CS420: Parallel Programming - Fall 2017
MP1
Due Date: October 11, 5:00PM
Submission Method: SVN

September 29, 2017

This assignment requires you to optimize a dense matrix-matrix multiply routine and understand the performance beahvior. You will also learn to use PAPI to measure cache misses and to analyze the performance based on the profiling output.

You are required to implement this algorithm using the stubbed code provided in the given tarball `mp1.tar`. There are four parts to this assignment, as described in detail below. We have provided you with a *Makefile* and submission script *run.sub* for compiling and running on the campus cluster. You are to use PAPI for measuring different metrics regarding memory performance.

In most of the experiment, you will use `-O0` to disable any compiler optimization.

WARNING: Do not wait until the last minute to do all the experiments to avoid issues with the shared cluster.

Note on PAPI

Documentation for PAPI can be found here
`http://icl.cs.utk.edu/projects/papi/wiki/Main_Page`.

Part A: Setup the environment

We provided a basic implementation of matrix-matrix multiply in `basic_seq.c`. Upload the tarball `mp1.tar` to the cluster then load the `papi` and Intel compiler modules. Extract the source code and compile the given code. If done correctly, you will find the two binaries `cc_without_papi_1000` and `cc_with_papi_1000` as the results. Submit the run script and use its output and the source code to explain the default behavior of the code. Note that the output of the runscript in this MP will appear in `out-JOBID.txt`. Focus on the following and expand on them:

- What does the overall binary do?
- In which file(s) is the PAPI profiling implemented? What are the neccessary steps for profiling?
- What are the events that are being profiled by default?
- Is there any performance difference between the two compiled binaries?

Part B: Simple improvement

The basic implementation is inefficient due to unnecessary accesses to the C array. Using a temporary variable, modify the basic algorithm to reduce the number of memory accesses. Do this in the `basic_seq_b.c` file. Make sure the result is correct.

Recompile and run the basic and the better version. Explain the behavior, focus on what is being improved in your version, and whether that is reflected in the timing and profiling information.

Part C: Cache-aware tiling implementation

Use your better version implemented in Part B as a baseline and implement a tiled version of the matrix-matrix multiply algorithm in `cache_aware_seq.c`. Be sure to tile every dimension. Choose a block size that runs best according to your knowledge of the system and/or by running experiments. Make sure the result is correct.

Recompile and compare your implementation in part B and the new implementation using the following inputs and discuss the results:

- N=100, M=100
- N=100, M=1000
- N=1000, M=100

- N=1000, M=1000
- N=2000, M=2000
- N=3000, M=3000

You can set these sizes by editing the Makefile and remember to change the runscript to run the corresponding binary (with correct suffix).

Choose the largest input size above and modify the PAPI profiling to measure the L3 cache misses rate for each implementation. Discuss the steps that you took to come up with that answer in the report.

Part D: Compiler optimization

Modify the `Makefile` to use the `-O3` optimization level. Recompile and rerun the experiment with the given inputs in Part C.

Discuss what the compiler does under that optimization. Do you still see improvement with your code in Part B and Part C compared to the basic implementation?

Submission instructions

- Everything should be submitted under the `mp1` directory in your subversion folder.
- Submit the two files that you should have modified: `basic_seq_b.c`, `cache_aware_seq.c`.
- Submit a report `NETID_mp1` with your writeup for each part. You may submit either a PDF (.pdf) or plain text (.txt) file at your convenience.