

1)

A) lscpu of zeus...

```
[deo15@zeus Project 5]$ lscpu
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:    1
Socket(s):              4
NUMA node(s):          1
Vendor ID:              GenuineIntel
CPU Family:             6
Model:                  44
Model name:             Westmere E56xx/L56xx/X56xx (Nehalem-C)
Stepping:               1
CPU MHz:                2393.998
BogoMIPS:               4787.99
Hypervisor vendor:     KVM
Virtualization type:    full
L1d cache:              32K
L1i cache:              32K
L2 cache:               4096K
NUMA node0 CPU(s):     0-3
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
                        clflush mmx fxsr sse sse2 syscall nx lm constant_tsc rep_good nopl pni pclmulqdq ssse3 cx16 ss
                        4_1 sse4_2 x2apic popcnt aes hypervisorlahf_lm
```

C)

ArraySize (Bytes)	Row Compute Time (Seconds)	Row (Mega_elements/sec)	Column Compute Time (Seconds)	Column (Mega_elements/sec)
4000000	0.0060	166.226	0.0086	116.508
100000000	0.1674	149.333	0.4106	60.887
576000000	1.1251	127.987	4.1146	34.997
1600000000	3.6404	109.878	22.4997	17.778

Row first computes faster than Column first and computes more mega_elements/sec than column first. As the size grows the compute times increase dramatically in column first.

D)

In my code I tried to simulate the cold and warm cache by looping my code multiple times with different sizes but I was not able to see any real differences in the timings.

2)

A)

Frequency of error: 4/10000

Erroneous result: Sum = 300000.000000

Why? With a lack of synchronization, threads may be accessing data while others could be writing to that same data thus giving erroneous results.

Lock prevented all erroneous results!

B)

	Runtime (seconds)
Serial	0.0349
pThreads	0.0161

3)

A)

```
[deo15@zeus Project 5]$ ./dp
DOT_PRODUCT
C/OpenMP version

A program which computes a vector dot product.

Number of processors available = 4
Number of threads = 4

Sequential 1000 1.000000e+03 0.0000055395
Parallel 1000 1.000000e+03 0.0002072006

Sequential 10000 1.000000e+04 0.0000600005
Parallel 10000 1.000000e+04 0.0000222400

Sequential 100000 1.000000e+05 0.0005479790
Parallel 100000 1.000000e+05 0.0002618246

Sequential 1000000 1.000000e+06 0.0056165904
Parallel 1000000 1.000000e+06 0.0041086748

DOT_PRODUCT
Normal end of execution.
```

Vector Length	Speed up	Efficiency
1000	0.026734961	0.00668374
10000	2.733835432	0.683458858
100000	2.092924042	0.52323101
1000000	1.367007776	0.341751944

B)

‘i’ is private because it wouldn’t make sense for each thread to be working on the same index of the array as you would want to parallelize the array for more efficiency.