Using the Provided Benchmarks

Dylan Rudolph - September 9, 2013

1 Summary

The goal of the files in this set is do automate the benchmark-gathering process for any architecture which supports Python 2, has a fairly recent version of GCC, and runs a GNU/Linux or popular Unix-like operating system. Provided that the system is set up properly, only one command line call will be necessary to compile, run, and save the results of, all of the benchmarks.

2 Dependencies

The following are necessary for proper operation:

- A recent version of GCC.
- A version of the OpenMP standard, which may come bundled with GCC.
- The FFTW library and development headers (Version 3), along with the FFTW support for OpenMP.
- Some version of Python 2, where 2.7 is preferred.
- An operating system which supports GNU's Make functionality.

3 Directory Map

The main directory contains (1) the Python file `dylanbench.py` which does most of the automation process, (2) a set of Python scripts which call `dylanbench.py` to perform each benchmark, and (3) a shell script `do_everything.sh` which calls each of the individual Python scripts.

The folder `Benchmarks` contains all of the relevant benchmarks. Inside of this folder there is a makefile which builds all of the benchmarks. Inside of each of the individual benchmark folders there is also an individual makefile, along with the source code. If you do not want to compile all of the benchmarks, the individual makefiles can be called.

The folder `Savefiles` contains the `.bmark` files and a subfolder called `CSVs` which contains the output `.csv` files.

4 Usage

If all is well on the system, you need only call `sh do_everything.sh` .

Alternatively, individual benchmarks can be compiled by calling `make` in the folder containing the benchmark. Also, individual benchmarks can be run by calling `python nameofbenchmark.py`.

4.1 Notes

- Make sure to change the number of threads for each of the OpenMP benchmarks to match it for the platform.
- Ensure that the compiler optimization flags do not optimize out blocks of code for the system of interest.

5 Benchmark Terminology

Any benchmark appended with "naive" is a benchmark which has no compiler optimizations and is implemented in the most intuitive way. Any benchmark appended with "OneThread" is one which has been optimized (by compiler or algorithm) to make the best use of a single core on a host system. Finally, any benchmark labeled "OMP" is one which has been optimized and made parallel with OpenMP.

6 Save Files

The `.bmark` format is specified by the `dylanbench.py` file, and contains most any important information about a benchmarking run. Some of the data from the `.bmark` format can be exported to a `.csv` file. This is done automatically by the accompanying Python scripts. Note that the `.csv` files contain less information than the `.bmark` files, so the `.bmark` should be considered the primary storage. Exporting from `.bmark` to `.csv` can be done by calling the relevant function from `dylanbench.py`.