Automatic parallelization in Graphite

Li Feng

April 3, 2009

1 Synopsis

With the integration of Graphite to GCC4.4, a strong loop nest analysis and transformation engine was introduced. But now it does only non-parallel loop generation. My work is to detect synchronization free parallel loops and generate parallel code for them, which will mainly enable programs to run faster.

2 The Project

In GCC there already exists an auto-parallelization pass, which is base on the lambda framework originally developed by Sebastian. Since Lambda framework is limited to some cases, Graphite was developed to handle the loops that lambda was not able to handle . e.g. The following code can't be handled by autopar, which yields a scev_not_known dependency.

```
int Z[100][100];

int main(void)
{
  int i,j;
  for (i = 0; i <= 10; i++)
    for (j = 0; j <=10; j++)
```

```
        Z[i][j] = Z[j+10][i+11];

    return 0;
}
```

I made some experimental work on "What kind of loops can autopar handle and what it can't" ,you can find it here: http://gcc.gnu.org/wiki/AutoparRelated.

Graphite can analyze every loop, condition that can be analyzed by polyhedral analysis. Which means it can handle any comparision between linear expressions in it's conditions, while the autopar framework (with Banerjee analyzer and Omega analyzer) can't really handle all of this situation. Another limitation of autopar is that it is limited to some special loop nests (perfectly nested), while Graphite can support it well as long as the side effects of a statement are known. And even better, when the loop transformation is scheduled by Graphite, we can easily extend it to handle lots of loops.

So under Graphite, we could detect synchronization free parallel loops and generate parallel code for them. During Google Summer of Code, I want to solve the two issues.

The first issue will be parallel loop detection. This means that Graphite will recognize simple parallel loops using SCoP detection and data dependency analysis. SCoP detection is well supported by Graphite and Konrad Trifunic is now working on data dependency analysis based on polyhedral model.

Firstly I will write test cases for different kind of loops that can be parallelized under Graphite. This work will be done by some discussions about data dependency analysis under polyhedral model with Graphite developers.

The place for parallel loop detection will be after CLOOG generate the new loops, either CLAST(cloog AST after call cloog on polyhedra) or after clast_to_gimple. At this time as we have the polyhedral information (poly_bb_p) still available during code generation, we can try to update the dependency information using the restrictions CLOOG added and the

polyhedral dependency analysis to check if there is any dependency in the generated loops.

So the basic step of parallel loop detection will be:

1. Get the poly_bb_p after CLOOG.
2. Check that if there are dependencies.
3. If dependency doesn't exist, mark the loop as parallel. We may add annotations of more detailed information here. e.g. shared/private objects.

The second issue will be the parallel code generation. I'll try to connect Graphite to autopar's code generation part. I will mostly focus on the Graphite part. There are two ways of connecting to autopar that is under discussion. The first will be the first try.

1. By annotating loops: In the graphite pass we annotate gimple loops as parallel(in loop data structure). Later the autopar pass is scheduled, which bypasses its analysis if it sees a loop annotated parallel. This allows to schedule scalar optimizations in between graphite and autopar. Although we might lose polyhedral information which seems not that necessary in code generation part, we will get the scalar optimization(like PRE) which we definitely need.
2. Directly call autopar's code generation part: The autopar will be called after clast_to_gimple, where we still have the access to the polyhedral information which might give autopar additional information. But this seems not that necessary since we don't really need that information in code generation part.

3 Success Criteria

1. Graphite can recognize and mark loops that can be parallel.
2. Graphite will generate parallelized code for them.
3. Pass test cases for Graphite's auto-parallelization.

4 Road-map

1. Since data dependency analysis is not available, I will firstly try to connect Graphite to autopar, where I will focus mostly on the Graphite/polyhedral part.(Mid June)

    - Introduce a flag -fgraphite-force-parallel that forces auto-parallelization for all loops.
    - Try to generate parallel loop nests (mark all the loops as parallel if the flag is set).
    - Then the autopar pass will bypass it's analysis if it sees the loop marked as parallel and generate the parallel code for it.

2. Write test cases for the loops that can be parallelized. This will take a few discussions with Graphite developers to see which kind of loops we will detect and can be auto-parallelized. This part will take some time that there will be a detailed plan on what kinds of loops we should detect.(last week of June)

3. Work with Razya to fix bugs in autopar's code generation part since it might handle incorrectly for some test cases.(First week of July)

4. Write code for parallel code detection. This part of job will based on SCoP detection and data dependency, and at this time, data dependency analysis should have been done. This part of work will take some time. (First week of August)

5. Code cleaning and write documents.(Second week of August)

5 Profit for GCC

- When the auto-parallelization has been done in Graphite, developer can mostly take their effort to loop transformations. Graphite will be in charge of optimizations(generate parallelism) and the autopar code in Graphite will just detect and generate code for them.