ECSE 420-Parallel Computing
Assignment 1

1-Describe briefly the following terms, expose their cause, and work-around the industry has undertaken to overcome their consequences:

a. Memory wall

CPU speeds double approximately every eighteen months, while main memory speeds double only about every ten years. The diverging rates implies that memory speeds are causing a bottleneck (von Neuman bottleneck) in attaining system performance increases. The limited throughput between CPU and memory compared to the amount of memory available means that in modern machines, the CPU is continuously forced to wait for data to be transferred to or from memory.

Overcoming this issue means finding ways to overcome the "memory wall" i.e. to diminish the impact of slow memory on system performance. One solution is to introduce small amounts of very fast (and expensive!) **"cache"** memory between the CPU and main memory in order to reduce the cost of accessing frequently used data from main memory. Other methods such as **instruction level parallelism, branch prediction algorithms**, **out-of-order execution**, **speculative execution** etc…) try to reduce the time the CPU is waiting for data to be fetched from memory by performing (potentially) useful work before the CPU needs it.

b. Frequency wall

Since present day microprocessors are built using CMOS technology, how fast a processor can be clocked depends on how small of a switching time each transistor can have. The **switching time** of a MOSFET goes down with its size (inversely proportional to its size). Because of the **physical limits** of the technology being use to manufacture these transistors, we have reached a point at which is it is no longer possible to increase the frequency at which the CPU operates.
These limits come from the technology, the targeted chip size, the targeted cooling technology, and the targeted power consumption. Clock rates have plateaud at about 3.8 GHz. In order to overcome these limitations, **instruction level parallelism** methods as previously mentioned have been introduced in order to raise the number of instructions per clock cycle that are executed. However, increasing the frequency typically requires **deeper pipelines and more power**. Increasing the length of the pipeline increases the chances of resource conflicts in the instruction stream which will stall the pipeline. The current solution is to move to parallel computing processors (thread level parallelism) by designing **multi-core processor** systems which typically operate at a **lower frequency** but can increase the throughput/performance of the system by significant amounts.

SISD-Single Instruction Single Data

SIMD-Single Instruction Multiple Data

MIMD: Multiple Instruction Single Data

MISD: Multiple Instruction Single Data

[1] MISD (**M**ultiple **I**nstruction, **S**ingle **D**ata) is a type of parallel computing architecture where many functional units perform different operations on the same data. Pipeline architectures belong to this type, though a purist might say that the data is different after processing by each stage in the pipeline. Fault-tolerant computers executing the same instructions redundantly in order to detect and mask errors, in a manner known as task replication, may be considered to belong to this type. Not many instances of this architecture exist, as MIMD and SIMD are often more appropriate for common data parallel techniques. Specifically, they allow better scaling and use of computational resources than MISD does.

Some argue that a systolic array is an example of a MISD structure

2- You should extend Amdahl's and Gustafson-Barsis bound and make it slightly more realistic. Assuming the fixed overhead *o* in the communication and the setup of parallel processes, derive the expressions for both bounds that take the overhead into account.

Realistically speaking, the overhead in the communication and the setup of parallel processes should usually depend on the number of parallel processes that are being used. However, we will assume that the overhead incurred is a fixed value that we can simply incorporate it into Amdahl's law as a constant to give us a good estimate of the running time.

Amdahl's Law
S: is the fraction of a program that has to be executed sequentially.
N: Number of processors
T(1): Is the running for a given program running on a single processor.
T(N): Is the running time for the same program running on multiple processors.
O: Fixed overhead in the communication and the setup of parallel processes.

If the program's running time on 1 processor is T(1) then the running time on N processor would be:

$$T(N) = S \times T(1) + \frac{(1-S)\times T(1)}{N} + O$$

$$\text{Speedup} = \frac{T(1)}{T(N)} = \frac{T(1)}{S \times T(1) + \frac{(1-S)\times T(1)}{N} + O} = \frac{N \times T(1)}{S \times T(1) \times N + (1-S)\times T(1) + ON}$$

$$= \frac{N}{S \times N + (1-S) + \frac{ON}{T(1)}} = \frac{N}{N \times (s + \frac{O}{T(1)}) + (1-S)}$$

Therefore, as we increase N the upper bound on the Speedup is:

$$\text{Lim}_{\text{N->}\infty}(\text{Speedup}) = \frac{1}{(s + \frac{O}{T(1)})}$$

We must therefore keep O and S as low as possible in order to achieve maximal speedup as N increases.

Gustafson-Barsis

S: is the fraction of a program that has to be executed sequentially.
N: Number of processors
T(1): Is the running for a given program running on a single processor.
T(N): Is the running time for the same program running on multiple processors.
O: Fixed overhead in the communication and the setup of parallel processes.

If the program's running time on N processor is T(N) then the running time on one processor for that same problem size would be

$$T(1) = S \times T(N) + (1-S) \times N \times T(N) - O$$

$$\text{Speedup} = \frac{T(1)}{T(N)} = \frac{S \times T(N) + (1-S) \times N \times T(N) - O}{T(N)} = \frac{T(N) \times (S + (1-S) \times N) - \frac{O}{T(N)})}{T(N)}$$

$$= (S + (1-S) \times N) - \frac{O}{T(N)}$$

Therefore, as we increase N the upper bound on the Speedup is:

$$\text{Lim}_{\text{N->}\infty}(\text{Speedup}) = \infty$$

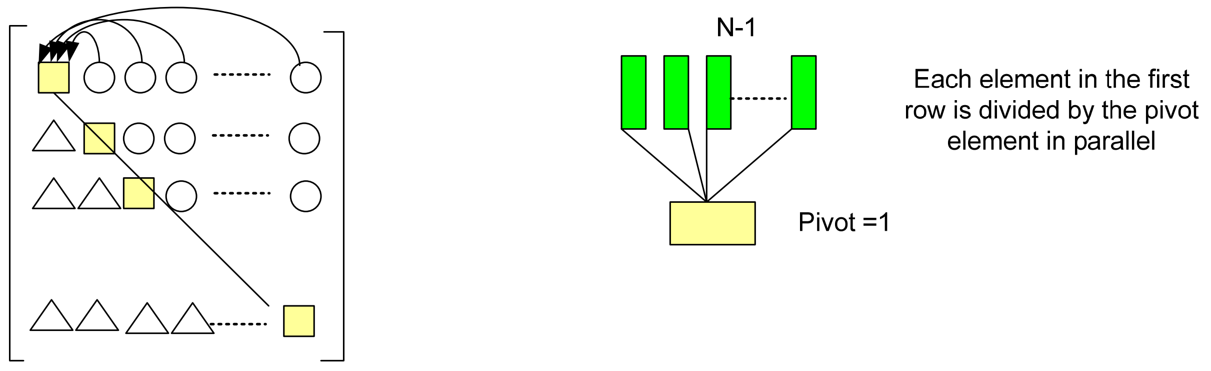
4-Gaussian elimination is a well-known technique for solving simultaneous linear systems of equations. Variable are eliminated one by one until there is only one left, and then the discovered values of variable are back-substituted to obtain the value of other variables. In practice the coefficients 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 the Fig 1. 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 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.

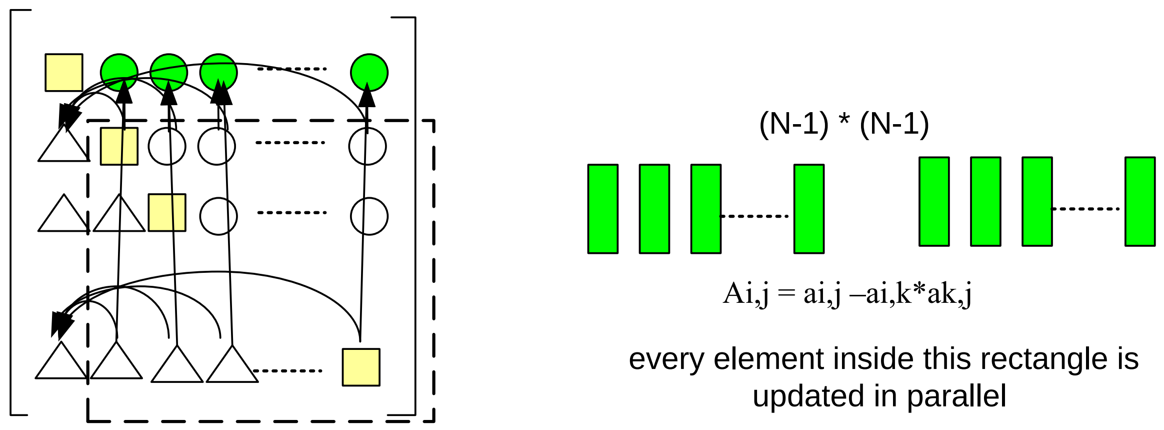
```
procedure Eliminate (A)             /*triangularize the matrix A*/
begin
for k ← 0 to n-1 do                 /*loop over all diagonal (pivot) elements*/
  begin
    for j ← k+1 to n-1 do           /*for all elements in the row of, and to the right of,
                                      the pivot 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
```

Fig.1 Pseudocode Describing Sequential Gaussian Elimination

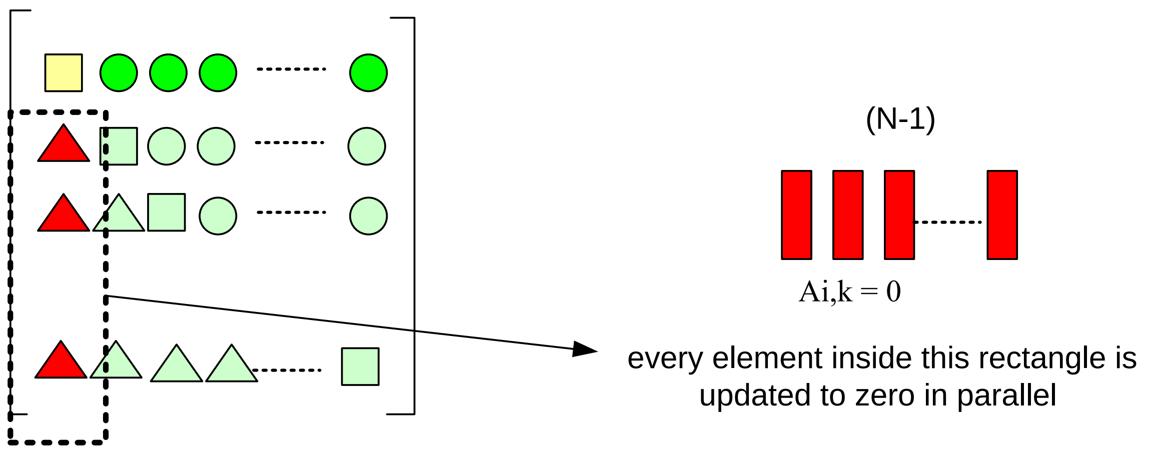
Gaussian Elimination (GE) is one of the direct methods of solving linear systems (Ax = b). In this method, first the matrix A is converted to the upper or lower triangular matrix. Thereafter, using substitution technique the value of the vector x is computed.



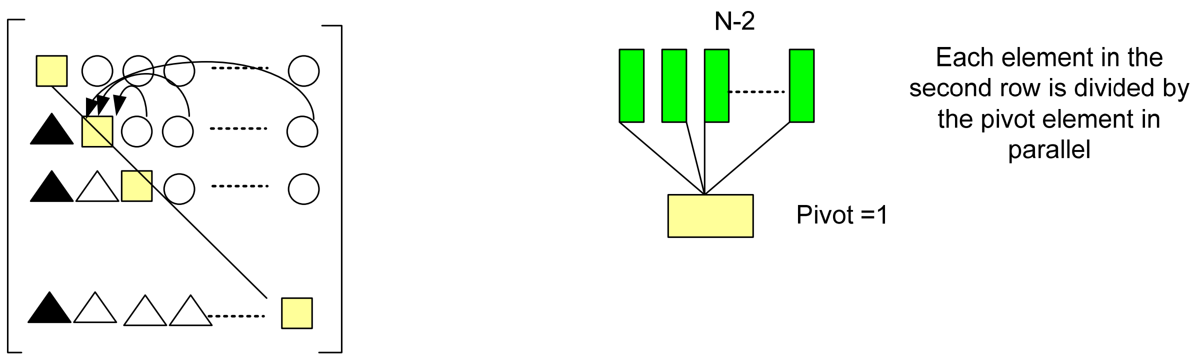
Each element in the first row is divided by the pivot element in parallel



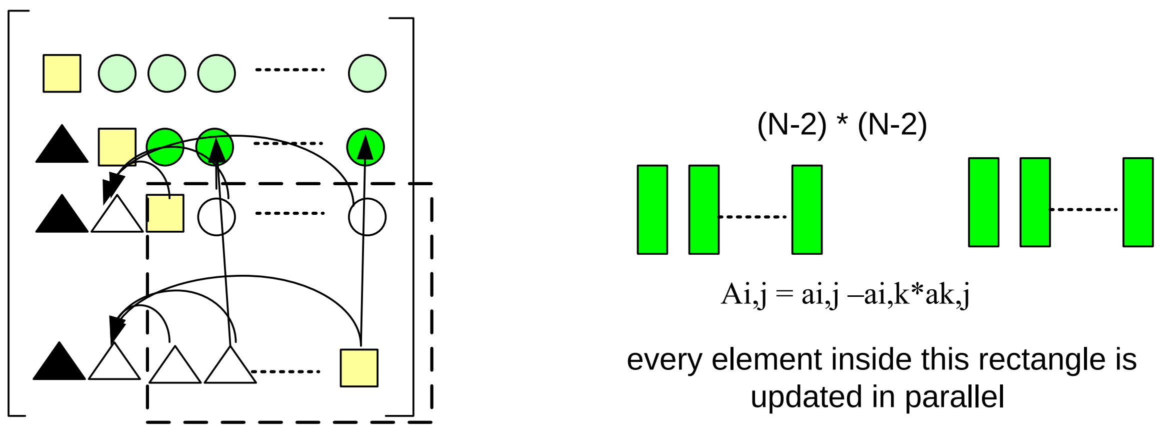
every element inside this rectangle is updated in parallel



every element inside this rectangle is updated to zero in parallel

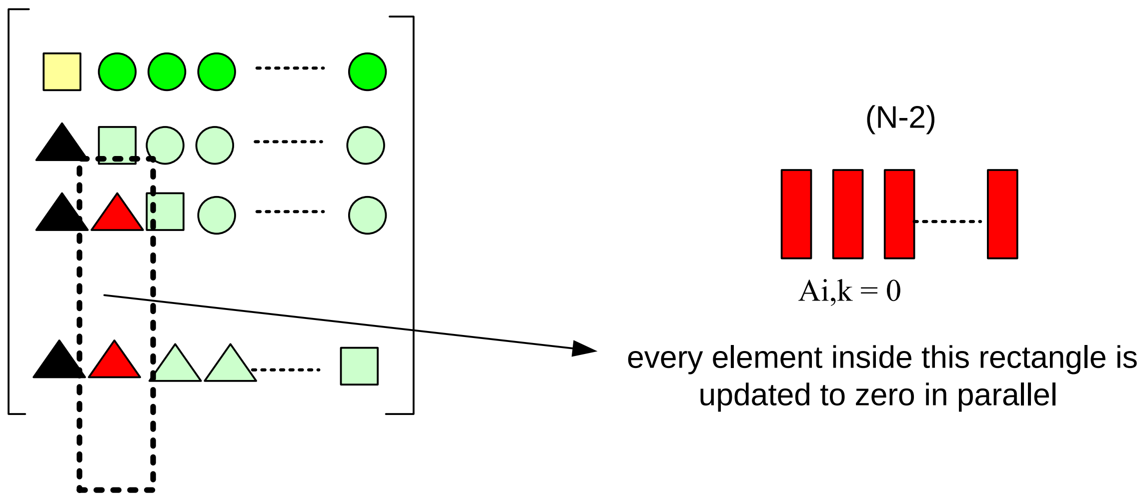


Each element in the second row is divided by the pivot element in parallel



$A_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$

every element inside this rectangle is updated in parallel



$A_{i,k} = 0$

every element inside this rectangle is updated to zero in parallel

```
For k<-0 to n-1
Begin
        (1)  For_all j<-k+1 to n-1 do
                 Ak,j= ak,j /ak,k

        (2)  Akk =1

        (3)  For_all i<- k+1 to n – 1do
                 For_all j<- k+1 to n-1 do
                         Ai,j=ai,j-ai,k*ak,j

        (4)For_all i<-k+1 to n-1 do
                 Ai,k = 0
End for
```

Pseudocode Describing the Parallel Gaussian Elimination without assignment

```
int n, nprocs; /*matrix dimension and number of processors to be used*/
float **A; /*A is global (shared) array representing the matrix/
BARDEC (bar1); /*barrier declaration for global synchronization between sweeps*/

main()
begin
  read(nprocs); /*read input matrix size and number of processes*/
  read (n);
  A ←G_MALLOC (n * n);
  initialize(A);
  **CREATE (nprocs–1, Eliminate, A);**
  Eliminate(A); /*main process becomes a worker too*/
  **WAIT_FOR_END (nprocs–1);** /*wait for all child processes created to terminate*/
end main
```

```
procedure Eliminate(A)
 begin
 **int mymin = 1 + (pid * n/nprocs);** /*assume that n is exactly divisible by*/
 **int mymax = mymin + n/nprocs - 1** /*nprocs for simplicity here*/

For (k =0 to n)
Begin
        BSRRIER(bar1,nprocs);
            if (k>=mymin && k<=mymax)
            begin
                        For  j<-k+1 to n-1 do
                        A[k][j]= a[k][j] /a[k][k]
            end
            a[k][k]=1;

            barier(bar1,nprocs);
            for i=max(mymin,k+1)to mymax
                        begin
                                    For_all j= k+1 to n-1 do
                                                A[i][j]=a[i][j]-a[i][k]*a[k][j]
                        end;

            barier(bar1,nprocs);
            for i=max(mymin,k+1)to mymax
                        a[i][k] =0

End;
```

Or

```
For (k =0 to n)
Begin
    BSRRIER(bar1,nprocs);
      if (k>=mymin && k<=mymax)
      begin
            For  j<-k+1 to n-1 do
            A[k][j]= a[k][j] /a[k][k]
      end
      a[k][k]=1;

      barier(bar1,nprocs);
      for i=max(mymin,k+1)to mymax
            begin
                  For_all j= k+1 to n-1 do
                        A[i][j]=a[i][j]-a[i][k]*a[k][j]
                  a[i][k] =0
            end;

End;
```

4. Suppose we have a machine with the message start-up time of 20000 ns and the asymptotic peak bandwidth of 900 MB/s. The machine is sending the messages with *n* bytes. The start-up time includes all SW and HW overhead on the two processors, accessing the network interface and cross the network – it can be thought of as the time to send the zero-length message. At what message length is machine reaching the half of the peak bandwidth?

Data Transfer Time can be obtained according to the following formula:

$$T(n) = n_0 + \frac{n}{B}$$

In this equation, n is the amount of data (e.g., in bytes), B is the transfer rate (bandwidth) in bytes per second and n0 is the start-up cost. Actually, bandwidth depends on the transfer size. However, asymptotic bandwidth is the bandwidth of a system taking into account the infinite number of data communication. Therefore, when n goes to infinity we can easily neglect the effect of n_0. Eventually, we obtain the following formula for B:

$$B = \lim_{n\to\infty} \frac{n}{T(n)}$$

Now, we are supposed to find the related message size for reaching half of the peak bandwidth. The easiest way to obtain the message size is using two previous formulas in the following way:

$$\frac{1}{2}B = \frac{n_{\frac{1}{2}}}{n_0 + \frac{n_{\frac{1}{2}}}{B}}$$

By solving above equation, we can reach to this equation

$N_{1/2}= Bn_0$

In this question we have:

T0 = 20000

B= 900 MB/s

$$n_{\frac{1}{2}} = 20{,}000ns \times 900MB/s = 20{,}000 \times 10^{-9} \times 900 \times 2^{20} \cong 18874.368 Byte$$

5- We are going to find the average of elements in grid of (n*n). Each element of this grid may be a mathematical expression. Therefore, every element of this grid requires computation. Based on the Amdhal's law compute the speed up of using K processors in these two following situation:

a. Each processor has its own private value for holding the sum.
b. The processors have to use one shared value to keep tracking of the sum. That is, every processor should sum its result to the shared variable of sum.

In this question you are supposed to figure out the effect of decomposition. Here, we have n^2 elements and K processors. So each processor could compute $\frac{n^2}{K}$ additions. If processors do not have a private variable, they have to add sequentially to the global variable the result of their intermediate additions. Hence, the below equation goes with T(K):

T(K) = $\frac{n^2}{K} + n^2 + 1$; therefore the maximum speedup is limited to :

$$\lim_{K \to \infty} \frac{T(1)}{T(K)} = \lim_{k \to \infty} \frac{2n^2}{\frac{n^2}{K} + n^2 + 1} = \frac{2n^2}{n^2 + 1} < 2$$

However, when each processor has a private sum the following formula goes with the speed up of N parallel processors. In this case, processors do not have to use the global sum sequentially. They use our own private sum keeping the result of intermediate additions.

$$\lim_{K \to \infty} \frac{T(1)}{T(K)} = \lim_{k \to \infty} \frac{2n^2}{\frac{n^2}{K} + \frac{n^2}{K} + 1} = \frac{n^2}{\frac{n^2}{k} + \frac{1}{2}} \cong k$$