Homework 3 assigned. Due November 10, 2017 11:59pm MT.

Preliminaries:

1) Submission:

The hard deadline is Friday, November 10 2017, 11:59pm MT. No extension will be provided.

Send me by email to pouchet@colostate.edu, subject "CS560: assignment 3 -- your name" an archive (.zip or .tar.gz) containing all the files asked below. Make sure they all have unique name, e.g., "gemm.question1-a.c" and "gemm.question2.c" so that I can easily see which file corresponds to which question. This archive shall also contain a PDF document with your answers to the various questions below, as well as the result/output of the following two commands executed on the machine you used to run all program variants:

$> cat /proc/cpuinfo
$> gcc --version
This will inform me about the processor you used to do the experiments, and which GCC version was used, as both influence a lot the final performance you could obtain.

2) Grading:

Exercise 1 is worth 20%, Exercises 2-3 40% each. An extra 20% is possible via Exercise 5 (optional, extra credit). It will not be graded/considered unless Exercises 1-3 are completed.

3) Advices:

- You will not be graded based on the performance of the program you obtained, but based on how you justified your choices. That is, there is no expectation of "minimal performance improvement" your solution must achieve to get 100% to a question.

- I advise you spend no more than 5 hours reading and studying the paper for exercise 1, but no less than a couple hours at least.

- You may face issues when building the compiler. Do not spend excessive time figuring out the problem if you are stuck: this is not the point of this assignment. Instead, email me directly so that I can assist you. I will only provide support for recent Linux machines (no Windows, no Mac: use instead a lab machine as needed).

- This assignment illustrates how to build an auto-tuning system to find out the best performing transformation for a (polyhedral) program, the typical state-of-practice when optimizing codes. While I provide you which compiler flags to use, you are asked to find by yourself a relevant set of tile sizes to test, something which has not been covered in class. This is your duty to look around on the internet, possibly quickly reading some research papers, to figure out what could be interesting tile sizes to explore. You will NOT receive help on Piazza about this.

Exercise 1:

Read the paper:

A Practical Automatic Polyhedral Parallelizer and Locality Optimizer
Uday Bondhugula, A. Hartono, J. Ramanujan, P. Sadayappan.
ACM SIGPLAN Programming Languages Design and Implementation (PLDI), Jun 2008, Tucson, Arizona.

https://www.researchgate.net/profile/J_Ramanujam/publication/228732823_PLuTo_A_practical_and_fully_automatic_polyhedral_program_optimization_system/links/0912f50b7b3c59b6c0000000.pdf

Question 1:

Given Algorithm 1 in the paper, that computes a schedule which maximizes tilability and minizes the reuse distance. When giving as input to this algorithm a polyhedral representation of a C program, will it always succeed in producing a schedule for each statement? Justify your answer (1-2 paragraphs).

Question 2:

Looking at Eq. (7), suppose that for the first schedule row the solution to (7) is such that \vec u = \vec 0 ("the 'u' vector is equal to 0).

a) Does it necessarily imply that the first schedule dimension models a loop?

b) Does it necessarily imply that the first schedule dimension does NOT strongly satisfy any dependence? As a reminder, strong dependence satisfaction means that for all pairs (x_R,x_S) in the dependence polyhedron D_{R,S}, we have \Theta_R(x_R) < \Theta_S(x_S).

For each, justify your answer (1-3 sentences).

Exercise 2:

For this exercise, you will use the PoCC compiler. The first stage is to download and build it. It is restricted to Linux machines, and as the point of this assignment is not for you to spend time building a tool if you encounter any difficulty please email me and we'll work through your setup together.

Preliminaries:

1) Download.
Visit http://pocc.sf.net and download the file:
pocc-1.4-selfcontained.tar.gz

2) Build. Execute the following commands:

```
$> tar xzf pocc-1.4-selfcontained.tar.gz
$> cd pocc-1.4
$> ./install.sh
```

[wait 10-30 minutes for the build to complete, do not interrupt the process!]

```
$> ./bin/pocc --help
```
If this last command prints the list of flags, then your build is successful and you can proceed. If there was an error/interruption, you shall trash the pocc-1.4 directory, and restart the process from scratch.

3) Download the PolyBench/C benchmarking suite.
Visit http://polybench.sf.net and download the file:
polybench-c-4.2.1-beta.tar.gz

4) Do a sanity check for PolyBench and PoCC:
First, we execute the base program for gemm, and measure execution time:

```
$> tar xzf polybench-c-4.2.1-beta.tar.gz
$> gcc -O3 polybench-c-4.2.1-beta/linear-algebra/blas/gemm.c
polybench-c-4.2.1-beta/linear-algebra/utilities/polybench.c -Ipolybench-c-4.2.1-beta/linear-algebra/utilities
-DPOLYBENCH_TIME -o gemm.orig
$> ./gemm.orig
```

Then, we repeat the process but using a code transformed by PoCC:

```
$> ./bin/pocc --verbose polybench-c-4.2.1-beta/linear-algebra/blas/gemm.c
$> gcc -O3 polybench-c-4.2.1-beta/linear-algebra/blas/gemm.pocc.c
polybench-c-4.2.1-beta/linear-algebra/utilities/polybench.c -Ipolybench-c-4.2.1-beta/linear-algebra/utilities
-DPOLYBENCH_TIME -o gemm.transfo
$> ./gemm.transfo
```

As we have not done any actual change of schedule, you should observe roughly the same performance for ./gemm.transfo and ./gemm.orig.

We focus on polybench-c-4.2.1-beta/linear-algebra/blas/gemm.c for the following.

Question 1:

a) Generate the transformed C file using PoCC that implements the
schedule found by pluto to maximize tilability and fusion. The command line is:

```
$> ./bin/pocc --verbose --pluto --pluto-fuse maxfuse <your file>
```

b) Execute this transformed C code, and compare its performance with
the original code: is it performing faster?
c) In the schedule found by pluto, which schedule dimension is parallel?

d) Generate and execute another transformed file, now using the flags:

```
$> ./bin/pocc --verbose --pluto --pluto-fuse smartfuse <your file>
```

Compare its performance with the code generated by a) above: is it now performing faster?

Question 2:

a) Generate the transformed C file using PoCC that implements the
schedule found by pluto to maximize tilability and fusion, and tiling using default tile sizes. The command line is:

```
$> ./bin/pocc --verbose --pluto --pluto-fuse maxfuse --pluto-tile <your file>
```

b) Execute this transformed C code, and compare its performance with the code generated for Question 1.d: is it now performing faster?

Question 3:

The tile size to use is specified in a 'tile.sizes' file that MUST be located in the current working directory. This file must have AT LEAST as many numbers as there are tilable dimensions. That is, for gemm, a file like this:

$> cat tile.sizes
32 32

Will be incorrect: we need at least 3 sizes since it is a 3D-tilable code. The approriate file is

$> cat tile.sizes
32 32 32

And the following file is also valid, simply additional numbers will not be read:

$> cat tile.sizes
32 32 32 32 32 32 32 32 32 32 32

a) We will apply a tiling of 50x80x64. Write a tile.sizes file to obtain:

$> cat tile.sizes
50 80 64

Then re-generate the tiled code with:

$> ./bin/pocc --verbose --pluto --pluto-fuse maxfuse --pluto-tile <your file>

Execute this transformed C code, and compare its performance with the code generated for Question 2.b: is it now performing faster?

b) Repeat the process with tile size 1x80x128, is this code faster than with tile size 50x80x64?

Question 4:

The objective is to design a small script that allows you to easily explore multiple tile sizes. Write a script/program in any language of your choice which automates Question 3 above, that is which for each tile size in a list you manually provide, will (a) generate the tile.sizes file; (b) call PoCC to generate the transformed code; (c) run the code to report its execution time; (d) output the program with the lowest execution time.

To test, you can simply use "50 80 64" and "1 80 128" as the two tile sizes to explore, but make sure your script is ready to operate on a much larger tile size list of a few tens of entries at least.

Question 5:

Use the script developed for Question 4 to find the best performing tile size, by exploring 20 different tile sizes in total. Include your script in your submission, as well as the list of 20 tile sizes you chose to explore if it is not inside the script already. Justify (1-2 paragraphs) why you chose these 20 tile sizes, out of the N^3 possible tile sizes for gemm.

Exercise 3:

Optimize the program:
polybench-c-4.2.1-beta/stencils/heat-3d/heat-3d.c
that computes how heat propagates in a 3D element with PoCC.

Find the best performing transformed program, under the following restrictions:
- You are not allowed to edit heat-3d.c or any .c file generated by PoCC
- You can explore only 20 different tile sizes
- You can only use combinations/subsets of the following flags from PoCC:

```
--pluto
--pluto-tile
--pluto-parallel
--pragmatizer
--pluto-fuse maxfuse
--pluto-fuse smartfuse
--pluto-fuse nofuse
```

Note that to exploit OpenMP parallelism in the transformed code (as generated when using the --pragmatizer flag, and when coarse-grain parallelism exists), you shall update the compilation line from, e.g.,

```
$> gcc -O3 polybench-c-4.2.1-beta/linear-algebra/blas/gemm.c
polybench-c-4.2.1-beta/linear-algebra/utilities/polybench.c -Ipolybench-c-4.2.1-beta/linear-algebra/utilities
-DPOLYBENCH_TIME -o gemm.orig
```

to:

```
$> gcc -O3 -fopenmp polybench-c-4.2.1-beta/linear-algebra/blas/gemm.c
polybench-c-4.2.1-beta/linear-algebra/utilities/polybench.c -Ipolybench-c-4.2.1-beta/linear-algebra/utilities
-DPOLYBENCH_TIME -o gemm.par.orig
```

Provide:
1- your script performing the exploration
2- the set of 20 tile sizes you used, justifying your choice (1-2 paragraphs)
3- the measured execution time of all transformations explored by your script
4- the final best-performing file you obtained

Exercise 4 (optional, extra credits):

Optimize the program:
polybench-c-4.2.1-beta/linear-algebra/kernels/2mm.c
that computes a sequence of 2 matrix multiplications With PoCC.

Find the best performing transformed program, under the following restrictions:
- You are not allowed to edit 2mm.c or any .c file generated by PoCC
- You can explore only 20 different tile sizes
- You can only use combinations/subsets of the following flags from PoCC:

```
--pluto
--pluto-tile
--pluto-parallel
```

--pragmatizer
--pluto-fuse maxfuse
--pluto-fuse smartfuse
--pluto-fuse nofuse
--letsee --letsee-dry-run

Using the flags "--letsee --letsee-dry-run" along with pluto flags will generate one C file per fusion/distribution structure possible for the program. This is described in:

Louis-NoÃ«l Pouchet, Uday Bondhugula, CÃ©dric Bastoul, Albert Cohen, J. Ramanujam and P. Sadayappan.
Combined Iterative and Model-driven Optimization in an Automatic Parallelization Framework.
In IEEE Conference on Supercomputing (SC'2010). New Orleans, LA, November 2010.
[bibtex-entry] [pdf] [slides]

Provide:
1- your script performing the exploration
2- the set of 20 tile sizes you used, justifying your choice (1-2 paragraphs)
3- the measured execution time of all transformations explored by your script
4- the final best-performing file you obtained

Thanks,

++