Project Assignment
Hardware and Software for Scientific Computing

Andreas Persson

Original version in Swedish by Fredrik Warg, Johan Moe and Lars Svensson

Department of Computer Science and Engineering
Chalmers University of Technology

March 26, 2009

1 Background

The solving of large systems of equations is an important and demanding task in many scientific and technical applications. Gaussian elimination is a commonly used method for solving systems of linear equations, that is, systems of the form $\mathbf{Ax} = \mathbf{b}$.

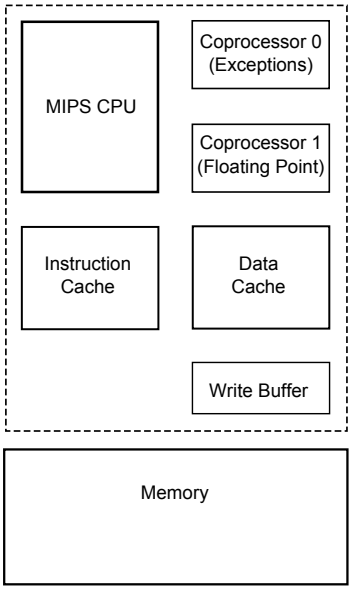
The elimination is a two-step process. In the first step, the coefficient matrix **A** is reduced to upper triangular form (a matrix with ones on the main diagonal and zeros in all positions below the main diagonal). The second step uses back substitution to find **x**. The C code in Figure 1 performs the first step of Gaussian elimination.

```
/*
 * reduce matrix A to upper triangular form
 */
void eliminate(double **A, int N)
{
  int i, j, k;
  for (k=0; k < N; k++) {          /* loop over all diagonal (pivot)
                                      elements */
    for (j=k+1; j<N; j++) {        /* for all elements in pivot row
                                      and right of pivot element... */
      A[k][j] = A[k][j] / A[k][k]; /* divide by pivot element */
    }
    A[k][k] = 1.0;                 /* Pivot element is now 1 */

    for (i=k+1; i<N; i++) {        /* for all elements below pivot row
      for (j=k+1; j<N; j++) {         and right of pivot column... */
        A[i][j] = A[i][j] - A[i][k] * A[k][j];
      }
      A[i][k] = 0.0;
    }
  }                                /* end pivot loop */
}                                  /* end eliminate */
```

Figure 1: Triangularization of a matrix A of size $N \times N$.

A computer manufacturer, Moon Macrosystems, has decided that their next product must handle this kind of computation more efficiently (their customers are just crazy about Gaussian elimination). They need your help with developing an assembler routine for the elimination algorithm in Figure 1 and with tailoring the memory subsystem of their new computer to fit the data access patterns of the algorithm. The goal is to find the best possible trade-off between price and performance.



Caches	
Word size:	4 bytes
Cache size:	32, 64, 128 words
Block size:	1, 2, 4, 8 words
Associativity:	1, 2, 4
Replacement policy:	LRU
Write policy:	Write back
Write Buffer	
Size:	0...8 words
Memory	
Access time:	80+20 ns, 50+12 ns

Figure 2: Computer hardware.

2 System Description

The computer system consists of a MIPS compatible microprocessor with five pipeline stages and delayed branch. There are also two coprocessors, one for memory management and exception handling (coprocessor 0) and one for floating point arithmetic operations (coprocessor 1). The processor core and the coprocessors may not be modified or replaced.

The memory system consists of separate instruction and data caches. The block replacement policy is Least Recently Used (LRU) and the write policy is Write Back. The data cache has a write buffer to reduce the number of stall cycles due to memory writes.

Cache size, block size, associativity and write buffer size are configurable parameters. Available configurations are given in Figure 2. Depending on what type of memory module you choose, the main memory has an access time of either 80 ns for the first word in a block and 20 ns for following words or 50 ns for the first word and 12 ns for following words. Component costs for the possible configurations are given in Appendix A.

$$\begin{pmatrix} 57 & 20 & 34 & 59 \\ 104 & 19 & 77 & 25 \\ 55 & 14 & 10 & 43 \\ 31 & 41 & 108 & 59 \end{pmatrix} \qquad \begin{pmatrix} 1.00 & 0.35 & 0.60 & 1.04 \\ 0.00 & 1.00 & -0.86 & 4.73 \\ 0.00 & 0.00 & 1.00 & -0.41 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{pmatrix}$$

(a) Before elimination (b) After elimination

Figure 3: An example of gaussian elimination.

3 Algorithm Description

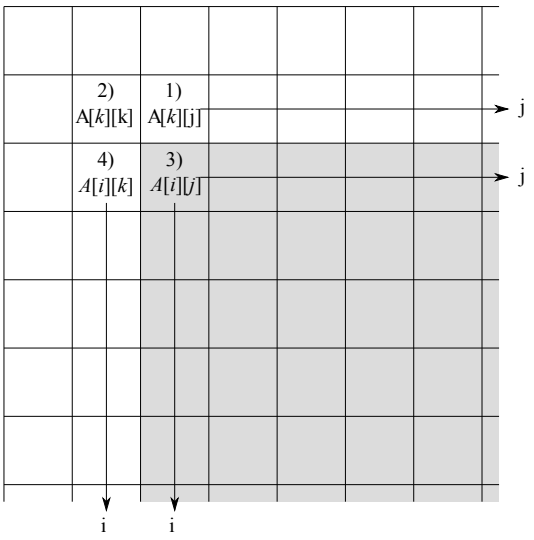
As mentioned in Section 1, the first step of Gaussian elimination is to reduce a matrix to triangular form. In this assignment you don't need to worry about the second step (back substitution). Figure 3 shows a 4×4 matrix before and after the elimination step.

The elimination routine contains division operations. Thus, the result will not always be integer numbers. If you would round all results to integers you would end up with unacceptably large errors, especially at the bottom of the matrix where elements are updated many times. In order to maintain good precision throughout the computation you should use floating point representation for the matrix elements and floating point instructions for all matrix computations. Matrix computations are highlighted with **bold font** in Figure 1. Chapter 3.6 of the course book[1] gives more information about floating point representation and MIPS floating point instructions. Single precision floating point representation (32 bits) gives sufficient numeric precision for the application at hand.

Figure 4 is a graphical visualization of the algorithm from Figure 1. It is not necessary to know exactly what the algorithm does, but you will probably find it interesting to analyze the data access patterns. Is it possible to take advantage of locality in order to reduce the number of cache misses?

An alternative version of the elimination algorithm is given in Appendix C. What is the difference between the two versions? Can you think of any situations where the alternative algorithm will perform better?

[1] Hennessy & Patterson: Computer Organization and Design, Third Edition.



1) First *j* loop:

```
A[k][j] = A[k][j] / A[k][k]
```

2) After first *j* loop:

```
A[k][k] = 1.0
```

3) Nested *j* loop:

```
A[i][j] = A[i][j] - A[i][k] * A[k][j]
```

4) *i* loop:

```
A[i][k] = 0.0
```

Figure 4: Data access pattern for one iteration of the elimination algorithm (k = 1).

4 The Assignment

1. Implement the elimination routine in MIPS assembler. Use single precision floating point instructions for all matrix computations. Your program should run in the Mars MIPS simulator (see Appendix B).

2. Construct a computer system considering both prize and performance. $Performance$ is measured as the time it takes your program to triangularize a 24×24 reference matrix. *Price* is the sum of the component costs for processor, caches and memory. Your goal is to minimize the product $Price \times Performance$. The unit of this measure is $\mu sC\$$ (microSecondChalmersDollar).

Competition: Who constructs the best system? A scoreboard with the top results will be published on PingPong.

5 Examination

The following is required in order to pass the assignment:

1. A live demonstration of your program (during a lab session or group exercise).
2. A report **written in English** containing:
   - A description of the computer system and the program. Someone with appropriate background (for example a course like this) should be able to understand the report even if he or she hasn't read this document.
   - The source code for assignment 1 as an appendix.
   - The answer to assignment 2: which is the best $Price \times Performance$ value you have obtained? A good motivation for your choice of parameters is necessary. A working program alone is not sufficient. We want to know what considerations you made when choosing components and what measures you took to improve the execution time of the program. **Your motivation is more important than the exact result. You should include data from several test runs with different configurations to show that your solution is good**.
   - To simplify comparison and recreation of results, you should report the following data for each of your test cases:
     - **Configuration:** Cache size, associativity, block size, processor frequency, memory access time and write buffer size.
     - **Results:** The number of clock cycles needed to triangularize the 24×24 matrix, execution time (μs), total component cost ($C\$$) and $Price \times Performance$ ($\mu sC\$$).
   - The length of the report should be 2-4 pages (not counting source code).

Deadlines for the demonstration and the report are given on PingPong.

6 Hints

- The assembler file `gauss_template.s` contains some code that might help you get started. It also contains the 4×4 matrix from Figure 3(a) and the 24×24 matrix you should use when reporting your results. Matrices are stored in row-major order.
- The assembler routines in `gauss_template.s` may be changed or not used at all. You are free to write your own helper routines. It is up to you to choose which algorithm to use. You may use your own algorithm, as long as the result is correct and the numeric precision is no worse then that of the given algorithms.
- It is probably a good idea to get a working program running using the given helper routines before bothering about optimizations.
- Use the small matrix when developing and debugging your code. However, you should try larger matrices as well to see how the performance scales with problem size. A small matrix might fit entirely in the data cache. The 24×24 matrix certainly does not.
- One tricky thing about the assembler code is that you need to keep track of a number of loop/array indices. Use code comments to document what you have in the different registers (there's nothing wrong with a comment for each line of assembler code!).
- For integer arithmetic, use instructions that do not result in overflow exceptions (addu, addiu, subu, subiu). In this particular exercise you may disregard from possible exceptions.
- Most of you are probably more familiar with Java than with C. We hope that the given C code is similar enough to java to be understandable. Contact a supervisor if you have questions about the code.

A Component Prizes

In this assignment we use a made-up currency called Chalmers Dollar (C$). The component prizes below does not correspond to actual prizes in any real currency.

CPU

The stock CPU operates at a clock frequency of 100 MHz. The prize is 2 C$.

Caches

128 byte instruction and data caches with block size 4 and associativity 1 are included in the price of the CPU. Larger caches will cost you more. The caches are the slowest components of the system. Hence, your choice of caches will affects the clock frequency of the CPU. The table below shows maximal clock frequencies for the available cache configurations. If you choose different configurations for the instruction and data caches, the slowest component will determine the system clock frequency. Available block sizes are 1, 2, 4 and 8 words. The choice of block size does not affect the prize or the clock frequency.

Cache Size (words)	Associativity	Max Clock Frequency (MHz)	Additional Cost (C$)
32	1	100	0
32	2	95	0
32	4	90	0
64	1	95	0.25
64	2	90	0.25
64	4	85	0.25
128	1	90	0.5
128	2	85	0.5
128	4	80	0.5

Write Buffer

The cost of buffer space is 0.05 C$/word. You are allowed to use a write buffer of up to 8 words, or none at all.

Memory

There are two types of memory modules to choose from. Type 1 has an access time of 80 ns for the first word in a block and 20 ns for following words. Type 2 has an access time of 50 ns for the first word in a block and 12 ns for following words. The cost is 0.5 C$ for type 1 and 0.75 C$ for type 2. One of them must be included in your system.

B MARS MIPS Simulator

For this assignment you should use the MARS simulator[2]. Compared to MipsIt (the tool used in the laboratory exercises), MARS is a high-level simulator. Rather than simulating the logic of the CPU, it simulates the behaviour of instructions and their effects on registers and memory. This approach has advantages, such as increased simulation speed and generality with respect to actual MIPS implementations. On the other hand, behavioral simulation does not reflect how pipeline hazards and cache memory misses affects the execution time of a program.

Evaluating Performance

We provide a plug-in for MARS that allows detailed measurements of program execution times. This utility is launched via the "Tools" menu in MARS (the tool name is "EDA331 Performance Evaluation Tool"). **Note: The performance evaluation tool is not part of the package distributed via the MARS web site. In order to use the tool, you need to download the MARS .jar archive from PingPong**.

The tool simulates a memory bus and instruction and data caches. It also contains a simplified model of a floating point coprocessor. The simulated coprocessor is non-pipelined and has multi-cycle instructions. The table below lists cycle counts for floating point operations. After issuing a floating point instruction, the processor will stall until the result is available.

Operation	Single	Double
move, absolute value, negate	1	1
add, compare	2	2
multiply	4	5
divide	12	19

[2] http://courses.missouristate.edu/KenVollmar/MARS/

C Alternative algorithm

```
/*
 * reduce matrix A to upper triangular form
 */
void eliminate(double **A, int N, int B)                 /* triangularize matrix A */
{
  int i, j, k, I, J;
  int M = N/B;                                           /* size of block (m*m) */
  for (I = 0; I < B; I++) {                              /* for all block rows */
    for (J = 0; J < B; J++) {                            /* for all block columns */
      for (k = 0; k <= Min((I+1)*M-1, (J+1)*M-1); k++) { /* loop over pivot elements */
        if (k >= I*M && k <= (I+1)*M-1) {                /* if pivot element within block */
          for (j = Max(k+1, J*M); j <= (J+1)*M-1; j++) { /* perform calculations on pivot
            A[k][j] = A[k][j] / A[k][k];                    row right of pivot element */
          }
          if (j == N) A[k][k] = 1;                       /* if last element in row */
        }
        for (i = Max(k+1, I*M); i <= (I+1)*M-1; i++) {   /* for all rows below pivot
                                                            row within block */
          for (j = Max(k+1, J*M); j <= (J+1)*M-1; j++) { /* for all elements in
                                                            row within block */
            A[i][j] = A[i][j] - A[i][k] * A[k][j];
          }
          if (j == N) A[i][k] = 0;                       /* if last element in row */
        }
      }
    }
  }
}                                                        /* end Eliminate */
```