

CS515: Programming Languages and Compilers I

Project 3: Loop-level, Memory-Hierarchy Optimizations for Matrix Multiply

Due date: Thursday, December 13

Project Description

Highly optimizing compilers use source-level transformations to improve the memory-hierarchy performance (execution time) of input programs. This project consists of two parts: the actual evaluation of different source-level transformations and their impact on the program performance, and (2) a brief report that summarizes your findings (2-3 pages).

You will apply your source-level, memory-hierarchy optimizations to your matrix-multiply example and measure the execution times using two different compilers. These compilers should not perform any advanced loop-level optimizations. This can be done by not specifying any optimizations level, i.e., using the default optimization level.

You may consult conference and journal articles, and the web to learn more about possible optimizations.

Experimental Methodology

You should start with the “standard” matrix multiply for square matrices of size “n”. The element type of the matrices should be double precision (8 bytes).

```
for i=1,n
  for j=1,n
    c(i,j) = ...

// --- TIMING START
for i=1,n
  for j=1,n
    for k=1,n
      c(i,j) = c(i,j) + a(i,k) * b(k,j)

// --- TIMING END
```

As your target machine, you should use one of the ilab machines (cereal cluster). Run the different versions of your program only on a single CPU in a Dual-Core system. Make sure that you choose the range of problem sizes n such that you can show the benefits of your optimized code versions over the original version. Make sure that your optimized program generates the correct result. You may use one of the following compilers: `f77` (Fortran), `javac`, or `gcc`.

Your report should show two graphs, one for each compiler. The x-axis of the compiler is the problem size n . The y-axis is the overall execution time (excluding the initialization loop for `c(i,j)`). Each graph has multiple lines, one line for each version of your optimized code. The graph should also show the baseline, which is the unoptimized, original version of the code.

The report should contain a conclusions section that discusses what you learned from performing these experiments.

Grading

Your project will be graded on correctness and clarity. You should have at least three optimized program versions for each compiler. Your report should include a justification and explanation of the observed results.

Good luck!