almost “linear” performance gain with the increase of thread number.

Program runs almost twice as fast with two threads as it does with one and twice as fast with four threads as it does with two.

Reason is that the threads are running at the same time, with the same amount of data to be calculated, with the increase of thread number, time for program to execute decrease almost linearly, given enough cores for the threads.

Calculate race condition times:

Two threads:

transactions	**test 1**	**test 2**	**test 3**	**test 4**	**test 5**	**test 6**	**test 7**	**test 8**	**test 9**	**test 10**	**FAILURE TIMES**
10,000	0	0	0	0	0	0	0	0	0	0	0
100,000	0	0	0	0	0	0	0	0	0	0	0
1,000,000	0	0	1	1	1	1	1	1	1	1	8

Three threads:

transactions	**test 1**	**test 2**	**test 3**	**test 4**	**test 5**	**test 6**	**test 7**	**test 8**	**test 9**	**test 10**	**FAILURE TIMES**
10,000	0	0	0	0	0	0	0	0	0	0	0
100,000	0	0	0	0	0	0	0	0	0	0	0
1,000,000	1	1	0	1	0	0	0	1	1	1	6

Four threads:

transactions	**test 1**	**test 2**	**test 3**	**test 4**	**test 5**	**test 6**	**test 7**	**test 8**	**test 9**	**test 10**	**FAILURE TIMES**

10,000	0	0	0	0	0	0	0	0	0	0	0
100,000	0	0	0	0	0	0	0	1	0	0	1
1,000,000	1	1	0	0	0	0	1	1	0	1	5

Analysis:

Since it is not synchronized, race condition is very easy to see. As all threads have access to shared variables at the same time, likelihood for race conditions increase with increase of thread number. Even though program executes faster, result is not accurate.

Parallel and Synchronized:

Two threads:

transactions	**test 1**	**test 2**	**test 3**	**test 4**	**test 5**	**test 6**	**test 7**	**test 8**	**test 9**	**test 10**	**AVERAGE TIME**
10,000	0.057	0.051	0.058	0.054	0.052	0.054	0.051	0.055	0.057	0.060	0.0549
100,000	0.490	0.481	0.493	0.495	0.496	0.486	0.503	0.497	0.504	0.483	0.4928
1,000,000	4.753	4.783	4.923	4.927	4.709	4.686	4.718	4.788	4.709	4.775	4.7771

Performance decrease:

Transactions 10,000: (0.0549−0.0423)÷0.0423×100%= 29.79%

Transactions 100,000: (0.4928−0.408)÷0.4358×100%= 19.46%

Transactions 1,000,000: (4.7771−3.944)÷3.944×100%= 21.12%

Three threads:

transactions	**test 1**	**test 2**	**test 3**	**test 4**	**test 5**	**test 6**	**test 7**	**test 8**	**test 9**	**test 10**	**AVERAGE TIME**
10,000	0.056	0.075	0.062	0.062	0.058	0.071	0.061	0.060	0.056	0.058	0.0618

100,000	0.545	0.533	0.557	0.540	0.538	0.540	0.553	0.548	0.551	0.542	0.5447
1,000,000	5.389	5.238	5.183	5.244	5.238	5.256	5.211	5.246	5.209	5.283	5.2497

Performance decrease:

Transactions 10,000: (0.0618−0.0423)÷0.0423×100%= 46.10%

Transactions 100,000: (0.5447−0.408)÷0.4358×100%= 31.37%

Transactions 1,000,000: (5.2497−3.944)÷3.944×100%= 33.11%

Four threads:

transactions	**test 1**	**test 2**	**test 3**	**test 4**	**test 5**	**test 6**	**test 7**	**test 8**	**test 9**	**test 10**	**AVERAGE TIME**
10,000	0.074	0.073	0.085	0.093	0.092	0.072	0.087	0.061	0.059	0.055	0.0751
100,000	0.510	0.526	0.512	0.605	0.702	0.730	0.692	0.823	0.737	0.753	0.659
1,000,000	7.041	7.396	7.297	7.216	7.494	7.553	7.494	7.325	7.418	7.499	7.3733

Performance decrease:

Transactions 10,000: (0.0751−0.0423)÷0.0423×100%= 77.54%

Transactions 100,000: (0.659−0.408)÷0.4358×100%= 57.60%

Transactions 1,000,000: (7.3733−3.944)÷3.944×100%= 86.95%

Analysis:

The result is slower, compared with the unsynchronized one. The reason is since the shared part is locked, there is only one thread that can access it at the same time, so it is very slow. Besides, locking and unlocking take time. But there is no race condition now, and result is accurate.