CSE211: Compiler Design

Homework 3: Parallelism
Assigned: Nov. 12, 2020
Due: Dec. 3, 2020

- Read through instructions fully before beginning assignment; the testing scripts described later can help you while developing!
- These assignments are done with Python 3; you'll need to explicitly call `python3` on the server unless you switch it in your environment. The default `python` command defaults to version 2.7.
- The first two parts produce C++ files which are then compiled by `clang++`. Please note I have built a debug version of clang, so compiling is probably slower than you are used to. Please don't be alarmed.
- This homework requires timing experiments! Please only run experiments on one thread at a time. Before you start an experimental campaign, please run `top`. If more than 4 people are running experiments (look at how much memory and CPU are being used), then wait for an hour or so. If resources become a major constraint, we can set up a reservation system.

1 Loop Unrolling Independent Iterations for ILP

Here we will consider `for` loops with independent iterations. However, each iteration will contain a chain of *dependent* instruction. This chain is intended to impede the processor's ability to exploit MLP, either through pipelining or superscalar components. The length of dependent instruction chains is a parameter of the program. Your assignment is to unroll the loops, first doing each iteration sequentially, and then interleaving the instructions from different iterations. Interleaving instructions should allow the processor to exploit more ILP and you should be able to see this in your timing experiments. You will measure the execution time of various dependent chain lengths and unrolling factors.

1.1 Preliminaries

1. Read through Chapter 9 of the dragon book (you can skim, there is a lot of implementation details you do not need to know in detail) to refresh yourself on ILP. You can also review the Nov. 10 lecture which should contain all required background.
2. Find the assignment skeleton at `/home/tsorensen/public/homework3/part1`. Make a copy over to some homework1/part3 directory within your home directory. e.g. run the following:

```
cp -r /home/tsorensen/public/homework3/part1/* ./
```

3. The coding aspect of this assignment is constrained largely to `skeleton.py`. Read through this code to understand the structure. The python code is writing a C++ file that is then compiled with `clang++` and executed. The C++ file will time your implementation loop against a reference.

4. you should be able to run the skeleton as it is initially. You will fail the assertion check when running the C++ program, as the homework loop is not implemented. Examine `homework.cpp` to see the structure.

5. You will need `clang++` in your path for this part of the assignment. Please add it to your path in your `/.bashrc` file by adding:

   ```
   export PATH="/home/tsorensen/software/llvm-project/build/bin/:$PATH"
   ```

1.2 Technical work

You will implement a function that unrolls the reference loop and interleaves the instructions to exploit ILP. Your function starts around line 68. Please read the long comment above the function for the specification and description of the parameters. There is also an example of how an interleaved and unrolled loop should look.

You can assume that the size is always a power of 2, and so is the unroll factor. That is, you do not need to implement ”clean” up iterations.

For the evaluation, you are not allowed to change the `clang++` compile line, the reference loop, or the main string. The timings you use for your evaluation need to come from a program that passes all of the assertions.

1.3 Evaluation

Run your program with all combinations of {1, 2, 4, 8, 16, 32, 64, 128} for both chain length and unrolling factor. Collect data for both interleaved and non-interleaved unrollings. My suggestion is to modify `skeleton.py` to accept command line arguments for the chain length, unrolling factor and interleaved value. Then make an additional script that runs skeleton with various parameters and parses the output to collect the speedup data.

Please report your timing information as a heatmap. The X axis should be the unrolling factor and the Y axis should be the dependency chain length. There are many tools that can produce heatmaps, for example:

- gnuplot: `http://gnuplot.sourceforge.net/demo_5.2/heatmaps.html`
- Excel: `https://www.excel-easy.com/examples/heat-map.html`

Please include numbers in your heatmap, similar to the Excel example.

Prepare two heatmaps: one for interleaved unrollings and one for non-interleaved unrollings.

1.4 Submission

Please create a directory in your home directory called: `submissions`. Within that directory, create a directory called `homework3_part1`. Move your `skeleton.py` into that directory along with a PDF containing both of your heatmaps.

2 Unrolling Reduction Loops for ILP

Part 2 is identical to part 1, except we are targeting a different type of `for` loop. Here we are targeting reduction loops, where each iteration depends on the previous one. Recall in class, we showed that these loops can be unrolled to exploit ILP. You should apply a ”chunking” unroll style, i.e. how we described in lecture. That is, the input array should be divided into N equal sized chunks (where N is the unrolling factor). Each loop iteration can then execute N reduction commands. At the end of the function, there needs to be a loop adding up the totals for each N.

There is only one parameter in this part, the unroll factor. Because of this, your timing results can be displayed as simple line graphs, where time is the Y axis, and the unroll factor is the X axis. You only have to go up to an unrolling factor of size 32 in this part (you will see why).

Again you can assume the size and unroll factor is always a power of 2.

There is an example in the code comments. All other aspects of this part can are the same as part 1. The skeleton can be found at:

`/home/tsorensen/public/homework3/part2`

2.1 Submission

Please create a directory in your home directory called: `submissions`. Within that directory, create a directory called `homework3_part2`. Move your `skeleton.py` into that directory along with a PDF containing your timing experiments.

3 Detecting SPMD Parallel Loops

In this part, you will be given some nested `for` loops, and two index calculations for a memory access (a read and a write). Your job is to determine if it is safe to make the outer-most loop parallel. That is, you will need to determine if the index calculations could conflict with two threads executing the outer-most loop. You will build up constraints that model a *reader thread* and a *writer thread*. You will use the Z3 constraint solver to check if the two threads can conflict.

3.1 Preliminaries

1. Review the lectures on Nov 12 as they go over the theory for this part of the homework.

2. Go through the Python Z3 tutorial at:

   https://ericpony.github.io/z3py-tutorial/guide-examples.htm

   You only need to go up to the ”Functions” section (immediately after the ”Machine Arithmetic” section).

   Find the assignment skeleton at `/home/tsorensen/public/homework3/part3`. Make a copy over to some homework1/part3 directory within your home directory. e.g. run the following:

   ```
   cp -r /home/tsorensen/public/homework3/part3/* ./
   ```

3. Your assignment is constrained to `skeleton.py`. Read through this code to understand the structure. I have done the work to parse the python AST for you. You will need to implement the constraint solving in `check_parallel_safety` around line 172. Please read

the specification and comments (especially the description at the top of the file) carefully. You do not need to fully understand the ast parsing, and those functions are labeled.

3.2 Technical Work

Your job is to implement constraints to determine if two threads would conflict if the outer-most loop is made parallel. The programs have an extremely limited form. They are an arbitrary nest of `for` loops followed by a read index calculation and a write index calculation. I have parsed the AST and provided you with a list of `ForLoops` and read/write index strings.

Use Z3 to create two variables per `for` loop: one for the reader thread, and one for the writer thread. Add the constraints to the solver such that these variables respect their loop bounds. The outer-most loop will need an additional constraint as the reader and writer thread cannot have the same value for the loop variable.

The read/write index strings will be expressions consisting only of numbers, loop variables, and operators (+ or *). I suggest using string replace functions to substitute the Z3 loop variables into these strings. You can then set a constraint where these two strings are equal to each other. This string can be evaluated using `eval`. This is not the cleanest solution, but I did not want to subject you to the actual Python AST.

At this point, ask Z3 to solve the equation. If the equation is satisfiable, it means there is some iteration from the writer thread that conflicts with some iteration of the reader thread, and thus it is not safe to make the outer-most loop parallel.

3.3 Evaluation

Much like the previous assignments, I have provided a `tester.py` that you can run. There are 8 test cases in the `test_cases` directory. You can run your skeleton directly given one of these files.

3.4 Submission

Please create a directory in your home directory called: `submissions`. Within that directory, create a directory called `homework3_part3`. Move your `skeleton.py` into that directory. Do not make this directory readable by anybody else. I have admin privileges that will allow me to copy from there.