# Midterm 2 - ECE 563 Spring 2012

This is a take-home test. You must do your own work, if found cheating you will be failed in the course and you will be turned in to the Dean of Students. To make it easy not to cheat, this exam is open book, open notes, and you may use passive online resources to solve it. By passive I mean you may look at existing online resources, you may not post queries or requests to information to any online resource.

For simplicity, some of the loop nests shown will lead to out-of-bounds array accesses. Ignore these when doing your test.

Please print and sign your name below. By doing so you signify that you have not received or given any prohibited help on this exam.

**Name: Key**

**Question 1 (14 pts):** Consider the $N$ by $N$ arrays A and B shown below.

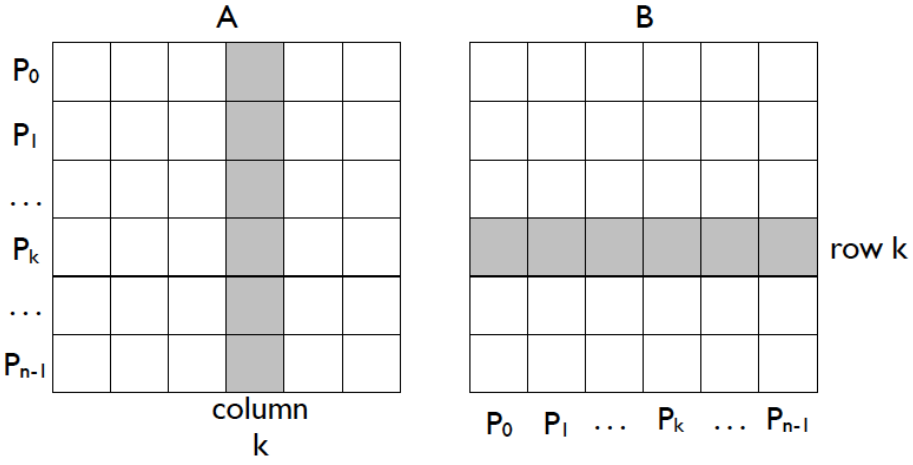
Each array has its rows distributed as indicated, i.e. array A is row distributed and array B is column distributed. The gray rows and columns contain non-zero values, and the white rows and columns contain zeros. In particular, column $k$ of $A$ is non-zero, and row $k$ of $B$ contains non-zero values, and all other rows and columns contain zero values.

The computation performed on these arrays is to compute $Z = A \times B$, i.e. matrices $A$ and $B$ are multiplied and the result is stored into $Z$, and then the sum of all of the elements of $Z$ are sent to each process. $Z$ is distributed by rows, just like $A$.

$Z = A \times B$
MPI_AllReduce(*Z(pid*N), sum, N*N/*num_procs, MPI_FLOAT, MPI_PLUS, MPI_COMM_WORLD);

It is the intention of the AllReduce for each process to send the rows of $Z$ that they own to the AllReduce.

**a.** How many operations are performed to compute $Z = A \times B$? You can count a*b+c as a single operation. You should not multiply the zero elements.
**b.** Derive the isoefficiency function for the matrix multiply and AllReduce. Assume that $A$ is distributed by rows, and $B$ is distributed by columns, as shown in the figure below.




**A solution:**

**Analysis for the $Z = A \times B$ computation**: Each square contains 1 non-zero element. Each element A[r][c] after performing $Z = A \times B$ contains A[r][k]*A[k][c], where c is a column and r is a row. A total of $N^2$ values are computed, requiring a total of $N^2$ multiply operations. Thus $T_1 = N^2$. Each processor does N2/P of these computations, and needs to get a value from each of the other P-1 processors. Thus $T_P = N^2/P + P - 1$.

As noted in the solution to hw9a:

$T_O = PT_P - T_1 = P(N^2/P + P - 1) - N^{2j} = N^2 + P^2 - P - N^2 = P^2 - P.$ Because the largest term is what determines the scalability, the work must scale proportional to $P^2$.

If broadcasts are used, then the final term is P log P overhead.

**Analysis for the reduction computation**: A total of N2-1 elements must be added, plus logP communication operations. Thus $T_O = P((N^2-1)/P + \log P) - (N^2 - 1)/P = P\log P$.

$P^2$ remains the dominate term, or if broadcasts are used, $P \log P$ is the final result

**Question 2 (6 points)**

```
  for (i = 0; i < n; i++) {
S1:     a[i] = a[i-2]
S2:     b[i] = a[i+3]
  }
```

a. What kind of dependences, if any, exists between the read and write of *a* in S1? What is its direction and distance?

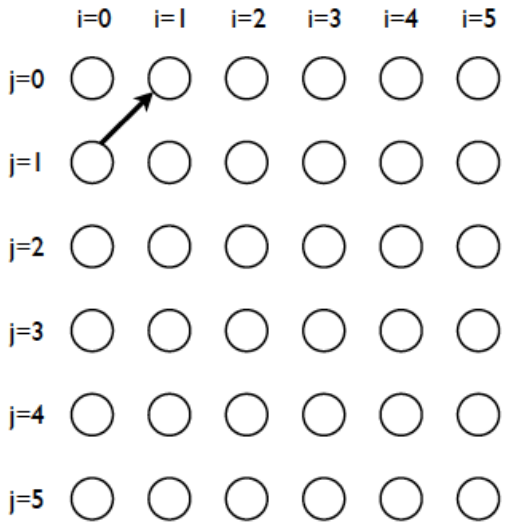
**Solution:** flow dependence from a[i] to a[i-1] with a direction of [<] or [1] and a distance of 2.

b. What kind of dependences, if any, exists between the write of *a* in S1 and the read of *a* in S2? What is its direction and distance?

**Solution:** anti dependence of a[i+3] to a[i] with a direction of [<] or [1] and a distance of 3.

**Question 3 (10 points):** Consider the loop nest:

```
for (i = 1; i < n; i++) {
   for (j = 0; j < n-1; j++) {
      a[i][j] = a[i-1][j+1]
   }
}
```




**a.** Fill in the iteration space shown to the right with at least one instance of every dependence that exists in the loop.

**Solution:** see the drawing to the right. The iteration space shown would be filled with these edges, but you were only required to show one.

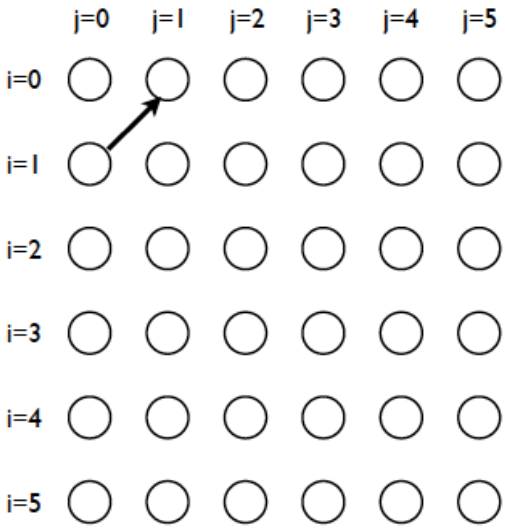
**b.** Consider the loop nest below:

```
for (j = 0; j < n-1; j++) {
   for (i = 1; i < n; i++) {
      a[i][j] = a[i-1][j+1]
   }
}
```

**c.** Fill in the iteration space shown to the right with at least one instance of every dependence that exists in the loop.

**d.**

**Solution:** see the drawing to the right. The iteration space shown would be filled with these edges, but you were only required to show one.




**e.** Does the loop in part **a** of this question give the same answer as the loop in part **b**? Say why it does, or does not.

No -- at the top data flows from iteration i=0, j=1 to iteration i=1, j=0. In the second loop the read in iteration i=1,j=2 occurs before the write in iteration i=1, j=0, that is the order of the references is reversed.

**Question 4 (14 points):** Consider the loops below:

```
DO I = 1, N
  A(I) = B(I-1) + C(I+1) // A(3) with B(2), C(4)
END DO

DO I = 1, N
  D(I) = A(I-1) + B(I-2) // A(3) with D(4)
END DO
```

**a.** Give statements that distribute A, B, C, and D to minimize communication. Use at least one TEMPLATE and at least one ALIGN statement. The arrays can be distributed onto the default 1-D processor grid.

**Solution**:

Because of the subscript patterns, not everything can be aligned. I have chosen to align the similarly colored references.

```
!HPF$ TEMPLATE T(-1:N+1)
!HPF$ ALIGN A(I) WITH T(I)
!HPF$ ALIGN B(I) WITH T(I+1)
!HPF$ ALIGN C(I) WITH T(I-1)
!HPF$ ALIGN D(I) WITH T(I-1)
!HPF$ DISTRIBUTE T(BLOCK) ONTO P
```

**b.** Is any communication necessary after performing your distribution? If so, say what kind is needed and where it is located (you can either say in words or specify what the MPI commands are.)

Yes, and it would require a shift of data from the a processor to its neighbor on the right. An MPI_SEND could be used to do this.

**Question 5 (14 points)**. Consider the loops below:

```
float a[n], b[n],c[n+1]
for (i = 1; i < n-1; i++) {
   a[i] = b[i] + c[i+1]
}

float *pa = (a-1);
for (i = 1; i < n-1; i++) {
   *pa++ = a[i-1] + b[i];
}
```

**a.** Convert the program above to a UPC program, where *a*, *b* and *c* are in shared memory. To the extent possible make *pa* point to the section of the array that is in the threads shared memory.

**Solution:**

```
shared [ceiling(N/P)] a[n];
shared [ceiling(N/P)] c[n];
shared [ceiling((N+1)/THREADS)] c[n];

upc_forall (i = 1; i < n-1; i++; ceiling(N/THREADS)] {
   a[i] = b[i] + c[i+1];
}

upc_barrier( );

float t[ceiling(N/P)];
shared float *ap = a[ceiling(P/THREADS)]
upc_forall (i = 1; i < n-1; i++; ceiling(N/THREADS)*MYTHREAD] {
   t[i] = = a[i-1] + b[i];
}

for (i = 0; i < ceiling(N/P); i++) {
   *pa++ = a[i-1] + b[i];
}
```

**b.** Is any communication necessary in your program? If there is, describe the communication.

**Solution:** yes -- depending on the number of THREADS the *c* array will not be aligned with the *a* and *b* arrays, and some accesses to c[i+1] in the first loop will need to come from the right (higher numbered) thread. Also, in the second loop, values of a[i-1] will need to come from the left (lower numbered) thread.

**Question 6 (14 points)**. Consider the loops below:

```
DO I = 1, N
  A(I) = B(I-1) + C(I+1)
END DO

DO I = 1, N
  D(I) = A(I-1) + B(I-2)
END DO
```

Write this as a Co-Array Fortran program.

**Solution:**

```
int size = N/num_images( );
int left = my_image( ) - 1
int right = my_mage( ) + 1
float A(size) [*]
float B(size)[*]
float C(size)[*]

a(1) = b(size)[left]
a(2:size-1) = b(1:size-1) + C(2:size-4)
a(size) = b(size-1) + c(1)[right]

sync_all( )

d(1) = a(size)[left] + b{size-1)[left]
d(2) = a(1) + b(size)[left]
d(3:size) = a{2:size-1) + b(1:size-2)
```

**Question 7 (14 points)**. Consider the C program below for this question.

```
Before either thread executes
for (i=0; i < n; i++) {
      a[i] = 0;
      flag[i] = 0;
}
```

**Thread 0**

```
for (i=0; i < n; i++) {
      a[i] = i;
      flag[i] = 1;
}
```

**Thread 1**

```
for (i=0; i < n; i++) {
    while (!flag[i]);
     printf("%d\n", a[i]);
}
```

**a.** Describe the output if this program only has sequentially consistent executions. If multiple outputs can exist (and they may, or may not) describe these multiple outputs. You can do this verbally.

**Solution:** prints 0, 2, ..., n-1

**b.** Describe at least one output that can occur if the program does not have sequentially consistent executions. If it is possible for this program to have a different output in a non-sequentially consistent execution than in a sequentially consistent execution, the output you give should be different than what you gave for **a**.

**Solution:** prints 0 or a value less than n, non-zeros are printed in ascending order.

**Question 8 (14 points)**. A programmer compiles a UPC program in two ways:

**1.** The program is compiled to execute using a strict consistency model (SC)
**2.** The program is compiled to execute using a weak (relaxed) consistency model that gives SC results for race free programs.

The programmer executes the program such that all possible outcomes are generated, and the results from the two versions are identical. Circle all that might be true.

**Solution:** correct answer are in **bold italics**

***a. The program is sequential***
***b. The program is race free (i.e. all shared variable accesses are synchronized)***
**c.** The program must have races (i.e. there are shared variable accesses that are not synchronized)
**d.** The program *must* be race free (i.e. all shared variable accesses are synchronized), since different access orders on variables involved in races always affect the outcome of a program. ***is ok, but since other reasons like f are possible*** must ***must is too strong.***
**e.** The hardware is sequentially consistent, and the compiler options are irrelevant in enforcing sequential consistency.
***f. The hardware is sequentially consistent and both versions the compiler performed no optimizations that might lead to a sequentially inconsistent execution.***