Date: 17/03/2016

ME766 - HW2 Report

Matrix multiplication using openMP and MPI frameworks

Yash Bhalgat | 13D070014

Experiments:

I ran the code on the server hosted by the Electrical Department on 10.107.1.5
The following are the machine specs (obtained using the 'lscpu' command):

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	4
On-line CPU(s) list:	0-3
Thread(s) per core:	1
Core(s) per socket:	4
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	60
Stepping:	3
CPU MHz:	800.000
BogoMIPS:	5985.88
Virtualization:	VT-x
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	6144K
NUMA node0 CPU(s):	0-3

I had run the code for N=100, 500, 1000, 2000, 5000, 10000
I was able to get sufficient parallelism till the no. of threads reached 4. After that, there wasn't enough speedup in the execution. This is compliant with the fact that the number of threads per core is 4, hence the result.

I used the function *MPI_Wtime()* from the MPI library to obtain exact time of execution.
I ran the code for **two different initialisations:**
random and test matrix given by Prof. Gopalakrishnan

Test matrix: for i,j=1:N:

Aij=i+j;

Bij=i×j;

Date: 17/03/2016

Results:

I. OpenMP:

Set the max number of threads (eg: 2) by running:

```
export OMP_NUM_THREADS=2
```

To compile the code, run the command:

```
mpic++ -fopenmp matrix_mul_openMP.cpp -o matrix_mul_openMP
```

Then run the code by just executing: `./matrix_mul_openMP`

Random initialisation:

No of Threads ->	1	2	3	4	6	8
N = 100	0.014719	0.009849	0.007297	0.005852	0.005966	0.005437
N = 200	0.121882	0.067444	0.059232	0.034547	0.035515	0.036952
N = 500	2.5453	1.3315	1.2757	0.7566	0.7871	0.7235
N = 1000	30.2888	14.1304	12.6788	7.1667	7.4358	7.7732
N = 2000	288.442	126.535	115.570	61.6679	63.5386	62.9978
N = 5000	3437.56	1672.37	1562.91	877.24	859.52	889.40
N = 10000	9152.54	4852.73	4188.96	2552.64	2461.14	2492.94

Using the test matrix:

No of Threads ->	1	2	3	4	6	8
N = 100	0.015663	0.010117	0.009997	0.009852	0.009536	0.009213
N = 200	0.114532	0.064131	0.058994	0.036252	0.037663	0.039246
N = 500	2.5453	1.3315	1.2757	0.7566	0.7871	0.7235
N = 1000	26.6634	12.5622	11.3311	6.1374	7.3215	6.4214
N = 2000	262.588	126.535	122.584	66.5855	63.1517	65.5443
N = 5000	4487.25	2481.73	1857.51	1318.27	1332.96	1315.58
N = 10000	12515.54	6051.58	5888.46	3233.64	3531.52	3137.87

Date: 17/03/2016

II. MPI:

I ran the code for N = 100, 500, 1000, 5000, 10000 and noted the results
To compile the code:

```
mpic++ matrix_mul_MPI.cpp -o matrix_mul_MPI
```

Then, you can run the code by (for 6 threads):

```
mpirun -np 6 ./matrix_mul_MPI
```

Random initialisation:

No of Threads ->	1	2	3	4	6	8
N = 100	0.009272	0.006204	0.004597	0.003686	0.003758	0.003425
N = 500	1.323556	0.69238	0.663364	0.393432	0.409292	0.37622
N = 1000	18.4761	8.61954	7.73406	4.37168	4.53583	4.74167
N = 5000	2096.91	1020.14	953.375	535.116	524.3072	542.534
N = 10000	5583.04	2960.16	2555.26	1557.11	1501.29	1520.69

Using the test matrix:

No of Threads ->	1	2	3	4	6	8
N = 100	0.008437	0.005645	0.004183	0.003354	0.003419	0.003116
N = 500	1.111787	0.58159	0.55722	0.33048	0.34380	0.31602
N = 1000	15.5199	7.24041	6.49661	3.67221	3.8100	3.983
N = 5000	1761.4	856.917	800.835	449.497	440.418	455.728
N = 10000	4689.75	2486.53	2146.41	1307.97	1261.08	1277.37

We can see that, we get more and more perfect parallelisation as the size of the matrix increases. So, the execution time decreases more proportionately in case of N=5000 than N=200.
Also, we can observe that the speedup gained in the MPI framework is more than the openMP framework

---

Thank you!