Final Project Part 3: Vectors

It will be somewhat helpful to complete Parts 1 and 2 before starting this part, but it's not critical.

This portion of the project is about vectorization (also known as SIMD). We will be using OpenMP's and GCC's facilities for SIMD programming.

Turning on SIMD

GCC includes a vectorizing pass and it's turned on by default with `-O3`. You can make it maximally effective by also passing `-march=native` to `gcc` (by adding it to `OPTIMIZE` in `config.env`) will compile code optimized for the machine you are currently running on.
Note: Passing `-march=native` on the DSMLP will result in different results than passing it on the autograder. This means that if you want to look at the assembly, you should build it on the autograder. For instance: `runlab --run-git-remotely --make build/model.s`.
The `build/` is important!

Tuning SIMD

GCC provide several command line options to control the vectorizer. Search the gcc man page for `-ftree-vectorize` more information.
Fiddling with these knobs is not usually a good use of your time relative to other ways you could improve performance.

Turning off SIMD

SIMD is on by default. If you want to turn it off, you can with `OPTIMIZE=-O3 -fno-tree-vectorize -fno-tree-loop-vectorize` in `config.env`.

Telling the Vectorizer to Vectorize a Loop

In principle, the compiler should vectorize all the loops it can vectorize. However, in practice, it seems to need some encouragement. The easiest way to do this is with OpenMP. Just put

```
#pragma omp parallel simd
```

before the loop you would like to vectorize.

Read the Source Code

We'll be studying the code in `microbench.cpp`. It contains many implementations of a very simple loop that performs a not-very-useful computation of updating element of an array 3 times.
The implementations are:

- `serial` – Basic serial version.
- `serial_improved` – an "improved" serial version.
- `openmp_threads` – parallized with OpenMP
- `openmp_simd` – vectorized with OpenMP
- `openmp_threads_simd` – parallelized and vectorized with OpenMP
- `gcc_simd` – hand-vectorized with gcc vector types.
- `openmp_threads_gcc_simd` – OpenMP threads + hand vectorization with gcc vector types

We will just be using a few of them.

The `main()` function contains some code to select among the implementation and control thread count, etc.

Seeing What the Vectorizer is Doing

The `gcc` vectorizer will tell you what it does if you pass `-fopt-info-vec`.
Here's part of the result for compiling `microbench.cpp` (`runlab --run-by-proxy --make build/microbench.o`) with `MICROBENCH_OPTIMIZE=-O3 -march=native -fopt-info-vec-optimized` (`MICROBENCH_OPTIMIZE` sets the optimization flags for compiling `microbench.cpp`):

```
g++-8 -c -Wall -Werror -g  -O3 -march=native -fopt-info-vec-optimized                    \
   -I/root/steve/cse141pp-archlab/pcm -pthread                                          \
   -fopenmp -I/root/steve/cse141pp-archlab/libarchlab -I/root/steve/cse141pp-archlab    \
   -I/usr/local/include                                                                 \
   -I/googletest/googletest/include -I/root/steve/CSE141pp-SimpleCNN -Ibuild/           \
   -I/home/jovyan/work/moneta/ -std=gnu++11 build/microbench.cpp -o build/microbench.o
build/microbench.cpp:49:34: note: loop vectorized
build/microbench.cpp:49:34: note: loop vectorized
...
```

It lists the file and line numbers for the loops that were successfully vectorized.

Passing `-fopt-info-vec-missed` shows you what it didn't vectorize and why. The output can be cryptic and long (~7000 lines for `build/microbench.o`) to say the least. Here's a sample:

```
build/microbench.cpp:115:21: note: not vectorized: not enough data-refs in basic block.
build/microbench.cpp:115:21: note: not vectorized: not enough data-refs in basic block.
build/microbench.cpp:115:21: note: not vectorized: not enough data-refs in basic block.
build/microbench.cpp:115:21: note: not vectorized: not enough data-refs in basic block.
build/microbench.cpp:114:9: note: not consecutive access j_8 = *.omp_data_i_7(D).j;
build/microbench.cpp:114:9: note: not consecutive access array_9 = *.omp_data_i_7(D).array;
build/microbench.cpp:114:9: note: not vectorized: no grouped stores in basic block.
build/microbench.cpp:126:7: note: not vectorized: no vectype for stmt: _44 = *v_27;
build/microbench.cpp:126:7: note: not vectorized: no vectype for stmt: *v_27 = _45;
build/microbench.cpp:126:7: note: not vectorized: no grouped stores in basic block.
build/microbench.cpp:126:7: note: not vectorized: not enough data-refs in basic block.
```

The format of these lines is `<filename>:<line>:<column>...`.
If you want more information, `-fopt-info-vec-all` will oblige: it generates 32K lines of output for `build/microbench.o`.

P1 (1pts) Add `-O3 -fopt-info-vec-optimized` to `MICROBENCH_OPTIMIZE` in `config.env` and then `runlab --run-git-remotely -- make build/microbench.o` to generate the optimization report. It'll show up in the terminal and `STDOUT.txt`. Paste in the lines that contain the string `microbench.cpp` below (there will be less than 10 of them).

```
The vectorization report.
```

P2 (1pts) Which functions contain loops that were vectorized?

```
Functions with vectorized loops
(use line numbers in the report
to get the function names in the cpp file)
```

Set `MICROBENCH_CMD_LINE_ARGS=--stat-set PE.cfg --impl openmp_simd serial` (leave `MICROBENCH_OPTIMIZE` as it is) and then `runlab --run-by-proxy -- make microbench.csv` (or commit and use `--run-git-remotely`). Use the resulting `microbench.csv` to answer the following question.

P3 (3pt) Compute the impact of SIMD on the following terms of the performance equation using the data for `openmp_simd` and `serial`. For each term, compute `(value for serial)/(value for openmp_simd)`.

```
IC:

CT:

CPI:

ET:
```

P4 (2pt) If Intel improved their processors so that the CPI for vector instructions matched that of normal instructions, how much speedup would `openmp_simd` achieve relative to `serial`?

```
Speedup:
```

P5 (10pt) Here's 10 free points because we couldn't get vectorization to do anything useful on our code base. Put whatever you'd like below.

Go forth and optimize!

There are bunch of loops in the code base just waiting to be vectorized. I would prioritize threading and tiling over vectorization.

Check `README.md` for performance targets for the end of this part of the lab. The expected speedups for the week 3 are attainable without vectorization.

YOU ARE DONE! PLEASE ENJOY YOUR SUMMER!
