Hoe Jun Leong A0214585B

Lab 2

10.

Used with matrix size 2000

IPCs

Num_threads	I7	xs
1	1.53	1.70
2	1.54	1.50
4	1.55	1.53
8	0.99	1.55
16	0.95	1.27
32	0.94	1.13
64	0.92	1.09
128	0.94	1.13
256	0.96	1.14

As the number of threads increases, the instructions per cycle decreases both machine types. The IPC drops significantly from 4 to 8 threads for i7 while for xs is a more steady drop

GFLOPs

Num_threads	I7	xs
1	0.21	0.17
2	0.42	0.29
4	0.81	0.55
8	1.04	1.07
16	1.00	1.48
32	1.00	1.76
64	0.99	1.79
128	1.01	1.84
256	1.01	1.82

For both machine types, the GFLOPs increases as num of threads increases. For i7, the GFLOPs increased from about 0.2 to 1 from thread count 1 to 8 and stays the same with increasing thread count. For xs, there's a steady increase of GFLOPs with thread count.

Based on data from question 12, the number of threads is not directly inversely proportional to the execution time, but there is a decrease in execution time with increase in thread count.

11. Within the #pragma omp parallel clause, index the matrices in the following way: result.element[i][k] += a.element[i][j] * b.element[j][k];

Instead of computing the full sum of products for result[i][j] within one round of iterations of the 3rd nested for loop (k loop), I iterated through the rows of both A and B and switched the cell that the product is added to for each iteration of k loop.

12.

xs-4114 with matrix size 2000

Num_threads	Normal implementation/seconds	Row-wise implementation/seconds
1	95.07	50.79
2	56.90	26.10
4	28.23	13.72
8	14.53	7.16
16	10.39	6.56
32	8.82	5.96
64	8.69	5.85
128	8.53	5.85
256	8.44	5.85

Open MP can better parallelize the matrix multiplication in the row wise implementation instead of the column wise implementation. One possible reason could be due to the locality of data. The rows should be stored as contiguous data, unlike columns, so the same chunk doesn't have to be loaded into the cache as often as the column access where possibly there's a lot more cache misses and RAM access to get the next column value. If that's the main bottleneck, that would mean the row-wise implementation would be faster.