15745 Optimizing Compiler Project Milestone: Implementation of Decoupled Software Pipelining (DSWP) in the LLVM Compiler Infrastructure

1 Group Information

Fuyao Zhao: `fuyaoz@cs.cmu.edu`
Mark Hahnenberg: `mhahnenb@andrew.cmu.edu`

2 Major Changes

We decided to use POSIX threads (pthreads) rather than Cilk as our parallel runtime system.

3 What You Have Accomplished So Far

This project contains several stages. So far, we have finished the following stages:

- Build Program Dependency Graph (PDG)–it was trivial to get register dependencies, we used the alias analysis pass to get memory dependencies, and we used the post-dominator tree to build control dependency [2])
- Find Strongly Connected Components in the PDG [1].
- Coalesce the the SCCs to get a DAG
- Thread Partitioning using the DAG and the heuristic algorithm in [3])
- Split the original loop into several loops where each loop is a separate function (haven't extracted the live-in and live-out variables).

3.1 Meet Your Milestone

We actually did not fully keep to our original schedule as the implementation ended up taking more time than we thought previously. This week we plan to get a system that could basically run, but this could prove to be a bit difficult. Since the remaining parts are code generation, which includes inserting inter-thread communication, we need to design this part carefully.

3.2 Surprises

There are many papers describe the DSWP and its variations. However, we found that the dependency analysis for DSWP (the basic one) is not exactly the same. In the original paper [3], they tried to conceptually peel the first iteration of the loop, and do standard control dependence analysis, and then combine the result back. However, in later paper [4], they didn't mention that, instead they just used the PDG [2]. We think using the PDG is sufficient to catch all the dependences, so we use this for the implementation.

3.3 Revised Schedule

- Rest of Week 4 : Finish code splitting and guarantee the correctness for all previous steps. Come out the initial plan for code gen.
- Week 5: Finish the implementation.
- Week 6: Refine the system, evaluate the our system on different benchmarks, finish the final report.

4 Resources

We will use SPEC2006 as benchmarks. Now we have a AMD 4 cores machine, but it might be better if we could run the system on a machine which can support more threads. We also need more accurate latency information for our all instructions on the specific experimental machine (now we use a table whose source is unknow).

References

[1] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[2] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, 9:319–349, July 1987.

[3] Guilherme Ottoni, Ram Rangan, Adam Stoler, and David I. August. Automatic thread extraction with decoupled software pipelining. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, pages 105–118, Washington, DC, USA, 2005. IEEE Computer Society.

[4] Easwaran Raman, Guilherme Ottoni, Arun Raman, Matthew J. Bridges, and David I. August. Parallel-stage decoupled software pipelining. In *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*, CGO '08, pages 114–123, New York, NY, USA, 2008. ACM.