Xian Wu 903142009

Question 4:

My Computer:

```
*******************************************************************************
lisa@lisaMbp:~/Downloads/hw1$ 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:    2
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 61
Stepping:              4
CPU MHz:               1410.804
BogoMIPS:              5799.82
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              3072K
NUMA node0 CPU(s):     0-3
*******************************************************************************
```

And this is all my code revised:

```
if (threads == 1) {
   time_spent=0;
   for(int count=0;count<20;count++){
   start = omp_get_wtime();
   sorted = bubble_sort_serial(unsorted,num_elems);
   end = omp_get_wtime();
   timer[count]=(double)(end-start);
   time_spent+=(double)(end-start);
   }
   time_spent/=20.0;
  } else {
   time_spent=0;
   for(int count=0;count<20;count++){
   start = omp_get_wtime();
   sorted = bubble_sort_parallel(unsorted,num_elems,threads);
   end = omp_get_wtime();
   timer[count]=(double)(end-start);
   time_spent+=(double)(end-start);
   }
```

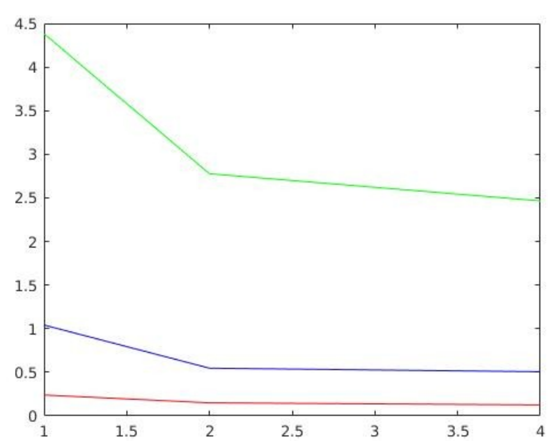
```
  time_spent/=20.0;
 }

 double tmp=0.0;
 for(int i=0;i<20;i++){
    tmp+=pow(timer[i]-time_spent,2);
 }
 tmp=pow(tmp,0.5)/time_spent;
 printf("variance %lg \n",tmp);
 printf("Time Spent %lg \n",time_spent);
```
****

Fluctuation:
I calculated the average of 20 times experiment to get each value in the table to reduce the fluctuation effect. Also I calculated the variance for each problem size. If (variance/average) is too large, the set of data for this problem size could be considered unreliable.

P \ N	1000	10000	20000	40000	60000	80000
1	0.00221152	0.238954	1.04162	4.38342	10.2365	17.9679
2	0.00214644	0.149752	0.54677	2.77651	5.17902	9.88461
4	0.00252255	0.12589	0.506221	2.46537	4.99839	9.72908
6	0.0100587	0.223523	0.793997			
8	0.0136448	0.275474	0.908305			

For problem size 1000, the variance/average ratio could be from 1.5 to 2.5, so we should not consider the 1000-column data reliable. There is serious fluctuation.

Strong Scaling
By keeping the problem size and varying the number of threads:
You can see clearly that within number of threads no more than 4, the more threads used the less runtime costed. But runtime does not reduces linearly, which is due to the overhead for processors to communicate and synchronize.
Also as for why a thread number larger than 4 fails to make it faster, it is because my computer has just 4 CPU cores, if you uses more than 4 threads, CPUs have to do context switch to simulate the multi-processor situation, which exactly increases the latency.
The speedup would be
See the graph below:



The x-label is number of processors and the y-label is the time spent.

Weak Scaling

By keeping the processor size proportional to the problem size and calculating the speedup:

P \ N	10000	20000	40000
1	0.238954	1.04162	4.38342
2		0.54677	
4			2.46537
speedup	--	1.9050	1.7780

Since there is not enough data for ploting a gragh, but you can still see a speedup around 1.9.
Here the sequential part of the program is removed out of consideration. So in the ideal case the speedup should be :

speedup=N times the problem size grows

However, here about the bubble sort algorithm, with single processor, when the problem size grows, the runtime does not increases linearly, which made the speedup of 40000-size large problem even less than 2.