18-645: How to write fast code, Spring 2015

Project #1 – Multicore Optimization

Due: February 4, 2015, 11:00pm EDT, 8:00pm PDT

The goal of this project is to use your understanding of parallel computing resources in a multicore microprocessor to optimize two fully functional applications. The applications are Matrix Multiple and K-means Clustering.

For a functional description of the applications, please refer to:

http://en.wikipedia.org/wiki/Matrix_multiplication

http://en.wikipedia.org/wiki/K-means

The code optimization techniques you may want to consider are explained in Module 1 and Module 2 which includes:

- Cache blocking
- OpenMP pragma-based optimizations
  - omp parallel
  - omp for
  - omp atomic
  - omp reduction
- Intrinsics Programming

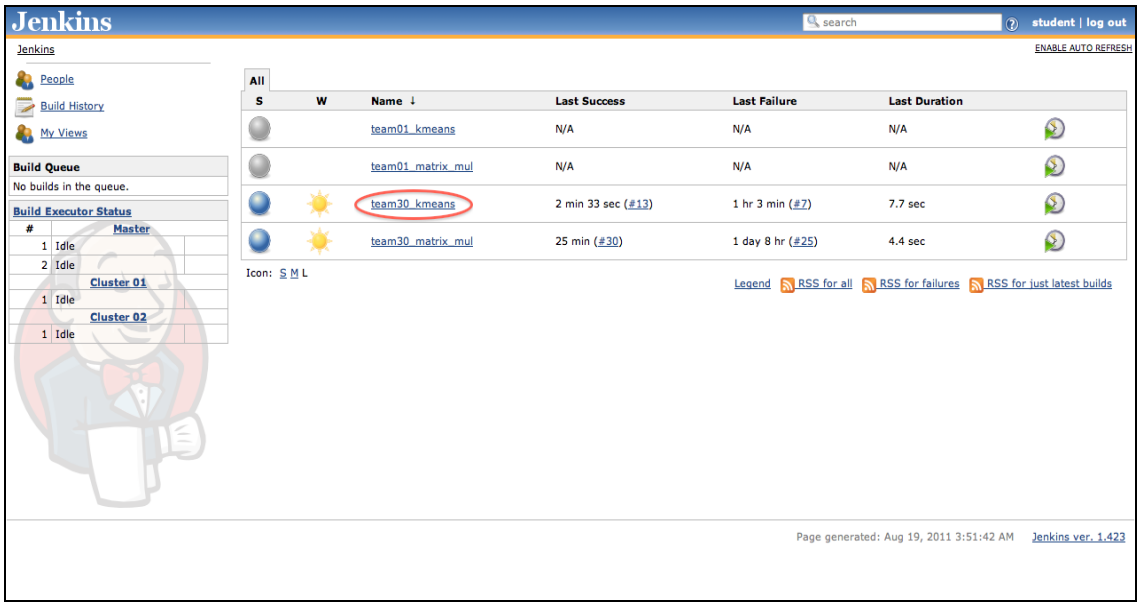
Task 0: Look at Jenkins

Jenkins is an Open Source software production management platform. It provides a web interface to keep track of the progress of the projects.

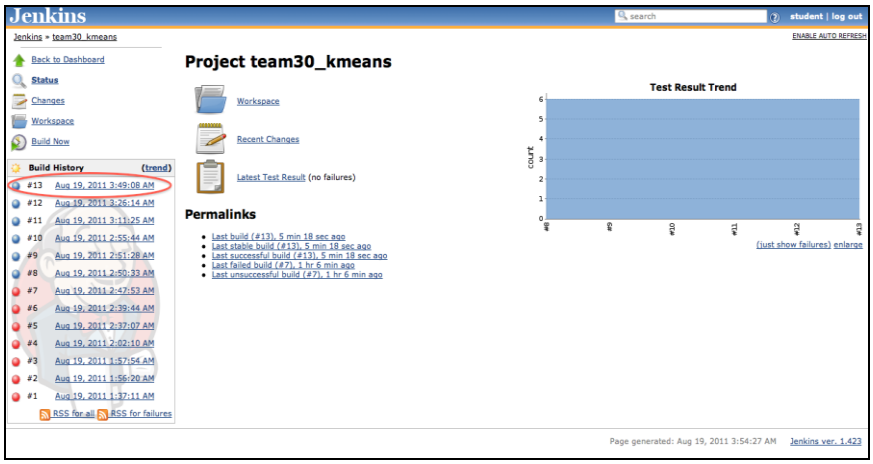
Visit
http://fast645.info:8080

User: student
Password: 18645

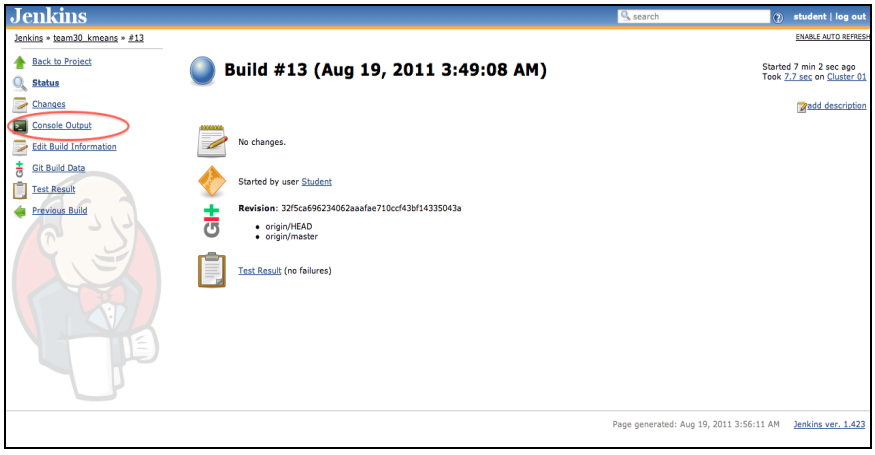
All students can view all team's results. For example to view Team30's execution results, click on the team030_kmeans project:



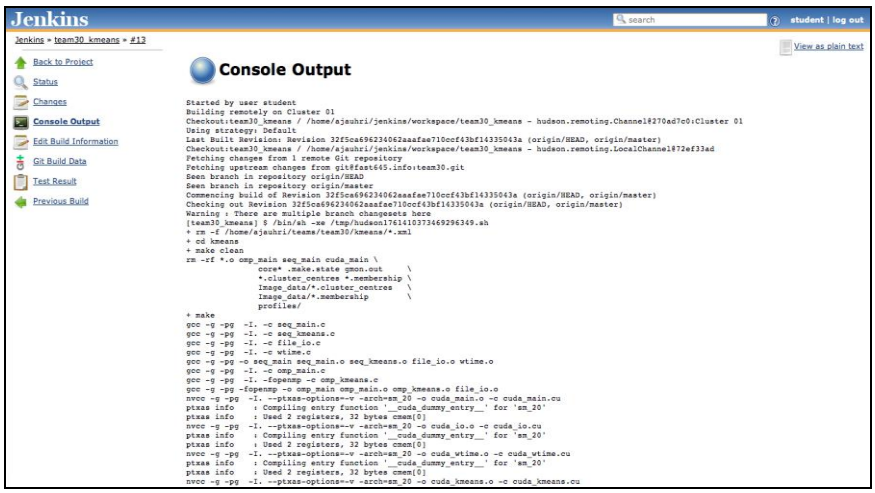
Then, click on a successful build:



To view the performance, use the Console Output link:



The performance would then be available:



Please check the console output for `matrix_mul` &`kmeans` algorithms, and answer questions 1 & 2.

Question 1: For your team's `matrix_mul` application, please report the runtime for sequential and OpenMP implementations.

Question 2: For your team's K-means project, please report:

a) The configuration of the K-means algorithm being tested
   i.e. `numObjs, numCoords, numClusters, threshold`

The runtime for I/O and computation for sequential and OpenMP implementations. (Note: The console output might show results of multiple input files. You are expected to report runtime for any one input file)

Step 1: Cloning a *git* repository

You can get the initial version of the code using (**already done if HW1 is complete**):

```
$ cd ~/645/fastcode
$ git pull origin master # To make sure you have the latest version of the code
```

The structure of the project

```
matrix_mul:
    Makefile
    matrix_mul_01.dat
    matrix_mul_02.dat
    cuda
        Makefile
        matrix_mul.cu
        matrix_mul.h
        tests.cpp
    omp
        Makefile
        matrix_mul.cpp  <<== To optimize
        matrix_mul.h
        tests.cpp
    sequential
        Makefile
        matrix_mul.cpp
        matrix_mul.h
        tests.cpp
    tests
        testutil.h

kmeans
    LICENSE
    Makefile
    README
    benchmark.sh
    cuda_io.cu
    cuda_kmeans.cu
    cuda_main.cu
    cuda_wtime.cu
    file_io.c
    gmon.out
    go
    kmeans.h
    omp_kmeans.c    <<== To optimize
    omp_main.c
    sample.output
    seq_kmeans.c
    seq_main.c
    wtime.c
    Image_data
```

Step 2: Code optimization

For project "`matrix_mul`" OpenMP implementations, please apply your optimization only to the content of the "`matrix_multiplication`" function in the file "`matrix_mul.cpp`".

For project "`kmeans`" OpenMP implementations, please make your changes only to the content of the "`omp_kmeans`" functions in files "`omp_kmeans.c`". Testing data for kmeans can be downloaded from below, assuming that you are working from a GHC Machine:

```
$ # To be performed by each team member
$ cd ~/645
$ wget https://cmu.box.com/shared/static/6odccg52untmq8p9szsk24m7wr84ptz2.gz -O 18645_spring_15_data.tar.gz
$ tar –xzf 18645_spring_15_data.tar.gz
$ # This should create a folder named data within ~/645
```

NOTE: Please DO NOT add kmeans testing data (kmeans01.dat ~ kmeans04.dat) to your git repository. They bloat the repository.

NOTE: It is crucial that your DO NOT change the interface. Any changes in the interface could result in your work not testable by our test infrastructure and you will receive no credit.

To compile the code, simply type "Make" in the project directory.

To run the code and see runtime:

For kmeans:

```
./omp_main –i ~/645/data/kmeans01.dat –n 3
```

For matrix_mul:

```
./matrix_mul –i ../matrix_mul_02.dat
```

Commit your code frequently to your local repository by doing:

```
git commit  –a   -m "description"
```

Step 3: Push your changes to the server

To share your optimizations with us, you will need to “push” the changes to the remote repository.

```
git push origin  master
```

http://www.kernel.org/pub/software/scm/git/docs/everyday.html

Step 4: Look at Jenkins

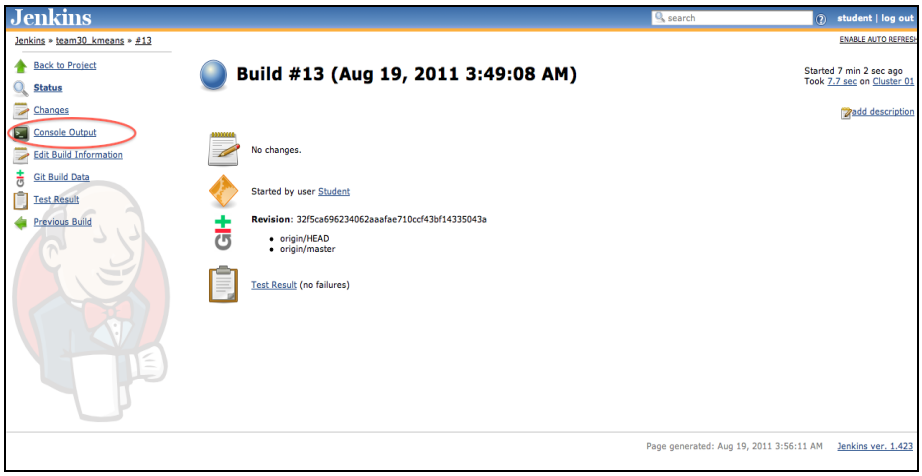
Visit http://fast645.info:8080

User: student
Password: 18645

Check the Jenkins website to see if your project ran through properly. Make sure you are looking at your team's project!

Then, click on the latest build:

To view the performance, use the Console Output link:



Step 5: Is the performance good enough?

If yes, look at the other application. If no, go back to step 2 and try another optimization. You are expected to try at least 3 optimizations per application.

Grading criteria

- 5% - Task 0 Answers
- 30% - Correctness - Correctness of the results (program output)
- 30% - Performance –
    - MatrixMultiply:
      For OpenMP version, achieving **at least 5X** speed up compared to **sequential version** (SUM of all the testcases – matrix_mul_02.dat)
    - K-means:
      For OpenMP version, achieving **at least 1.5x** speedup compared to current **OpenMP version** (SUM of all the tests)
- 25% - Write up – Clearly describing, for each performance optimization,
    - how the speed up works
    - what is the expected speed up
    - what is the observed speed up
    - an explanation of any difference between the expected and observed speed ups
- 10% - Code quality - Good coding practices and well commented code

Guidelines for the write up:

Minimum of one 8.5x11 page write-up for each optimization. The write up should include:

- Optimization goal:
    - Hardware resources being optimized toward?
      (cache? SIMD? multicore?)
    - What is the specification of the hardware you are optimizing for?
- Optimization process:
    - Data considerations
    - Parallelization considerations
- Optimization results:
    - Performance before optimization
    - Performance after optimization

Two teams with the fastest project in the class will be asked to do a 10min presentation each on what they tried.

We will look at the code of the slowest two implementations as a class. The class will discuss why their code is slow.