Lab Session 2

Submission deadline: Feb 13, 11:59pm

Please submit your lab results through CatCourses, including Makefile, source code, and a short report (up to one page). You may find the OpenMP tutorial from Lawrence Livermore National Lab useful (https://computing.llnl.gov/tutorials/openMP/).

1. **OpenMP hello world**

The first lab task aims to tell you how to compile and run a simple OpenMP program.

Copy the following program into a file named "openmp_simple.cc"

```
#include <iostream>
#include "omp.h"
int main() {
  #pragma omp parallel
 {
   std::cout << "Hello World\n";
 }
 return 0;
}
```

Compile the above program

```
g++ -fopenmp -o openmp_simple openmp_simple.cc
```

Change the number of OpenMP threads by setting the OMP environment variable

```
export OMP_NUM_THREADS=4
./openmp_simple
```

2. **Hello world extension**

   Extend the program above to make it parallel where every thread prints out its id and total number of threads.

   Hints: using APIs, omp_get_num_threads() and omp_get_thread_num(); See the above OpenMP tutorial.

3. **Using PAPI**

   PAPI is a tool that allows programmers to see the relationship between program performance and processor events (e.g., L1 cache misses) at runtime. It is a common tool to study performance problems in parallel programs.

   Here is a brief overview of PAPI (http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:Overview).

Here is an introduction to PAPI to help you get started.
(http://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:Getting_Started)

You will need to use the Chameleon server to use PAPI.

How to log into the Chameleon server:

1. Generate a local public ssh key in your computer
   $ssh-keygen

2. Send the generated public key ("~/.ssh/id_rsa.pub") in your computer to our TA Jiawen Liu (jliu265@ucmerced.edu) and log into the server after the response from the TA.

3. Log to the Chameleon server:
   $ssh your_account@129.114.109.4
   a. Your account is the first letter of your first name plus your last name. All letters are lower cases. For example, the account for the student "Jiawen Liu" is "jliu".
   b. Your password is the same as your account.

4. Please send an email to TA (jliu265@ucmerced.edu) if you have any questions regarding to the account/connection.

Install PAPI:

1. Download papi from CatCourse.
2. Download papi example: high_level.c
3. Copy high_level.c and papi_install.tar from your computer to the server (your_account@129.114.109.4) in the directory (/home/your_account/) by "scp":
   a. $scp high_level.c your_account@129.114.109.4:/home/your_account/
4. Unzip the papi_install.tar
5. Compile file with papi: $gcc high_level.c -o high_level –I/home/your_account/papi_instal/include /home/your_account/papi_instal/lib/libpapi.a
6. papi_avail is a tool to help you identify available PAPI events on your hardware. To use $papi_avail, set up the path by $export PATH=/home/your_account/papi_instal/bin:$PATH
7. Copy lulesh folder from your computer to the server:
   a. $scp –r lulesh your_account@129.114.109.4:/home/your_account/

Your task: Using PAPI APIs to collect the following two events for the main computation loop (Lines 2771-2780) of lulesh2.0 (see Task 3 in Lab1 for lulesh2.0): (1) last level cache misses; and (2) number of cycles stalled waiting for memory writes. The follow code gives you an idea on where to start and end event collection.

```
…
#include "papi.h"
…

 /*Setup PAPI library and begin collecting data
 from the counters before the main computation loop*/
int retval;
long long values[2];
int Events[2] = {PAPI_TOT_CYC, PAPI_TOT_INS};   //replace the two events with your own ones
```

```
 retval = PAPI_library_init(PAPI_VER_CURRENT);
if (retval != PAPI_VER_CURRENT)
      exit(-1);

/* Start counting */
if (PAPI_start_counters(Events, 2) != PAPI_OK)
      exit(-1);

 /*Main computation loop*/
 while((locDom->time() < locDom->stoptime()) && (locDom->cycle() < opts.its)) {
          …
 }

/* Stop counting */
if (PAPI_stop_counters(values, 2) != PAPI_OK)
 exit(-1);

/*read values and print*/
```

Use the environment variable to change the number of OpenMP threads, and measure the above two events.

Hints: To determine the PAPI event names for the two events, use “papi_avail”. If your computer cannot measure the event “number of cycles stalled waiting for memory writes”, replace it with the event “last level cache miss”.