# ECE 563 Midterm 1
# Spring 2015

To make it easy not to cheat, this exam is open book and open notes. 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.

**Answer all questions on the answer sheet that is the last sheet of this test. You may detach the answer sheet when taking the test and you will turn it in separately. There are 15 questions in this test.**

**Name**:

**The code below is used for Questions 1 and 2. The programs are identical except for the bounds on the *j* loop**

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

  int i;
  int j;
  int n = 1000;
  float v[n], a[n][n], b[n];

  #pragma omp parallel for private(j)
  for (i=0; i < n; i++) {
    for (j=i; j < n; j++) {
       v[i] += a[i][j]*b[i];
    }
  }
}
```

**Code 1A**

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

  int i;
  int j;
  int n = 1000;
  float v[n], a[n][n], b[n];

  #pragma omp parallel for private(j)
  for (i=0; i < n; i++) {
    for (j=0; j < n; j++) {
       v[i] += a[i][j]*b[i];
    }
  }
}
```

**Code 1B**

**Question 1.** What scheduling clause should be used for Code 1A to minimize load imbalance? Do not worry about the overhead of scheduling when answering this question.

**(a)** `schedule (static)`
**(b)** `schedule (static, k)` where k is small
**(c)** `schedule (dynamic, k)`, where k is small
**(d) a** and **b**
**(e) b** and **c**

**Question 2.** What scheduling clause should be used for Code 1B that will give a good load balance and low overhead.

**(a)** `schedule (static)`
**(b)** `schedule (static, k)` where k is small
**(c)** `schedule (dynamic, k)`, where k is small
**(d) a** and **b**
**(e) b** and **c**

**The code below is used for Questions 3 and 4. The programs are identical except for the assignment statements inside the second loops.**

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

  int i;
  int j;
  int n = 1000;
  float v[n], a[n][n], b[n];

  #pragma omp parallel for private(j)
  for (i=1; i < n; i++) {
    for (j=0; j < i; j++) {
       v[i] += a[i][j]*b[i];
    }
  }

  #pragma omp parallel for private(j)
  for (i=1; i < n; i++) {
    for (j=i; j < n; j++) {
       v[i] += a[i][j]*b[i];
    }
  }
}
```

**Code 2A**

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

  int i;
  int j;
  int n = 1000;
  float v[n], a[n][n], b[n];

  #pragma omp parallel for private(j)
  for (i=0; i < n; i++) {
    for (j=0; j < i; j++) {
       v[i] += a[i][j]*b[i];
    }
  }

  #pragma omp parallel for private(j)
  for (i=1; i < n; i++) {
    for (j=i; j < n; j++) {
       v[i] = v[i-1] + a[i][j]*b[i];
    }
  }
}
```

**Code 2B**

**Question 3.** Is it desirable to use a `nowait` clause on the first loop of Code 2A?

**(a)** Yes, because the earlier iterations of the first $i$ loop have more work than later iterations of the first $i$ loop and the opposite is true of the second $i$ loop. Therefore using `nowait` will give a better load balance
**(b)** No, because even though what was said about the work in the loops is true, the `nowait` clause will not help.

**Question 4.** Regardless of the usefulness of using a `nowait` clause, is it legal to use a nowait clause in the first loop of Code 2B, i.e., will it always give the same answer as when not using the `nowait` clause?

**(a)** Yes, because OpenMP will insure all iterations in the second $i$ loop execute on the same processor as in the first loop.
**(b)** No, because even though OpenMP will insure all iterations of second $i$ loop execute on the same processor as in the first loop, elements of v read in the second loop may have been assigned in a different thread in the first loop.
**(c)** It cannot be determined since the default scheduling of OpenMP threads is unknown and therefore we don't know what threads will execute the iterations of either $i$ loop.

**The code below is used for Question 5.**

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

  int i;
  int j;
  int n = 100000;
  int t = 0;

  #pragma omp parallel for
  for (i=0; i < n; i++) {
     t = t + 1;
  }
  printf("t = %d\n", t);
}
```

**Question 5.** What statement is most true about the value of t that is printed?

**(a)** 100000 will always be printed.
**(b)** The value will be <= 100000, and values less than 100000 are possible.
**(c)** any value from -MAXINT to MAXINT can be printed

**The code below is used for Questions 6 and 7.**

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

  int i;
  int j;
  int n = 10;
  int t = 0;

  omp_set_num_threads(2);

  #pragma omp parallel private(j)
  {
     #pragma omp for
     for (i=0; i < n; i++) {
       printf("a\n");
     }

     for (j=0; j < n; j++) {
       printf("b\n");
     }
  }
}
```

**Question 6.** How may a's are printed by the program?

**(a)** 10 a's
**(b)** 20 a's
**(c)** unknown

**Question 7.** How may b's are printed by the program?

**(a)** 10 b's
**(b)** 20 b's
**(c)** unknown

**The code below is used for Questions 8 and 9.**

```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

  int i;
  int j;
  int n = 10;

  omp_set_num_threads(2);

  #pragma omp parallel num_threads(2)
  {

    #pragma omp critical
    printf("a\n");

    #pragma omp single
    printf("b\n");
  }
}
```

**Question 8.** How may a's are printed by the program?

**(a)** 10 a's
**(b)** 20 a's
**(c)** unknown

**Question 9.** How may b's are printed by the program?

**(a)** 10 b's
**(b)** 20 b's
**(c)** unknown

**The code below is used for Questions 10 and 11.**

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char* argv[]) {
   int rank, sendbuf, recvbuf, numtasks;
   MPI_Status* status;
   MPI_Request* request;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

   sendbuf = rank;

  if (rank) {
     MPI_Isend(&sendbuf, 1, MPI_INTEGER, 0, 32766, MPI_COMM_WORLD, request);
     sendbuf = 4;
     MPI_Recv(&recvbuf, 1, MPI_INTEGER, 0, MPI_ANY_TAG, MPI_COMM_WORLD, status);
  } else {
     MPI_Isend(&sendbuf, 1, MPI_INTEGER, 1, 32766, MPI_COMM_WORLD, request);
     sendbuf = 8;
     MPI_Recv(&recvbuf, 1, MPI_INTEGER, 1, MPI_ANY_TAG, MPI_COMM_WORLD, status);
  }
  printf("me: %d, them: %d\n",rank, recvbuf);
  MPI_Finalize();
}
```

**Question 10.** What is printed by process 0?

**(a)** `me: 0, them: 1`
**(b)** `me: 1 them: 0`
**(c)** `me: 0, them: 8`
**(d)** `a or c`
**(e)** `b or c`

**Question 11.** Answer with which is most correct about non-blocking sends, i.e., `MPI_Isend(...)`

**(a)** As soon as the send returns the data being sent can be modified by the sender process.
**(b)** `MPI_test` can be used to block the process until the send has completed.
**(c)** `MPI_wait` can be used to block the process until the send has completed.
**(d)** all MPI communication statements that send data ensure that the data being sent is safe to use by the sending process before returning.

**The code below is used for Questions 12.**

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char* argv[]) {
   int rank, sendbuf, recvbuf, numtasks;
   MPI_Status* status;
   MPI_Request* request;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

   sendbuf = rank;

  if (rank==0) {
     MPI_Send(&sendbuf, 1, MPI_INTEGER, 2, 0, MPI_COMM_WORLD);
     MPI_Send(&sendbuf, 1, MPI_INTEGER, 2, 10, MPI_COMM_WORLD);
  } else if (rank==1) {
     MPI_Send(&sendbuf, 1, MPI_INTEGER, 1, 1, MPI_COMM_WORLD);
  } else if (rank==2) {
     MPI_Recv(recvbuf, 1, MPI_INTEGER, 0, MPI_ANY_TAG, MPI_COMM_WORLD, status);
  }
  printf("me: %d, them: %d\n",rank, recvbuf);
  MPI_Finalize();
}
```

**Question 12.** Pick the answer that is most true when the `MPI_Recv` that is **bold and red** is executed.

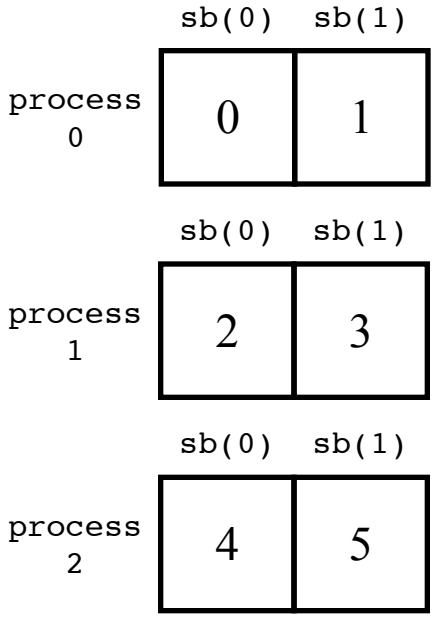
**(a)** The message with tag "0" first sent by processor 0 will be received
**(b)** The message with tag "1" sent second by processor 10 will be received
**(c)** Either the message sent by with tag "1" or tag "10" will be received but it cannot be determined beforehand which will be received.

**The code below is used for Questions 14 and 15.**

**Question 13.** Given the statement

```
MPI_Gather(sb, 2, MPI_INTEGER, rb, 2, MPI_COMM_WORLD);
```

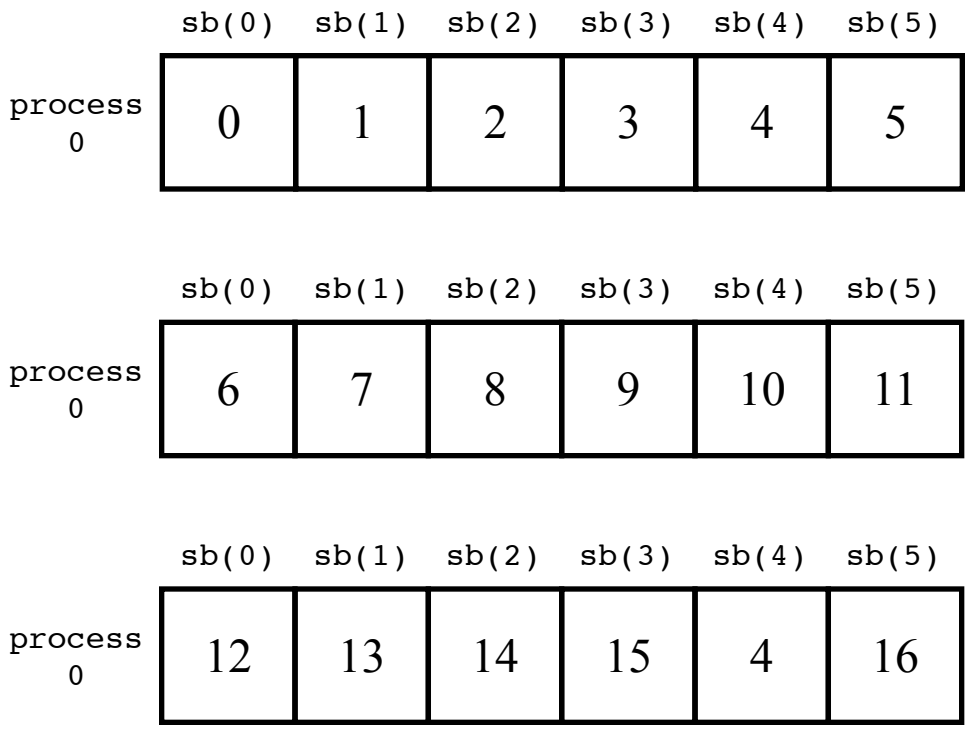
and the sb (send buffer) contents on each process as shown below, fill in the sb for each process on the answer sheet. If a rb (receive buffer) element on a process does not receive any data leave that element blank.




**Question 14.** Given the statement

```
MPI_Scatter(sb, 2, MPI_INTEGER, rb, 2, MPI_COMM_WORLD);
```

and the sb (send buffer) contents on each process as shown below, fill in the sb for each process on the answer sheet. If a rb (receive buffer) element on a process does not receive any data leave that element blank.



As the IBM manuals used to say (and may still) this page intentionally left blank.  Even though it isn't really blank.

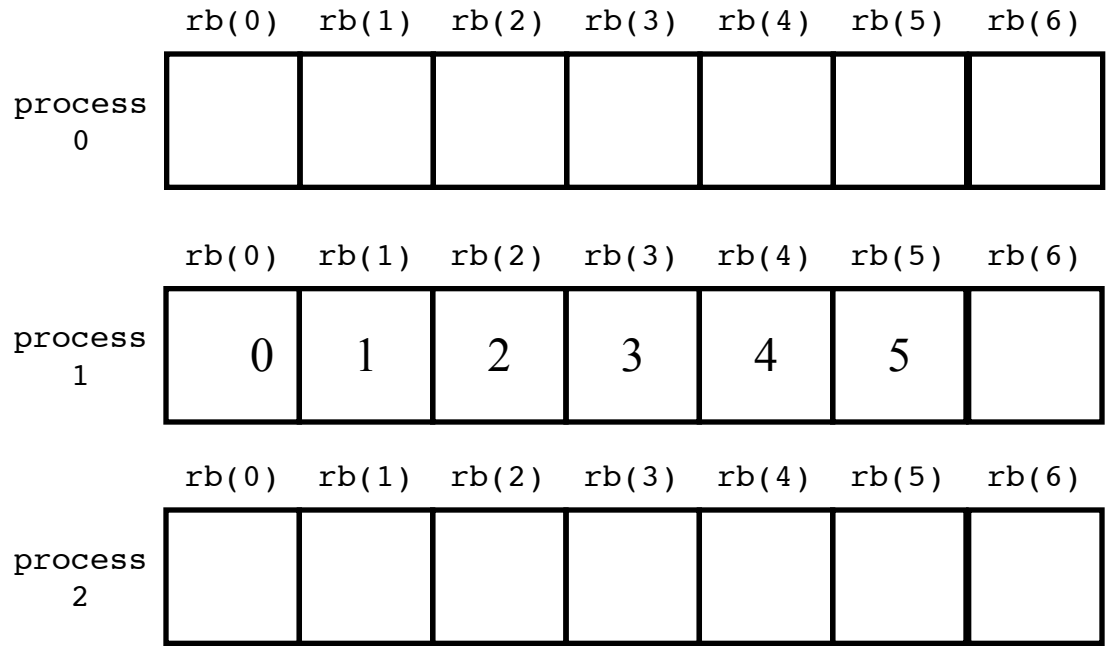
# Answer sheet

To make it easy not to cheat, this exam is open book and open notes. 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.

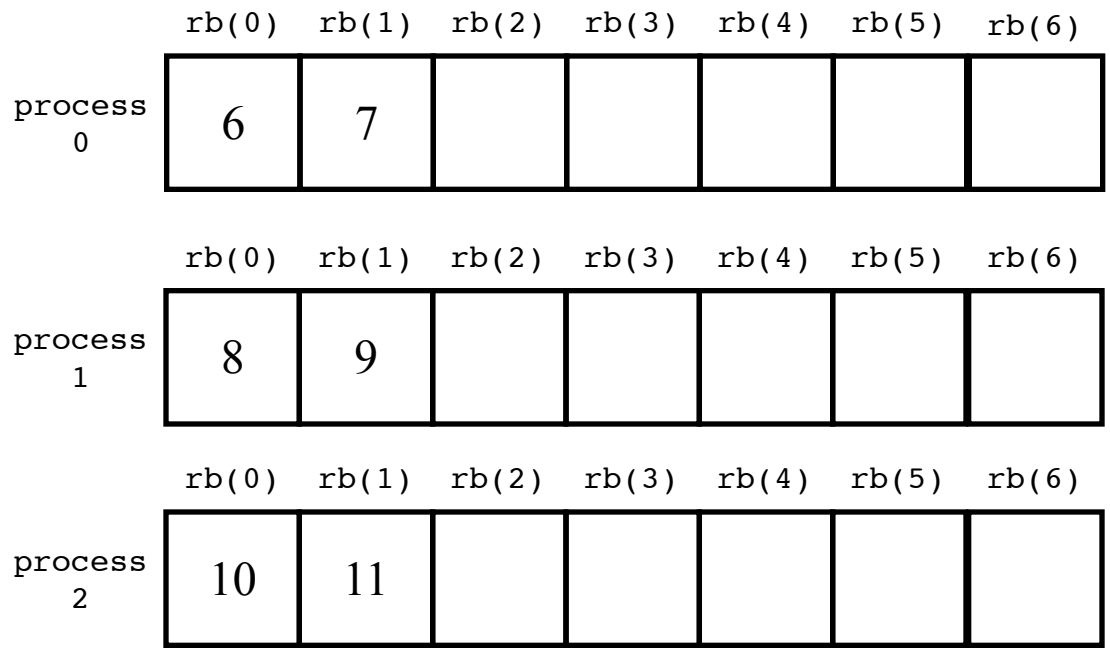
Name:

1. anything
2. a or d
3. a
4. b
5. b
6. a
7. b
8. b
9. a
10. d
11. c
12. a

13.




14.




Errors in Questions 9, 10, 13 and14 were counted as 1/2 wrong because of errors in the original questions. If they were wrong they counted as 3.5 points off. All other questions counted as 7 points off.

For 13 and 14, anything that looked like gather and a scatter received credit, regardless of values or receiving processor. Look for this to return on the next test.