Lab1 5451

MingxuanJiang
5588030

Feb 29th 2020

1 Question1

1.1

When calculate the matrix C, it is obvious that openMP needs to be inserted. Set openMP parallel for private i0 (tasks for each thread), i1 (rows of A), i2 (columns of B), i3 (columns of B). Then, determine the tasks for each thread to do. Let *nt* be the number of threads used. For example, if *i0* is equal to 0, then *i1* is the loop from 0 to $\frac{m}{nt}$. It means that each thread does $\frac{m}{nt}$ tasks.
When the number of threads is 32, $\frac{2000}{32}$ is equal to 62.5, which isn't an integer. In order to correct this, I define a variable called *divide*. If $\frac{m}{nt}$ isn't an integer, let *divide* is equal to $[\frac{m}{nt}]+1$. So each thread is able to run almost the same number of tasks and none of the tasks is left. They are now parallel and speeds up.

1.2

When using 1 thread, time = 6.523922
When using 4 threads, time = 1.969409
When using 8 threads, time = 1.170179
When using 16 threads, time = 0.7983141
When using 32 threads, time = 0.5867541

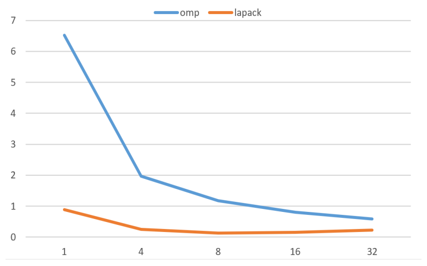

Figure 1: The time used

1.3

Using 2 operations: * and +
the Best Flops Rate = $\frac{2000*2000*2000*2}{0.5867541} = 27GFLOPS$

1.4

If define the size of matrix as 4000 and the number of threads as 32, the time should be 8*0.586= 4.688. Actually the time is 4.440, smaller than 4.688. We can see the performance doesn't drop. This is because except calculating time, there exists start-up time and per-hop time. We suppose the calculating time and per-word transfer time is always the same. But with the increasing number of the dimension of matrix, the proportion of time spent at the start-up time and per-hop time will become smaller and smaller. Therefore, the performance doesn't drop.

2

The execution result of testSaxpy

```
** vecLen = 1024, Mflops = 304.39, err = 1.42e-06
** vecLen = 2048, Mflops = 613.13, err = 9.24e-07
** vecLen = 4096, Mflops = 1306.45, err = 1.06e-06
** vecLen = 8192, Mflops = 2694.88, err = 1.00e-06
** vecLen = 16384, Mflops = 4755.67, err = 8.29e-07
** vecLen = 32768, Mflops = 10851.87, err = 1.78e-06
** vecLen = 65536, Mflops = 20769.01, err = 2.43e-06
** vecLen = 131072, Mflops = 46159.18, err = 2.14e-06
** vecLen = 262144, Mflops = 89318.57, err = 1.99e-06
** vecLen = 524288, Mflops = 166972.16, err = 2.74e-06
```

The execution result of testSaxpyC

```
** vecLen = 1024, Mflops = 39.69, err = 2.75e-09
** vecLen = 2048, Mflops = 69.78, err = 9.76e-11
** vecLen = 4096, Mflops = 112.78, err = 3.22e-10
** vecLen = 8192, Mflops = 170.26, err = 2.01e-09
** vecLen = 16384, Mflops = 214.12, err = 2.31e-08
** vecLen = 32768, Mflops = 248.08, err = 2.78e-10
** vecLen = 65536, Mflops = 262.43, err = 3.15e-09
** vecLen = 131072, Mflops = 256.11, err = 2.99e-09
** vecLen = 262144, Mflops = 237.94, err = 9.53e-10
** vecLen = 524288, Mflops = 204.30, err = 2.40e-09
```

Explanation:
Data transfers excluded or included in timing has quite different results. With data transferring excluded in timing, Mflops starts from 304 and ends with 166972. While with data transferring included in timing, Mflops starts from 39

to 204. Therefore, we can make a conclusion that time using in data transferring from GPU to CPU takes a lot of time. Also, we can see the error between saxpy_par and saxpy_check almost equal to zero. This means our calculation is correct.