CS 4504 Computer Organization
Prof. Kirk W. Cameron
Spring 2006 Project 2: Speedup of Matrix Multiply

Important Dates

Assigned: Feb 28, 2006
Official due date: March 21, 2006
Extended due date: March 23, 2006 **(-10 points for late submission)**

Summary

In this project you are expected to speedup a naïve implementation of Matrix Multiply available as C code at the class home page. You will produce a report (< 6 pages) describing your approach and results. You may attempt additional work for extra credit (adding 2 additional pages to the report).

Details

You are to optimize two versions of the naive matrix multiply (matmul.c). A is an m x k matrix and B is a k x n matrix.

Version 1: A is a general m x k matrix, B is a general k x n matrix. For simplicity you may assume m = k = n or square matrices. Compare your "hand optimized version" to the compiler's version with optimizations turned off (the base case) and full optimizations turned on. See if you can beat the compiler through hand tuning.
Version 2: Modify the matrix multiply Version 1 optimized for non-square matrices.
Extra-credit: Create a naïve matrix transpose code that takes a general matrix with m x k entries of doubles and transposes it in memory to a k x m matrix. Use the un-optimized version as the base case and see if you can optimize the transpose to beat the compiler.
Extra-extra-credit: Modify your matrix transpose code to spawn two threads that perform transposes on the data and then exchange two rows or two columns of data from the matrices. For row or column "interchange," determine what gives better performance: 1) the case where you transpose a row or column, then exchange it OR 2) just doing a normal in-place exchange of the row or column (i.e. no transpose).

For ALL versions, your code must produce correct results. This should be verified in your report as well as your timing data. You may time your code using standard Unix utilities or the equivalent on other systems. I recommend "getrusage" for most accurate timing – see the LMbench code for examples in "timers.h". Hardware counters are even better. If you must, use of the "clock" command is acceptable, but may not be accurate enough for your work.

Here are some other guidelines:

1. You may not make any optimizations based on the numerical values of the matrix entries (though they are calculated for you in the naive example).

2. You must turn off ALL compiler optimizations when taking performance measurements of your hand-tuned versions of the code and for baseline comparisons.
3. You must be prepared to repeat your experiments in person upon request.
4. To see how you're doing, you can temporarily turn ON the compiler optimizations to gauge whether or not you can beat the compiler's performance.
5. You will deliver a short report (< 6 pages) illustrating your results under the detailed guidelines below.
6. You will turn in a hard copy of your report on the due date.
7. You will turn in a soft copy (electronic version) of your code upon request.

You may use any architecture you wish, and any programming language you wish, however support from the instructor and TA will be limited to Unix and C implementations only. We'll do our best to answer conceptual questions on other platforms. Small non-monetary prizes will be awarded to the best achieved speedup in various categories.

Experimentation

You should compile and run the naïve Matrix Multiply with **compiler optimizations turned OFF**. How to do this will be explained in the documentation of your compiler (This typically involves specifying a command line option when invoking the compiler.) This is the base performance of your system for matrix multiply[1]. Your assignment is to speed up the performance of this code by changing the algorithm implementation, thus circumventing the optimizations of the compiler. Discussions in Chapter 5 of your text provide a good starting point for such techniques. You will have to use timing functions native to your platform embedded in your code to measure the performance. An example is the getrusage() function in Unix environments. Hardware counters can also be used. All platforms will provide some similar mechanism. You will be graded on the speedup achieved in relation to your peers and the quality of the short report describing your results.

Required sections of paper:

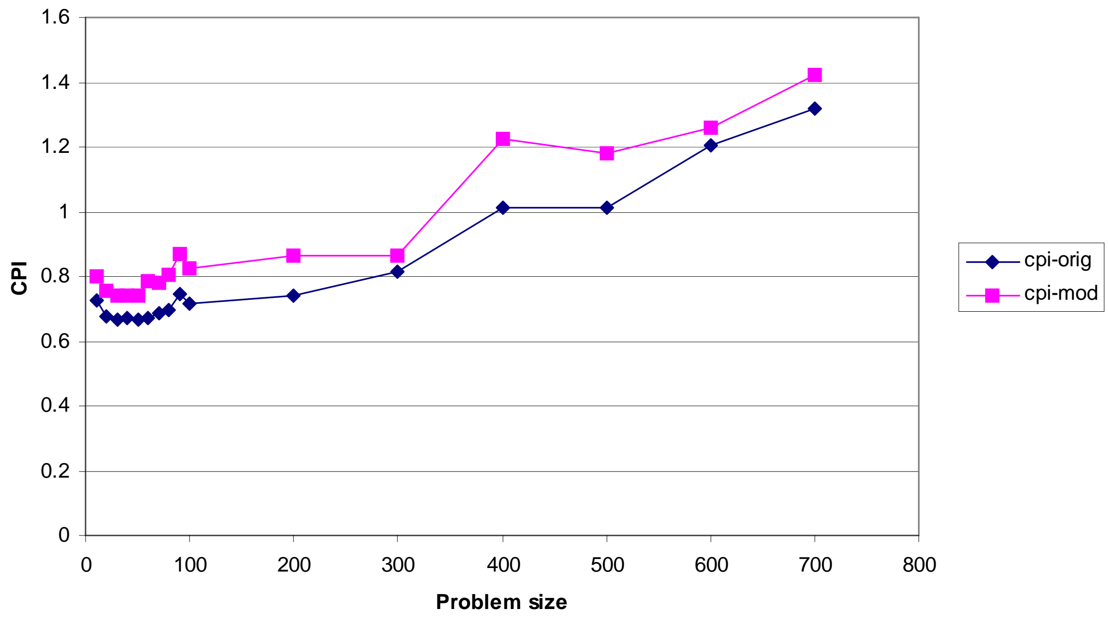
< Failure to follow these guidelines will result in points taken off. >

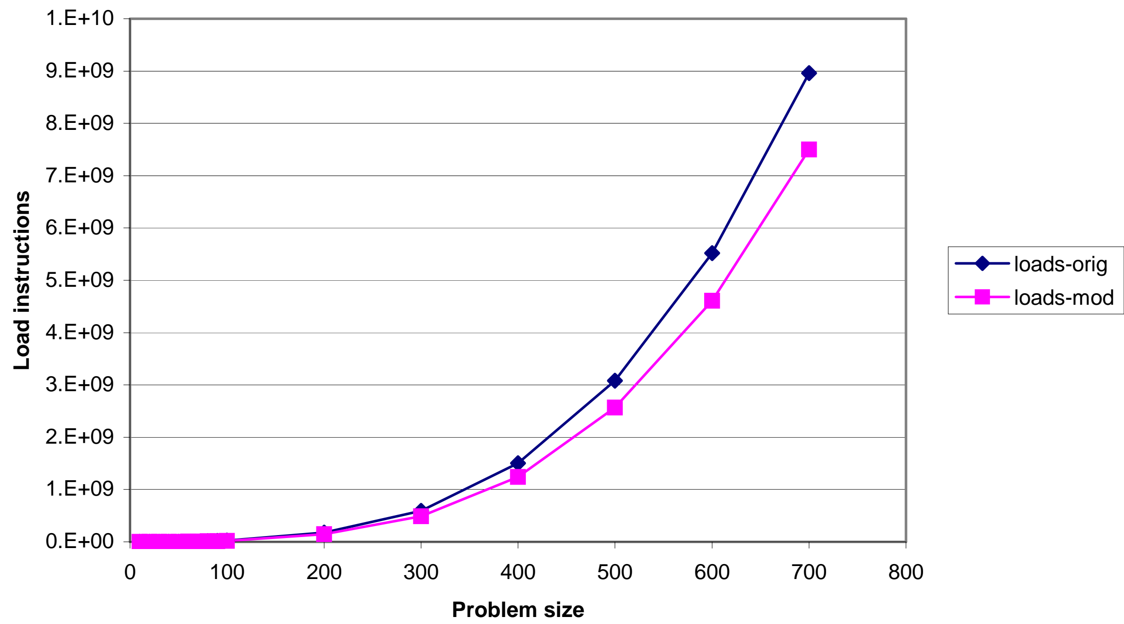
1. *Title and author info:* You name, affiliation, address, email, phone, etc. should appear towards the top of the first page of your report. I suggest 14-point font for title, and 12-point font for the rest of your document. Document should be single-spaced, single column, 12 point font. Margins should be 1 inch from all sides. Use numeric headings and subheadings as necessary. If English is not your first language, I suggest you utilize on-campus editorial resources or your peers to ensure correct grammar. This page does not count against the 6 page limitation.

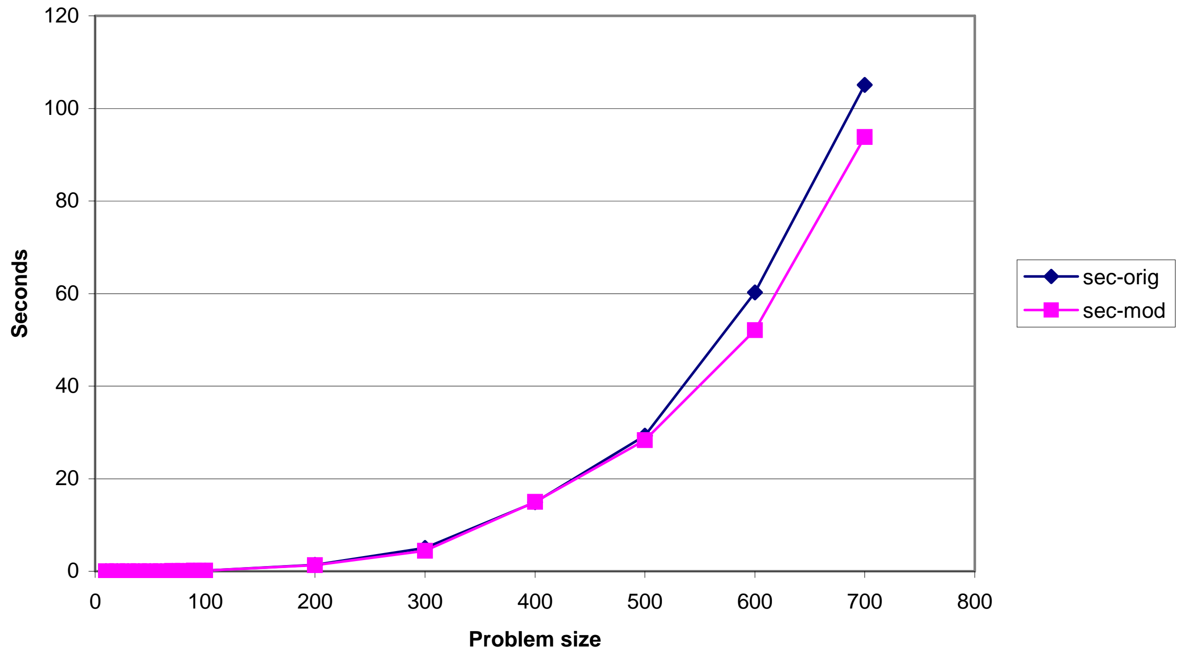
[1] Yes, this is possible in ALL compilers worth their salt. For example, in MS Visual C++ 6.0, see the pop up menu for Project-->Settings under Optimizations.

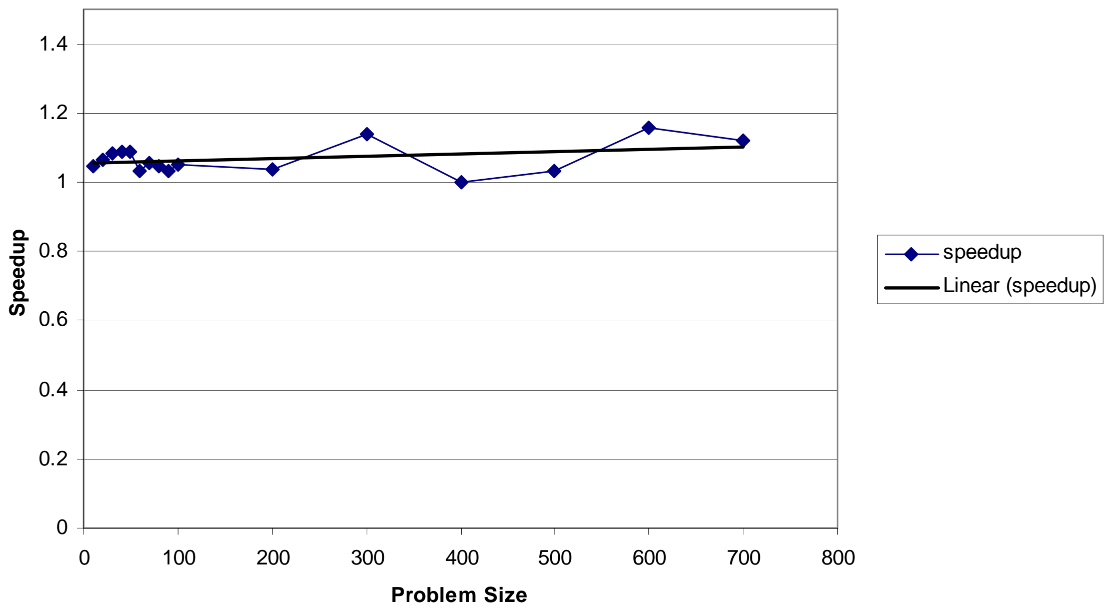
2. *Abstract:* On the first page of your document, provide an abstract that summarizes your work, approach, and findings. This should include exact numbers for your achieved speedup. This should provide the reader with an overview of your contribution without having to read the paper. This should be no longer than 200 words.
3. *Related Work:* If you utilized algorithms or approaches that you did not invent, you should cite them accordingly in your paper. This is to be your own work, where you apply optimizations by hand to the code given by the instructor. You may use other algorithms and techniques provided you understand and can explain how they work, and you implement the code yourself. Here you should simply identify others' work and discuss its relationship to your optimized matrix multiply. This section should be less than 300 words. I will use electronic means to compare your code to others in the class and the work of others. Cheating in any form will result in an F on the project and probable expulsion from the class. Save the particulars of your approach for the next section.
4. *Methodology:* This section should describe exactly the method you used to speedup the application. You should provide a step-by-step accounting of your approach. Why you did what you did, and how it might be different or the same from previous approaches. You should discuss your methods for attempting to speed up the code given and how you believe they will affect the code generally. Save exact numbers for the next section. Be succinct. Keep this section under 500 words.
5. *Results:* Here you should provide the empirical timing measurements for the matrix multiply you implemented verse the base version(s). You should summarize the details of the system upon which implementation took place (architecture, platform) and the timing methods you used (and why you chose both arch and timing method). You should provide charts and tables as necessary to show your results and discuss the results in the context of your methodology from the last section. This section should contain mostly graphs (like those found at the end of this document) and some analysis of what worked, why you think it worked, and what didn't work.
6. *References:* IEEE Journal citation format should be followed. You must cite all references. Plagiarism is unacceptable. Formatting details can be found at:
   http://www.ece.uiuc.edu/pubs/ref_guides/ieee.html

The next several pages provide some results from my simple experiments. I used an IRIX platform and the 200 MHz SGI MIPS R10000 processor. Problem sizes from 10x10 to 700x700 are shown (although I have data up to 1000x1000) for the naïve matrix multiply original code (orig) and an algorithmically modified version (mod).


CPI for matrix multiply
CPI
Problem size
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
0
100
200
300
400
500
600
700
800
cpi-orig
cpi-mod


Number of loads
Load instructions
Problem size
0.E+00
1.E+09
2.E+09
3.E+09
4.E+09
5.E+09
6.E+09
7.E+09
8.E+09
9.E+09
1.E+10
0
100
200
300
400
500
600
700
800
loads-orig
loads-mod


Total seconds for matrix multiply
Seconds
Problem size
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
800
sec-orig
sec-mod


Speedup Curve
Speedup
Problem Size
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
100
200
300
400
500
600
700
800
speedup
Linear (speedup)