Enhancing the Automatic Generation of Fused Linear Algebra Kernels
Erik Silkensen, Ian Karlin, Geoff Beter, Elizabeth Jessup, Jeremy Siek. University of Colorado at Boulder

Abstract

The performance of scientific applications is often limited by the cost of memory access inside linear algebra kernels. We developed a compiler that takes such kernels for memory efficiency, showing significant speedups relative to vendor- and BLAS libraries. Our compiler accepts annotated MATLAB code and outputs C code with varying levels of loop fusion. In this poster, we present an analysis of memory bound matrix vector multiplications that suggested ways to improve the compiler. In particular, we analyze and add the loop fusion optimization to the compiler, which we call "fully fused". We also present hardware performance counter data, which shows we need to consider register allocation in the memory model, and help identify when other optimizations such as cache blocking and array interleaving become profitable.

1. Generating Fused Linear Algebra Kernels

- Input annotated MATLAB code
- Use exhaustive search to enumerate opportunities for optimization by loop fusion
- Select most efficient routine and output C code

2. Searching for the Best Routine

- Empirically testing each routine is accurate but quickly becomes expensive as the number of optimization opportunities increases with kernel complexity.
- Analytic modeling estimates the performance of each routine by predicting the amount of data accessed from each memory structure in a machine and then converting it into execution time.
- The compiler uses a hybrid analytic/emperical approach to reduce the number of routines that must be empirically tested and then to select the best one.

3. Testing the Compiler

- Using sequences of matrix-vector multiply
  - nevs = number of matrix-vector multiplies done (e.g., nevs = 2 in previous pseudo-code)
- Looking for opportunities to improve the analytic model
  - 167,894 possible routines when nevs = 8
  - Does the model predict which is best?
- Tests show that it is most profitable to fuse all loops until nevs = 4, when it is better to fuse only outer loops
  - This drop in performance for fully fused isn’t predicted by the analytic model
  - How can we explain the crossover?

4. Ruling out Reasons for the Crossover

- The fully fused routine has fewer L1 cache misses than the routine with all outer loops fused and nearly identical L2 cache and TLB misses, so the crossover isn’t due to a cache or TLB issue.

5. Identifying the Reason for the Crossover

- An inspection of the assembly code for fully fused shows no register spilling at nevs = 4
  - Fully fused, nevs = 4
  - However, with each increase of nevs after 4, the fully fused code requires an extra memory access
  - While, when all outer loops are fused, the inner loops don’t require any extra memory access, even as nevs increases

- Reason for the crossover: register pressure
- Suggests ways to improve the analytic model
- Incorporate registers into its prediction

6. Conclusions

- Compiler for annotated MATLAB uses empirical/analytic approach to choose optimizations, producing tuned C code.
- Our tests show that the analytic model can be enhanced by considering register allocation.
- Future work: use results to identify when other optimizations such as cache blocking and array interleafing, in addition to loop fusion, become profitable.

Acknowledgements

This work was supported by the National Science Foundation under grant CCF-0810543.