Parallel Computer Architecture: Homework 1

February 1, 2012

1 2.7

Problem description:

2.7 Gaussian elimination is a well-known technique for solving simultaneous linear systems of equations. Variables are eliminated one by one until there is only one left, and then the discovered values of variables are back-substituted to obtain the values of other variables. In practice, the coefficients of the unknowns in the equation system are represented as a matrix A, and the matrix is first converted to an upper-triangular matrix (a matrix in which all elements below the main diagonal are 0). Then back-substitution is used. Let us focus on the conversion to an upper-triangular matrix by successive variable elimination. Pseudocode for sequential Gaussian elimination is shown in Figure 2.18. The diagonal element for a particular iteration of the k loop is called the *pivot element*, and its row is called the *pivot row*.

a. Draw a simple figure illustrating the dependences among matrix elements.

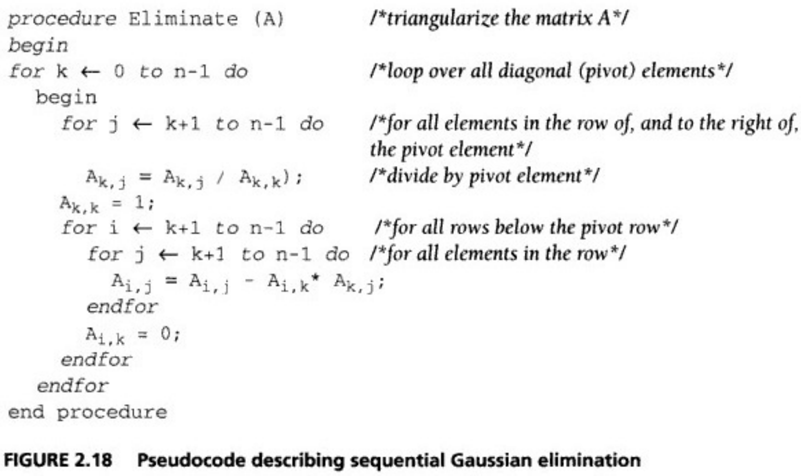
b. Assuming a decomposition into rows and an assignment into blocks of contiguous rows, write a shared address space parallel version using the primitives used for the equation solver in this chapter.

c. Write a message-passing version for the same decomposition and assignment, first using synchronous message passing and then any form of asynchronous message passing.

d. Can you see obvious performance problems with this partitioning? (We will discuss this further in the next chapter.)

e. Modify both the shared address space and message-passing versions to use an interleaved assignment of rows to processes.

f. Discuss the trade-offs (programming difficulty and any likely major performance differences) in programming the shared address space and message-passing versions.



```
procedure Eliminate (A)             /*triangularize the matrix A*/
begin
for k ← 0 to n-1 do                 /*loop over all diagonal (pivot) elements*/
  begin
    for j ← k+1 to n-1 do           /*for all elements in the row of, and to the right of,
                                      the pivot element*/
      A_{k,j} = A_{k,j} / A_{k,k});  /*divide by pivot element*/
    A_{k,k} = 1;
    for i ← k+1 to n-1 do            /*for all rows below the pivot row*/
      for j ← k+1 to n-1 do         /*for all elements in the row*/
        A_{i,j} = A_{i,j} - A_{i,k}* A_{k,j};
      endfor
      A_{i,k} = 0;
    endfor
  endfor
end procedure
```

FIGURE 2.18 **Pseudocode describing sequential Gaussian elimination**

Figure 1: figure 2.18

1.1 2.7(a)

There are three cases as showed following. Red is used to mark the target entry. Shadow area shows the dependent entries in the matrix.

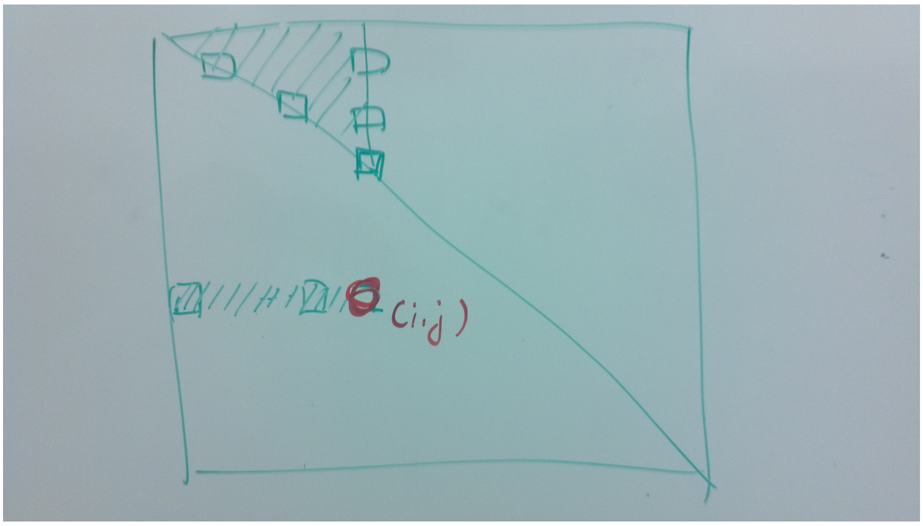


Figure 2: dependences of entry $a_{i,j}$, where i ¿ j

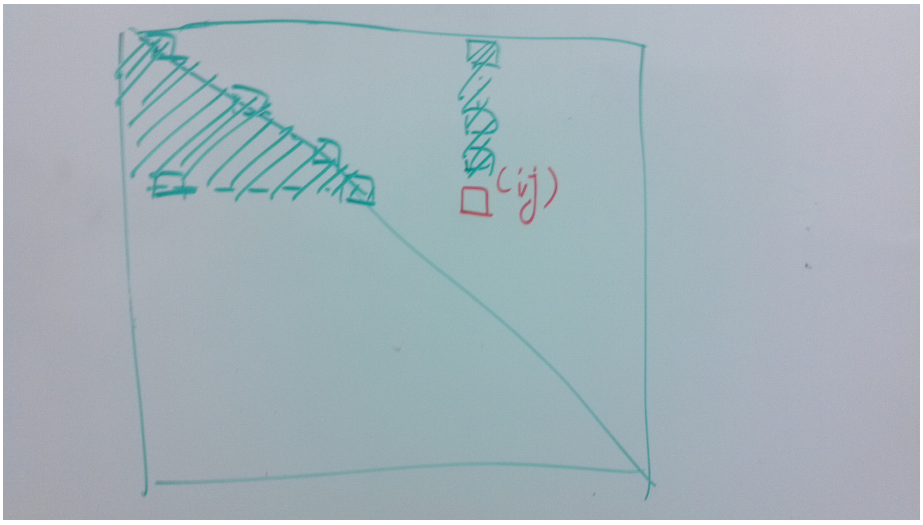


Figure 3: dependences of entry $a_{i,j}$, where i ¡ j

In the last case, we have $a_{i,j}$ for $i = j$.

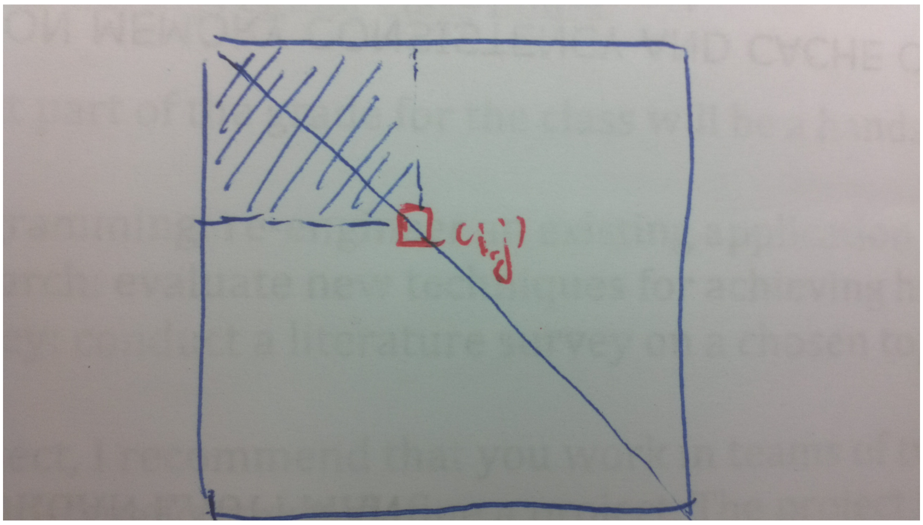


Figure 4: dependences of entry $a_{i,j}$, where i = j

1.2 2.7(b)

See src/b.cpp.

1.3 2.7(c)

TODO: what's any form of async message passing?

Here's sync message passing program. (draft code, not compiled)
See src/c.cpp

1.4 2.7(d)

The only issue I see about the partitioning is that each pivot has different number of workload. This imbalance could caue some threads/processes remaining idle.

1.5 2.7(e)

I didn't see a reason why we want an interleaved version. There's no dependency between consecutive rows.

1.6 2.7(f)

In this particular problem, I would prefer to use shared variable approach.

1. It's much shorter, as I mentioned in the class.
2. Message passing version involves a huge data communication cost,

especially in a network environment. Because in each iteration, we need to spread out almost the whole matrix.