FIT5139 Advanced Distributed

and Parallel System

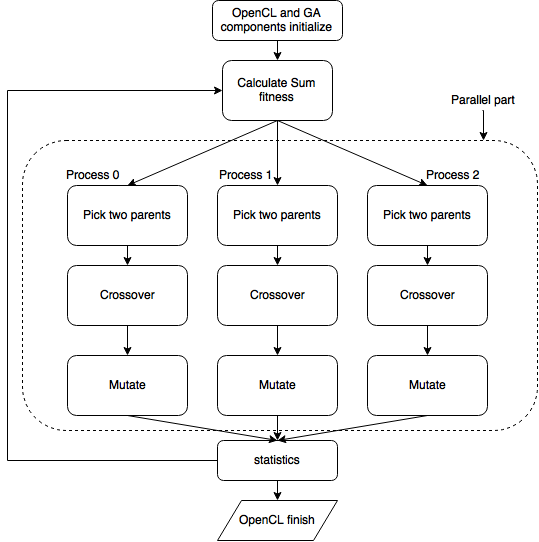
Assignment Part B report

Yuxuan Lin

26575809

2016 15 MAY

1. The program structure



In brief, this OpenCL program changes the generation function in sequential program into a kernel of the new OpenCL program.

At the “OpenCL and GA components initialize” stage, it needs to initialize all components used by GA and OpenCL kernel. The OpenCL part includes:

1. Discover and initialize the platforms;
2. Discover and initialize the devices;
3. Create context;
4. Create a command queue;
5. Read kernel file;
6. Create and compile program;
7. Create the kernel;
8. Configure the work-item structure;
9. Create device buffers.

The GA initialize some important variables and structures from “sga3.var” file like: population size, old population container, new population container, chromosome length, max generation, probability of crossover, probability of mutation, max individual fitness, min individual fitness, average individual fitness, sum of all fitness, and number of chromosome in one individual

Then we get in the “Calculating Sum fitness” stage. As we can see it is a beginning of a huge loop which occupies the body of this program. It calculates the sum of all fitness from generation. This value is used for picking parents for further steps like crossover and mutation.

Then we pass this value along with population and other data into kernel, starting the parallel computing.

Each process gets the data and individually picks two parents from the population passed in. Then they individually mix two parents’ gens including N chromosome in a user set probability from sga3.var file, and also decide whether and where to mutate in a setting probability.

After all the parallel part finish, OpenCL parallel kernel running end for a moment and pass back the generated new population back to main program. Then “statistics” stage begins to calculate this new population’s related statistic data. For now, this loop finishes one whole loop.

Then we back to the “Calculate sum fitness” stage and loop that part again and again until reaching the max generation.

1. The OpenCL based parallel GA scheme

This OpenCL based parallel GA scheme uses a “Master-slave” model to utilize the devices’ parallel computation ability to accelerate the running process. Typically, here I draw out all the parts which can use slave processes to separately compute results, and collect results after slave processes finished back to main process.

I check the GA C code and consider which parts can be paralleled, or transferred into kernel. The sum of fitness is not appropriated be assigned into slave processes and running in parallel. Because it can be done in single process and the result is singularity. If I put this part into kernel it should be a burden for slave processes to do because it could be finished by one process and the calculation result can be passed into slave processes to do more jobs. So I eliminate this function.

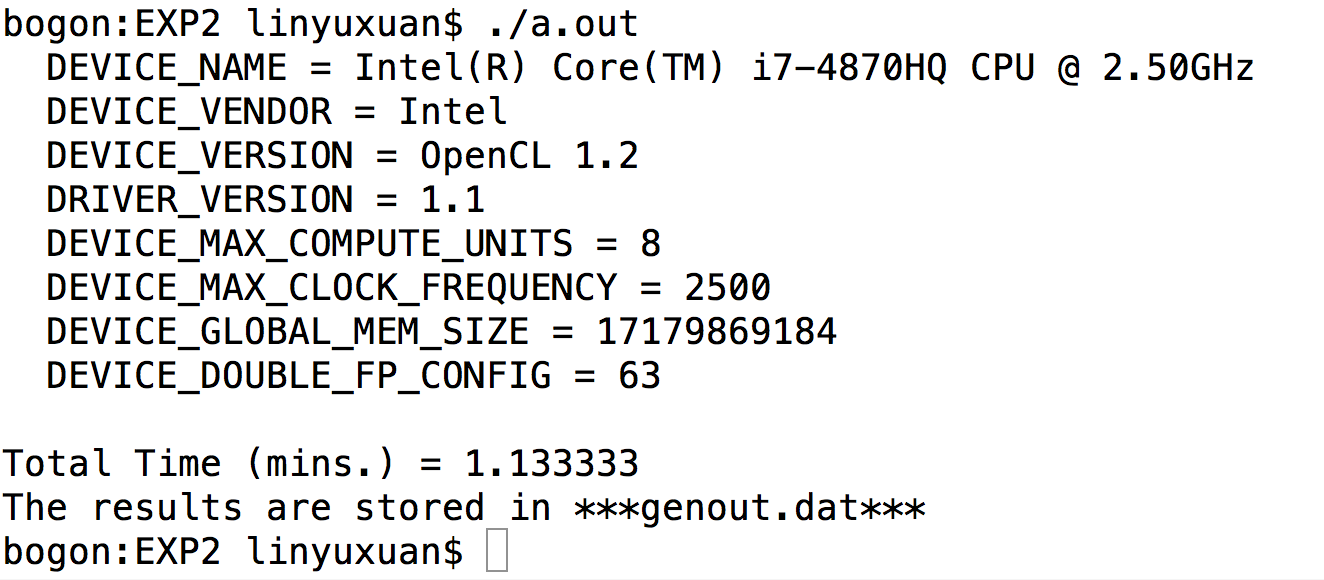
Then the parents picking part is obviously can be paralleled into slave processes because it aims to generate a set of parents’ part, and this iteration operation doesn’t rely on the result of the previous iteration. And the following crossover and mutation also should be scheduled into kernel being processed individually.

In the end, the statistics part is better to stay in main process sequentially.

1. Performance metrics that include a comparison of the execution time for the parallel GA code versus the original GA code (provided with this assignment). The choice of other metrics will be at the discretion of each student.

OpenCL:

Running time: 1.133333 Min.

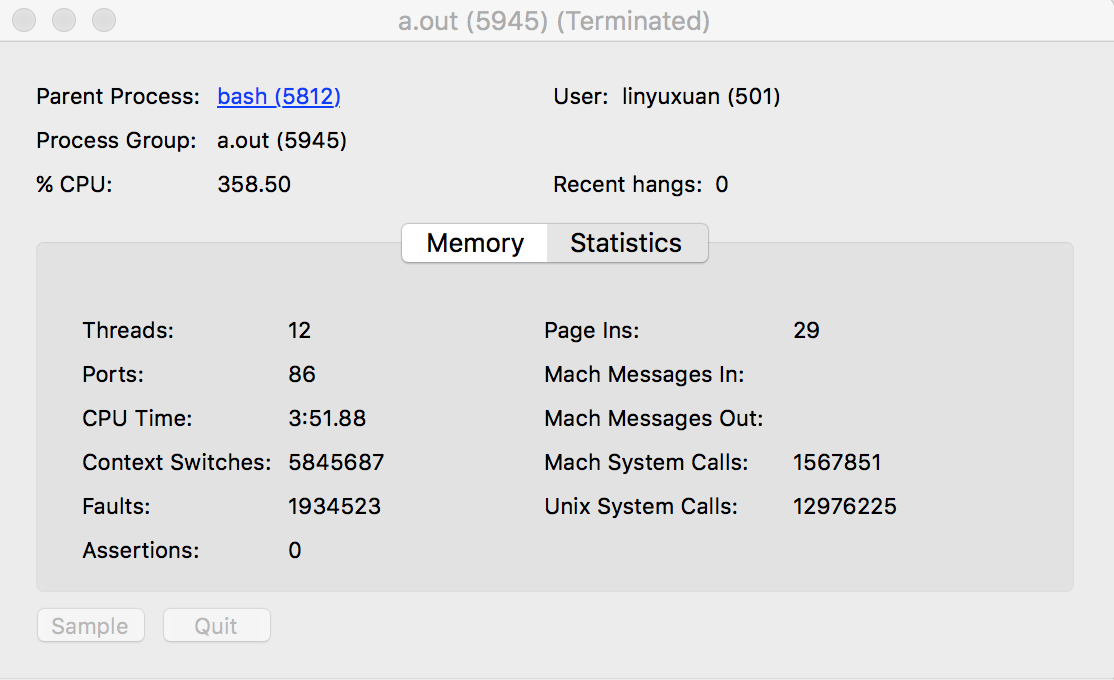


CPU utility %: 358.5

Number of times it has been stored in memory: 29

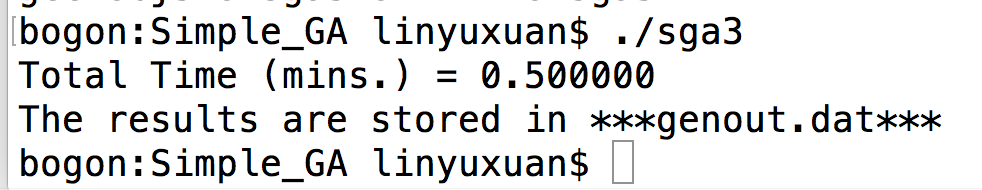
Threads used: 12

Faults triggered in paging: 1934523



Original GA code:

Running time: 0.5 Min.

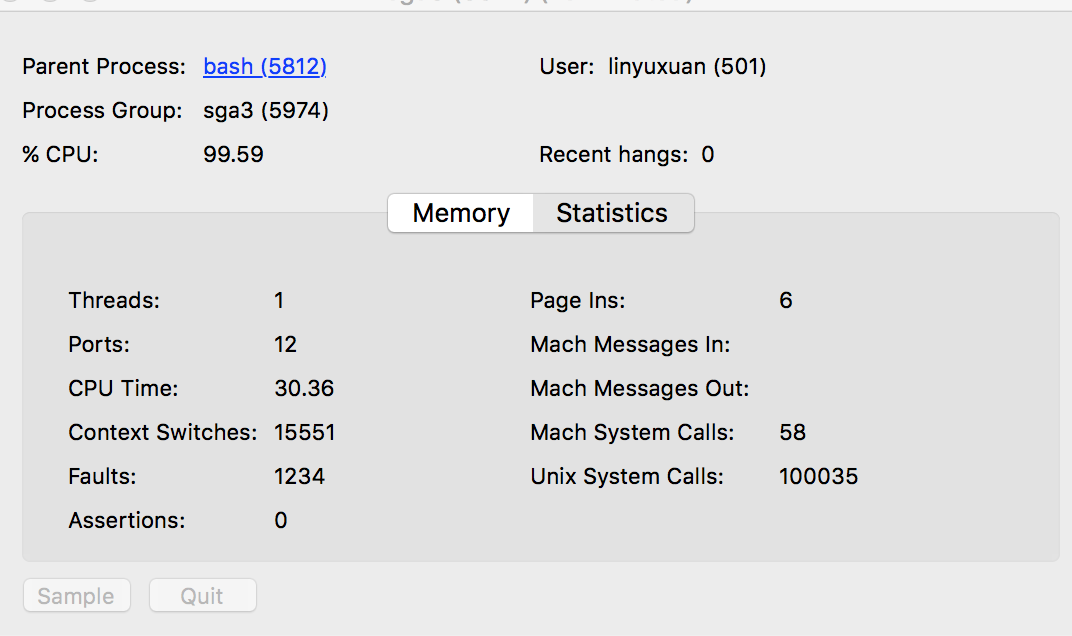


CPU utility %: 99.59

Number of times it has been stored in memory: 6

Threads used: 1

Faults triggered in paging: 1234



From the compare above we can see in this typical GA code, the OpenCL implementation and “Master-Slave” mode can’t boost the sequential code’s speed, and conversely increase running time, resources using and complexity.

The reason why this could happen is this OpenCL implementation contains tons of data read/write from host to device. For each iteration a writing and reading data operations pair happens at the begin and the end of the iteration. So the resources and time wasted in data transmission is a lot.

When I do this assignment and design OpenCL GA parallel scheme, I also find some papers proposing “Community” way to make sequential algorithm into a parallel program. It claims that divide the original population into several groups, and pass the groups into several processes to let them do all iteration inside the kernel, and exchange the gens with highest fitness regularly. In my imagination and experience from others’ work, this “Community” parallel GA should be much more effective than “Master-Slave” mode parallel GA. However, I choose the “Master-Slave” mode for curiosity of how slow can it be? Obviously I’ve got the answer. I’ll try OpenMPI + “Master-Slave” mode and OpenCL + “Community” those two modes combinations in winter holiday to take further discoveries into parallel programming world.