

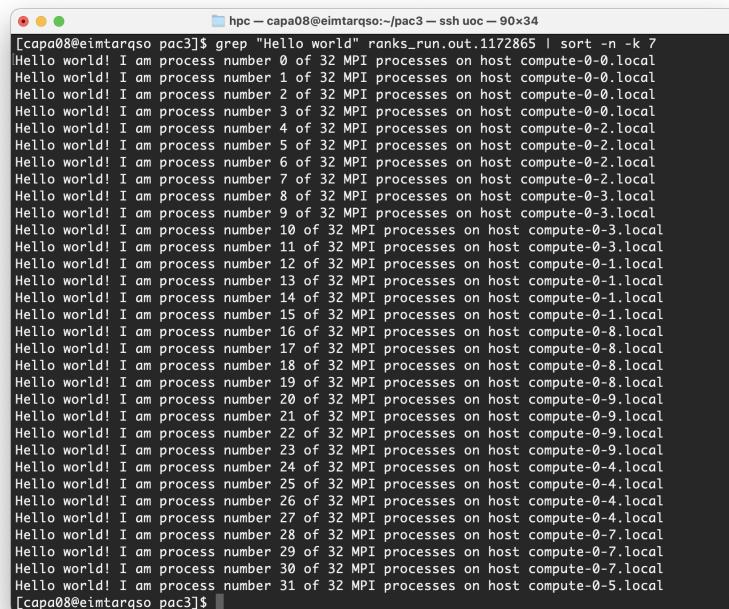
# High-performance Computing, Autumn 2024

Robert Buj

November 23, 2024

## Question 1

### 1.1



```
hpc — capa08@eimtarqso:~/pac3 — ssh uoc — 90x34
[capa08@eimtarqso pac3]$ grep "Hello world" ranks_run.out.1172865 | sort -n -k 7
Hello world! I am process number 0 of 32 MPI processes on host compute-0-0.local
Hello world! I am process number 1 of 32 MPI processes on host compute-0-0.local
Hello world! I am process number 2 of 32 MPI processes on host compute-0-0.local
Hello world! I am process number 3 of 32 MPI processes on host compute-0-0.local
Hello world! I am process number 4 of 32 MPI processes on host compute-0-2.local
Hello world! I am process number 5 of 32 MPI processes on host compute-0-2.local
Hello world! I am process number 6 of 32 MPI processes on host compute-0-2.local
Hello world! I am process number 7 of 32 MPI processes on host compute-0-2.local
Hello world! I am process number 8 of 32 MPI processes on host compute-0-3.local
Hello world! I am process number 9 of 32 MPI processes on host compute-0-3.local
Hello world! I am process number 10 of 32 MPI processes on host compute-0-3.local
Hello world! I am process number 11 of 32 MPI processes on host compute-0-3.local
Hello world! I am process number 12 of 32 MPI processes on host compute-0-1.local
Hello world! I am process number 13 of 32 MPI processes on host compute-0-1.local
Hello world! I am process number 14 of 32 MPI processes on host compute-0-1.local
Hello world! I am process number 15 of 32 MPI processes on host compute-0-1.local
Hello world! I am process number 16 of 32 MPI processes on host compute-0-8.local
Hello world! I am process number 17 of 32 MPI processes on host compute-0-8.local
Hello world! I am process number 18 of 32 MPI processes on host compute-0-8.local
Hello world! I am process number 19 of 32 MPI processes on host compute-0-8.local
Hello world! I am process number 20 of 32 MPI processes on host compute-0-9.local
Hello world! I am process number 21 of 32 MPI processes on host compute-0-9.local
Hello world! I am process number 22 of 32 MPI processes on host compute-0-9.local
Hello world! I am process number 23 of 32 MPI processes on host compute-0-9.local
Hello world! I am process number 24 of 32 MPI processes on host compute-0-4.local
Hello world! I am process number 25 of 32 MPI processes on host compute-0-4.local
Hello world! I am process number 26 of 32 MPI processes on host compute-0-4.local
Hello world! I am process number 27 of 32 MPI processes on host compute-0-4.local
Hello world! I am process number 28 of 32 MPI processes on host compute-0-7.local
Hello world! I am process number 29 of 32 MPI processes on host compute-0-7.local
Hello world! I am process number 30 of 32 MPI processes on host compute-0-7.local
Hello world! I am process number 31 of 32 MPI processes on host compute-0-5.local
[capa08@eimtarqso pac3]$
```

Figure 1: Sorted by rank

```

$ qconf -sp orte
pe_name      orte
slots        9999
user_lists   NONE
xuser_lists   NONE
start_proc_args  /bin/true
stop_proc_args  /bin/true
allocation_rule $fill_up
control_slaves  TRUE
job_is_first_task FALSE
urgency_slots   min
accounting_summary TRUE

```

Figure 2: Get the allocation rule

The allocation rule is interpreted by the scheduler thread and helps the scheduler to decide how to distribute parallel processes among the available machines.

- `$fill_up`: Starting from the best suitable host/queue, all available slots are allocated. Further hosts and queues are "filled up" as long as a job still requires slots for parallel tasks.

## 1.2

I failed to assign statically the ranks to the nodes as they are automatically assigned by SGE.

On the other hand, when using an SGE parallel environment with OpenMPI you no longer have to specify the `-np`, `-hostfile`, `-host`, etc. options to `mpirun`. This is because SGE will automatically assign hosts and processors to be used by OpenMPI for your job. You also do not need to pass the `-byslot` and `-bynode` options to `mpirun` given that these mechanisms are now handled by the `fill_up` and `round_robin` modes specified in the SGE parallel environment. REF: Sun Grid Engine (SGE) QuickStart, Submitting OpenMPI Jobs using a Parallel Environment

```
hpc — capa08@eimtarqso:~/pac3 — ssh uoc — 90x34
[capa08@eimtarqso pac3]$ grep "Hello world" ranks_run.out.1172870 | sort -n -k 7
Hello world! I am process number 0 of 32 MPI processes on host compute-0-2.local
Hello world! I am process number 1 of 32 MPI processes on host compute-0-2.local
Hello world! I am process number 2 of 32 MPI processes on host compute-0-2.local
Hello world! I am process number 3 of 32 MPI processes on host compute-0-2.local
Hello world! I am process number 4 of 32 MPI processes on host compute-0-3.local
Hello world! I am process number 5 of 32 MPI processes on host compute-0-3.local
Hello world! I am process number 6 of 32 MPI processes on host compute-0-3.local
Hello world! I am process number 7 of 32 MPI processes on host compute-0-3.local
Hello world! I am process number 8 of 32 MPI processes on host compute-0-1.local
Hello world! I am process number 9 of 32 MPI processes on host compute-0-1.local
Hello world! I am process number 10 of 32 MPI processes on host compute-0-1.local
Hello world! I am process number 11 of 32 MPI processes on host compute-0-1.local
Hello world! I am process number 12 of 32 MPI processes on host compute-0-8.local
Hello world! I am process number 13 of 32 MPI processes on host compute-0-8.local
Hello world! I am process number 14 of 32 MPI processes on host compute-0-8.local
Hello world! I am process number 15 of 32 MPI processes on host compute-0-8.local
Hello world! I am process number 16 of 32 MPI processes on host compute-0-9.local
Hello world! I am process number 17 of 32 MPI processes on host compute-0-9.local
Hello world! I am process number 18 of 32 MPI processes on host compute-0-9.local
Hello world! I am process number 19 of 32 MPI processes on host compute-0-9.local
Hello world! I am process number 20 of 32 MPI processes on host compute-0-4.local
Hello world! I am process number 21 of 32 MPI processes on host compute-0-4.local
Hello world! I am process number 22 of 32 MPI processes on host compute-0-4.local
Hello world! I am process number 23 of 32 MPI processes on host compute-0-4.local
Hello world! I am process number 24 of 32 MPI processes on host compute-0-5.local
Hello world! I am process number 25 of 32 MPI processes on host compute-0-5.local
Hello world! I am process number 26 of 32 MPI processes on host compute-0-5.local
Hello world! I am process number 27 of 32 MPI processes on host compute-0-5.local
Hello world! I am process number 28 of 32 MPI processes on host compute-0-7.local
Hello world! I am process number 29 of 32 MPI processes on host compute-0-7.local
Hello world! I am process number 30 of 32 MPI processes on host compute-0-7.local
Hello world! I am process number 31 of 32 MPI processes on host compute-0-7.local
[capa08@eimtarqso pac3]$
```

Figure 3: Sorted by rank, 2nd run

## Question 2

### 2.1

There is a deadlock because the application makes use of the schema, send and then receive the message to each other, and both are sending large messages, which are sent in synchronous mode. It means that both ranks are blocked waiting for the acknowledgement after sending the message, and for this reason they can't run the next sentence to receive the message from partner.

### 2.2

There is no deadlock, as small messages are caught and stored by the recipient into an auxiliary buffer who send the acknowledgement to the sender, so the application continues its execution on both ranks.

```
Rank 0 is reported
Rank size: 2
Rank 1 is reported
```

Figure 4: Output of deadlock execution

```
Rank 0 is reported
Rank size: 2
Rank 0 sends to 1
Rank 0 receives from 1
Rank 1 is reported
Rank 1 sends to 0
Rank 1 receives from 0
```

Figure 5: Output of deadlock execution with smaller size, SIZE=1024

I couldn't figure out how to reproduce the same behaviour with short messages. I tried to fulfill the buffer to force synchronous messages without success. See deadlock.SIZE.v2.c

```
...
#define N 100000
...
if (rank == 0) {
    int partner = 1;
    for (unsigned int i=0;i<N;i++) {
        MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
        printf("Rank %d sends to %d\n", rank, partner);
    }
}
...
} else if (rank == 1) {
    int partner = 0;
    for (unsigned int i=0;i<N;i++) {
        MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
        printf("Rank %d sends to %d\n", rank, partner);
    }
}
...
```

Figure 6: source/src/deadlock.SIZE.v2.c

## 2.3

Option 2: For one of the ranks, swap MPI\_Send and MPI\_Recv, for instance rank 1.

```
if (rank == 0) {
    int partner = 1;
    MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
    printf("Rank %d sends to %d\n", rank, partner);
    MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Rank %d receives from %d\n", rank, partner);
} else if (rank == 1) {
    int partner = 0;
    MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Rank %d receives from %d\n", rank, partner);
    MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
    printf("Rank %d sends to %d\n", rank, partner);
}
```

Figure 7: source/src/deadlock.2.c

The user numeric part modulo 3 is 2.

```
$ whoami
capa08
$ echo 8 % 3 | bc
2
```

Figure 8: get the username and option number from its numerical part

The other two options have also been implemented:

- Option 0: Non-blocking communications: source/src/deadlock.0.c
- Option 1: “Chain” sending operations (MPI Sendrecv): source/src/deadlock.1.c

## 2.4

As it's shown in the figure below, both ranks ran the same sentences in the same order, except MPI\_Send and MPI\_Recv which are swapped on rank 1 (THREAD 1.2.1).

Rank 1 started listening before rank 0 (THREAD 1.2.1) started sending the message. It doesn't matter much, because in case of node 1 wasn't ready, the message would be sent again after a timeout.

Rank 0 sent the entire message to rank 1. Rank 1 received the message and it sent the acknowledgement to rank 0, and it started to read the message.

Rank 1, after receiving the acknowledgement from node 0, started listening. Rank 0, after reading the message sent by node 0, started to send to rank 1 its own message.

Rank 0, after receiving the message from rank 0, sent an acknowledgement to rank 1. Rank 1 received the acknowledgement and started the MPI\_Finalize. Rank 0 completed the reading of the message and started the MPI\_Finalize.

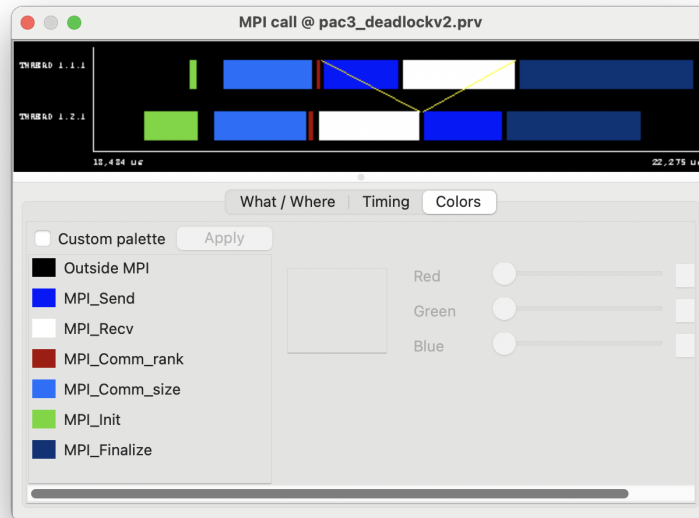
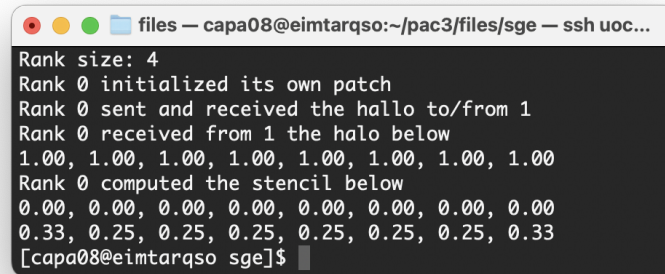


Figure 9: Trace of option 2: swap send and receive on rank 1

### Question 3

A screenshot of a terminal window with a dark background. The window title is "files — capa08@eimtarqso:~/pac3/files/sge — ssh uoc...". The output text is as follows:

```
Rank size: 4
Rank 0 initialized its own patch
Rank 0 sent and received the halo to/from 1
Rank 0 received from 1 the halo below
1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00
Rank 0 computed the stencil below
0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00
0.33, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.33
[capa08@eimtarqso sge]$
```

Figure 10: Screenshot of the execution

The user numeric part modulo 4 is 0.

```
$ whoami
capa08
$ echo 8 % 4 | bc
0
```

Figure 11: get the username and option number from its numerical part

Two approaches have been used for halo exchange. The patches at the top and at the bottom use `MPI_Sendrecv`, because they only have one adjacent patch and they only have to send and receive on halo each one.

```
/* exchange the halos */
MPI_Sendrecv(x[ROWS - 1], COLS, MPI_DOUBLE, partner, 100,
             halo, COLS, MPI_DOUBLE, partner, 100,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

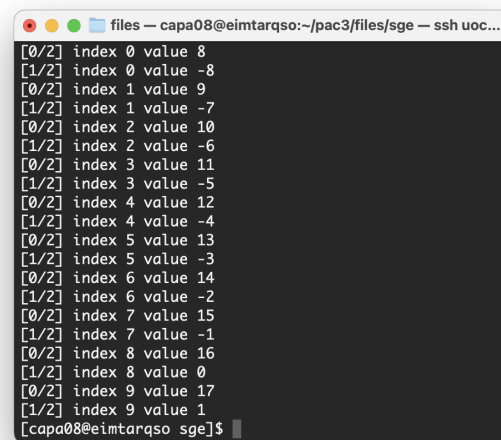
Figure 12: patch at the top

On the other hand, the other patches have two adjacent patches, and they have to send and receive two halos to/from their adjacent patches. In this case, the halos are exchanged by using non blocking sentences and they wait for the completion.

```
/* exchange the halos */
MPI_Isend(x[1], COLS, MPI_DOUBLE, partners[0], 100,
          MPI_COMM_WORLD, &requests[0]);
MPI_Isend(x[2], COLS, MPI_DOUBLE, partners[1], 100,
          MPI_COMM_WORLD, &requests[1]);
MPI_Irecv(halos[0], COLS, MPI_DOUBLE, partners[0], 100,
          MPI_COMM_WORLD, &requests[2]);
MPI_Irecv(halos[1], COLS, MPI_DOUBLE, partners[1], 100,
          MPI_COMM_WORLD, &requests[3]);
MPI_Waitall(4, requests, MPI_STATUSES_IGNORE);
```

Figure 13: inter patches

## Question 4



```
files — capa08@eimtarqso:~/pac3/files/sge — ssh uoc...
[0/2] index 0 value 8
[1/2] index 0 value -8
[0/2] index 1 value 9
[1/2] index 1 value -7
[0/2] index 2 value 10
[1/2] index 2 value -6
[0/2] index 3 value 11
[1/2] index 3 value -5
[0/2] index 4 value 12
[1/2] index 4 value -4
[0/2] index 5 value 13
[1/2] index 5 value -3
[0/2] index 6 value 14
[1/2] index 6 value -2
[0/2] index 7 value 15
[1/2] index 7 value -1
[0/2] index 8 value 16
[1/2] index 8 value 0
[0/2] index 9 value 17
[1/2] index 9 value 1
[capa08@eimtarqso sge]$
```

Figure 14: Screenshot of the execution



The numerical part of the userid is 8.

```
$ whoami  
capa08
```

Figure 15: get the numerical part from the userid

## List of commands

```
cd source
./autogen.sh
./configure --prefix=$HOME
make
make install
```

Figure 16: Get binaries from source and install them on \$HOME/bin folder which is on \$PATH environment variable

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:/share/apps/gnuplot/bin:$HOME/bin

export PATH

echo "$(uname -n) says"
cat << EOF

_ _ _ _ _
| |   | |   | |
| |__| |__| |__| |
| | \ / \ / \ / \
| | | | | | | ( ) |
| | | | \ \ \ \ \
_ _ _ _ _

EOF
```

Figure 17: Content of ~/.bash\_profile

```
cd sge; qsub ranks.sge; qsub ranks.sge
grep "Hello world" ranks_run.out.1172865 | sort -n -k 7
grep "Hello world" ranks_run.out.1172870 | sort -n -k 7
```

Figure 18: Question 1

```
cd sge
BASEDIR=$PWD
qsub mpi_send_recv.sge
cd trace/mpi_send_recv
cp /share/apps/extrac/share/example/MPI/extrac.xml ./
cp ~/bin/pac3_mpi_send_recv ./
qsub mpi_send_recv.extrac.sge
cd $BASEDIR
qsub deadlock.sge
qsub qsub deadlockv0.sge
qsub qsub deadlockv1.sge
qsub qsub deadlockv2.sge
cd $BASEDIR
cd trace/deadlockv0
cp /share/apps/extrac/share/example/MPI/extrac.xml ./
qsub deadlockv0.extrac.sge
cd $BASEDIR
cd trace/deadlockv1
cp /share/apps/extrac/share/example/MPI/extrac.xml ./
qsub deadlockv1.extrac.sge
cd $BASEDIR
cd trace/deadlockv2
cp /share/apps/extrac/share/example/MPI/extrac.xml ./
qsub deadlockv2.extrac.sge
```

Figure 19: Question 2

```
cd sge
qsub stencil.sge
```

Figure 20: Question 3

```
cd sge  
qsub rma.sge
```

Figure 21: Question 4

# File Content

## Source Files

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */
#include <unistd.h>

int main(int argc, char **argv)
{
    int rank, numprocs;
    char hostname[256];

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    gethostname(hostname,255);
    printf("Hello world! I am process number %d of %d MPI processes on host %s\n",
    rank, numprocs, hostname);
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

Figure 22: source/src/ranks.c

```

#include "mpi.h"
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */

int main(int argc, char* argv[]) {
    int MyProc, tag=1;
    char msg='A', msg_recpt;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &MyProc);

    printf("Process # %d started \n", MyProc);
    MPI_Barrier(MPI_COMM_WORLD);

    if (MyProc == 0) {
        printf("Sending message to Proc #1 \n") ;
        MPI_Send(&msg, 1, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
        MPI_Recv(&msg_recpt, 1, MPI_CHAR, 1, tag, MPI_COMM_WORLD, &status);
        printf("Recv'd message from Proc #1 \n") ;
    } else {
        MPI_Recv(&msg_recpt, 1, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
        printf("Recv'd message from Proc #0 \n") ;
        printf("Sending message to Proc #0 \n") ;
        MPI_Send(&msg, 1, MPI_CHAR, 0, tag, MPI_COMM_WORLD);
    }
    printf("Finishing proc %d\n", MyProc);
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
    return EXIT_SUCCESS;
}

```

Figure 23: source/src/mpi\_send\_recv.c

```

#include "mpi.h" /* MPI_Init, MPI_Finalize, MPI_Send, MPI_Recv,
MPI_Comm_size, MPI_Comm_rank, MPI_COMM_WORLD, MPI_STATUS_IGNORE */
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */
#define SIZE 100000

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

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* Get my rank */
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Rank %d is reported\n", rank);
    if (rank == 0) printf("Rank size: %d\n", size);
    int v1[SIZE];
    int v2[SIZE];
    if (rank == 0) {
        int partner = 1;
        MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
        printf("Rank %d sends to %d\n", rank, partner);
        MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
        printf("Rank %d receives from %d\n", rank, partner);
    } else if (rank == 1) {
        int partner = 0;
        MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
        printf("Rank %d sends to %d\n", rank, partner);
        MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
        printf("Rank %d receives from %d\n", rank, partner);
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}

```

Figure 24: source/src/deadlock.c

```

#include <mpi.h>
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */

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

    int size, rank;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Rank %d is reported\n", rank);
    if (rank == 0) printf("Rank size: %d\n", size);
    int v1[SIZE];
    int v2[SIZE];
    if (rank == 0) {
        int partner = 1;
        MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
        printf("Rank %d sends to %d\n", rank, partner);
        MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("Rank %d receives from %d\n", rank, partner);
    } else if (rank == 1) {
        int partner = 0;
        MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
        printf("Rank %d sends to %d\n", rank, partner);
        MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("Rank %d receives from %d\n", rank, partner);
    }
    MPI_Finalize();
    return 0;
}

```

Figure 25: source/src/deadlock.SIZE.c



```

#include <mpi.h>
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */
#define N 100000

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

    int size, rank;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Rank %d is reported\n", rank);
    if (rank == 0) printf("Rank size: %d\n", size);
    int v1[SIZE];
    int v2[SIZE];
    if (rank == 0) {
        int partner = 1;
        for (unsigned int i=0;i<N;i++) {
            MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
            printf("Rank %d sends to %d\n", rank, partner);
        }
        for (unsigned int i=0;i<N;i++) {
            MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD,
                    MPI_STATUS_IGNORE);
            printf("Rank %d receives from %d\n", rank, partner);
        }
    } else if (rank == 1) {
        int partner = 0;
        for (unsigned int i=0;i<N;i++) {
            MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
            printf("Rank %d sends to %d\n", rank, partner);
        }
        for (unsigned int i=0;i<N;i++) {
            MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD,
                    MPI_STATUS_IGNORE);
            printf("Rank %d receives from %d\n", rank, partner);
        }
    }
    MPI_Finalize();
    return 0;
}

```

Figure 26: source/src/deadlock.SIZE.v2.c

```

#include "mpi.h" /* MPI_Init, MPI_Finalize, MPI_Isend, MPI_Irecv,
                 MPI_Waitall, MPI_Comm_size, MPI_Comm_rank,
                 MPI_Request, MPI_COMM_WORLD, MPI_STATUSES_IGNORE */
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */
#define SIZE 100000

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

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* Get my rank */
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Rank %d is reported\n", rank);
    if (rank == 0) printf("Rank size: %d\n", size);

    int v1[SIZE];
    int v2[SIZE];
    MPI_Request requests[2];
    if (rank == 0) {
        int partner = 1;
        MPI_Isend(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD, &requests[0]);
        MPI_Irecv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD, &requests[1]);
        MPI_Waitall(2, requests, MPI_STATUSES_IGNORE);
        printf("Rank %d sends to %d\n", rank, partner);
        printf("Rank %d receives from %d\n", rank, partner);
    } else if (rank == 1) {
        int partner = 0;
        MPI_Isend(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD, &requests[0]);
        MPI_Irecv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD, &requests[1]);
        MPI_Waitall(2, requests, MPI_STATUSES_IGNORE);
        printf("Rank %d sends to %d\n", rank, partner);
        printf("Rank %d receives from %d\n", rank, partner);
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}

```

Figure 27: source/src/deadlock.0.c

```

#include "mpi.h" /* MPI_Init, MPI_Finalize, MPI_Sendrecv, MPI_Comm_size,
MPI_Comm_rank, MPI_COMM_WORLD, MPI_STATUS_IGNORE */
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */
#define SIZE 100000

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

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* Get my rank */
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Rank %d is reported\n", rank);
    if (rank == 0) printf("Rank size: %d\n", size);

    int v1[SIZE];
    int v2[SIZE];
    if (rank == 0) {
        int partner = 1;
        MPI_Sendrecv(v1, SIZE, MPI_INT, partner, 100,
                     v2, SIZE, MPI_INT, partner, 100,
                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Rank %d sends to %d\n", rank, partner);
        printf("Rank %d receives from %d\n", rank, partner);
    } else if (rank == 1) {
        int partner = 0;
        MPI_Sendrecv(v1, SIZE, MPI_INT, partner, 100,
                     v2, SIZE, MPI_INT, partner, 100,
                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Rank %d sends to %d\n", rank, partner);
        printf("Rank %d receives from %d\n", rank, partner);
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}

```

Figure 28: source/src/deadlock.1.c

```

#include "mpi.h" /* MPI_Init, MPI_Finalize, MPI_Send, MPI_Recv,
                MPI_Comm_size, MPI_Comm_rank */
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */
#define SIZE 100000

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

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* Get my rank */
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Rank %d is reported\n", rank);
    if (rank == 0) printf("Rank size: %d\n", size);

    int v1[SIZE];
    int v2[SIZE];
    if (rank == 0) {
        int partner = 1;
        MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
        printf("Rank %d sends to %d\n", rank, partner);
        MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
        printf("Rank %d receives from %d\n", rank, partner);
    } else if (rank == 1) {
        int partner = 0;
        MPI_Recv(v2, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
        printf("Rank %d receives from %d\n", rank, partner);
        MPI_Send(v1, SIZE, MPI_INT, partner, 100, MPI_COMM_WORLD);
        printf("Rank %d sends to %d\n", rank, partner);
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}

```

Figure 29: source/src/deadlock.2.c

```

#include "mpi.h" /* MPI_Init, MPI_Finalize, MPI_Sendrecv,
                 MPI_Isend, MPI_Irecv, MPI_Waitall, MPI_Request
                 MPI_Comm_size, MPI_Comm_rank, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE, MPI_STATUSES_IGNORE,
                 MPI_DOUBLE */

#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */
#define SIZE 8
#define COLS SIZE
#define ROWS 2

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

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        double x[ROWS + 1][COLS], x_new[ROWS][COLS], halo[COLS];
        int partner = rank + 1;
        unsigned int i, j;

        printf("Rank size: %d\n", size);
        /* patch initialization */
        for (i = 0; i < COLS; i++)
            x[ROWS][i] = -1.0;
        for (i = 0; i < ROWS; i++)
            for (j = 0; j < COLS; j++)
                x[i][j] = (double)rank;
        printf("Rank %d initialized its own patch\n", rank);
        /* exchange the halos */
        MPI_Sendrecv(x[ROWS - 1], COLS, MPI_DOUBLE, partner, 100,
                    halo, COLS, MPI_DOUBLE, partner, 100,
                    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Rank %d sent and received the halo to/from %d\n",
                rank, partner);
        for (i = 0; i < COLS; i++)
            x[ROWS][i] = halo[i];
        /* print the received halo */
        printf("Rank %d received from %d the halo below\n",
                rank, partner);
        for (j = 0; j < COLS; j++) {
            if (j == COLS - 1)

```

```

        printf("%.2f\n", halo[j]);
    else
        printf("%.2f, ", halo[j]);
}
/* stencil computation of the 1st row */
i = 0;
j = 0;
x_new[i][j] = (x[i][j + 1] + x[i + 1][j]) / 2.0;
for (j = 1; j < COLS - 1; j++)
    x_new[i][j] = (x[i][j + 1] + x[i][j - 1]
                  + x[i + 1][j]) / 3.0;
x_new[i][j] = (x[i][j - 1] + x[i + 1][j]) / 2.0;
/* stencil computation of the 2nd row */
i = 1;
j = 0;
x_new[i][j] = (x[i][j + 1] + x[i + 1][j] + x[i - 1][j]) / 3.0;
for (j = 1; j < COLS - 1; j++)
    x_new[i][j] = (x[i][j + 1] + x[i][j - 1] + x[i + 1][j]
                  + x[i - 1][j]) / 4.0;
x_new[i][j] = (x[i][j - 1] + x[i + 1][j] + x[i - 1][j]) / 3.0;
printf("Rank %d computed the stencil below\n", rank);
/* print the block result */
for (i = 0; i < ROWS; i++) {
    for (j = 0; j < COLS; j++) {
        if (j == COLS - 1)
            printf("%.2f\n", x_new[i][j]);
        else
            printf("%.2f, ", x_new[i][j]);
    }
}
} else if (rank < size - 1) {
    double x[ROWS + 2][COLS], x_new[ROWS][COLS], halos[2][COLS];
    MPI_Request requests[4];
    int partners[2] = {rank - 1, rank + 1};
    unsigned int i, j;
    /* patch initialization */
    for (i = 0; i < COLS; i++)
        x[0][i] = x[ROWS + 1][i] = -1.0;
    for (i = 1; i < ROWS; i++)
        for (unsigned int j = 0; j < COLS; j++)
            x[i][j] = (double)rank;
    /* exchange the halos */
    MPI_Isend(x[1], COLS, MPI_DOUBLE, partners[0], 100,
              MPI_COMM_WORLD, &requests[0]);
    MPI_Isend(x[2], COLS, MPI_DOUBLE, partners[1], 100,

```

```

MPI_COMM_WORLD, &requests[1]);
MPI_Irecv(halos[0], COLS, MPI_DOUBLE, partners[0], 100,
MPI_COMM_WORLD, &requests[2]);
MPI_Irecv(halos[1], COLS, MPI_DOUBLE, partners[1], 100,
MPI_COMM_WORLD, &requests[3]);
MPI_Waitall(4, requests, MPI_STATUSES_IGNORE);
for (i = 0; i < COLS; i++) {
    x[0][i] = halos[0][i];
    x[ROWS + 1][i] = halos[1][i];
}
/* stencil computation of the two rows */
for (i = 1; i < i + ROWS; i++) {
    j = 0;
    x_new[i][j] = (x[i][j + 1] + x[i + 1][j] + x[i - 1][j]) / 3.0;
    for (j = 1; j < COLS - 1; j++)
        x_new[i][j] = (x[i][j + 1] + x[i][j - 1] + x[i + 1][j]
                        + x[i - 1][j]) / 4.0;
    x_new[i][j] = (x[i][j - 1] + x[i + 1][j] + x[i - 1][j]) / 3.0;
}
} else {
    double x[ROWS + 1][COLS], x_new[ROWS][COLS], halo[COLS];
    int partner = rank - 1;
    unsigned int i, j;
    /* patch initialization */
    for (i = 0; i < COLS; i++)
        x[0][i] = -1.0;
    for (i = 0; i < ROWS; i++)
        for (j = 0; j < COLS; j++)
            x[i][j] = (double)rank;
    /* exchange the halos */
    MPI_Sendrecv(x[1], COLS, MPI_DOUBLE, partner, 100,
                 halo, COLS, MPI_DOUBLE, partner, 100,
                 MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    for (i = 0; i < COLS; i++)
        x[0][i] = halo[i];
    /* stencil computation of the 1st row */
    i = 1;
    j = 0;
    x_new[i][j] = (x[i][j + 1] + x[i + 1][j] + x[i - 1][j]) / 3.0;
    for (j = 1; j < COLS - 1; j++)
        x_new[i][j] = (x[i][j + 1] + x[i][j - 1] + x[i + 1][j]
                        + x[i - 1][j]) / 4.0;
    x_new[i][j] = (x[i][j - 1] + x[i + 1][j] + x[i - 1][j]) / 3.0;
    /* stencil computation of the 2nd row */
    i = 2;

```

```

    j = 0;
    x_new[i][j] = (x[i][j + 1] + x[i - 1][j]) / 2.0;
    for (j = 1; j < COLS - 1; j++)
        x_new[i][j] = (x[i][j + 1] + x[i][j - 1] + x[i - 1][j]) / 3.0;
    x_new[i][j] = (x[i][j - 1] + x[i - 1][j]) / 2.0;
}
MPI_Finalize();
return EXIT_SUCCESS;
}

```

Figure 30: source/src/stencil.c

```

#include "mpi.h"
#include <stdio.h>
#include <stdlib.h> /* EXIT_SUCCESS */

#define SIZE 10
#define USERID 8

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

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* Get my rank */
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int v1[SIZE];
    for (int i = 0; i < SIZE; i++) {
        if (rank == 0)
            v1[i] = USERID + i;
        else if (rank == 1)
            v1[i] = (USERID - i) * (-1);
    }

    /* <<< here your RMA code >>> */
    /* create the window */
    MPI_Win win;
    MPI_Win_create(&v1, /* pre-allocated buffer */
        SIZE * sizeof(int), /* size in bytes */

```



```

        sizeof(int), /* displacement units */
        MPI_INFO_NULL, /* info object */
        MPI_COMM_WORLD, /* communicator */
        &win /* window object */);

/* start access epoch */
MPI_Win_fence(0, win);

int remote_v1[SIZE];
if (rank == 0){
    MPI_Get(&remote_v1, /* pre-allocated buffer on RMA origin process */
           SIZE, /* count on RMA origin process */
           MPI_INT, /* type on RMA origin process */
           1, /* rank of RMA target process */
           0, /* displacement on RMA target process */
           SIZE, /* count on RMA target process */
           MPI_INT, /* type on RMA target process */
           win /* window object */);
}

/* end access epoch */
MPI_Win_fence(0, win);

if (rank == 0){
    for (int i = 0; i < SIZE; i++) {
        printf("[0/%d] index %d value %d\n", size, i, v1[i]);
        printf("[1/%d] index %d value %d\n", size, i, remote_v1[i]);
    }
}

MPI_Win_free(&win);
MPI_Finalize();

return EXIT_SUCCESS;
}

```

Figure 31: source/src/rma.c

## Build System

```
AC_INIT([M1.209-pac3],[0.1],[rbuj@uoc.edu])
AC_COPYRIGHT([Copyright (C) 2024 Robert Buj])
AM_INIT_AUTOMAKE([foreign])
AM_SILENT_RULES([yes])
AC_PROG_CC([mpicc])
AC_PROG_CC_STDC
AC_CONFIG_FILES([
Makefile
src/Makefile
])
AC_OUTPUT

echo "
Configure summary:

${PACKAGE_STRING}
`echo $PACKAGE_STRING | sed "s/./=/g"`
prefix:                ${prefix}
compiler:               ${CC}
cflags:                 ${CFLAGS}
"
```

Figure 32: source/configure.ac

```
#!/bin/sh
rm -rf autom4te.cache
rm -f aclocal.m4 ltmain.sh

touch README

echo "Running aclocal..." ; aclocal $ACLOCAL_FLAGS || exit 1
echo "Running autoconf..." ; autoconf || exit 1
echo "Running automake..." ; automake --add-missing --copy --gnu || exit 1
```

Figure 33: source/autogen.sh

```
SUBDIRS = src
```

Figure 34: source/Makefile.am

```
bin_PROGRAMS = \  
pac3_deadlock pac3_deadlock_1024 pac3_deadlock2_1024\  
pac3_deadlockv0 pac3_deadlockv1 pac3_deadlockv2 \  
pac3_hello \  
pac3_mpi_send_recv \  
pac3_ranks \  
pac3_rma \  
pac3_stencil  
  
# AM_CFLAGS = -Wall -Wextra -Werror  
AM_CFLAGS = -Wall -Wextra  
  
pac3_deadlock_SOURCES = deadlock.c  
pac3_deadlock_1024_SOURCES = deadlock.SIZE.c  
pac3_deadlock_1024_CPPFLAGS = -D SIZE=1024  
pac3_deadlock2_1024_SOURCES = deadlock.SIZE.v2.c  
pac3_deadlock2_1024_CPPFLAGS = -D SIZE=1024  
pac3_deadlockv0_SOURCES = deadlock.0.c  
pac3_deadlockv1_SOURCES = deadlock.1.c  
pac3_deadlockv2_SOURCES = deadlock.2.c  
pac3_hello_SOURCES = hello.c  
pac3_mpi_send_recv_SOURCES = mpi_send_recv.c  
pac3_ranks_SOURCES = ranks.c  
pac3_rma_SOURCES = rma.c  
pac3_stencil_SOURCES = stencil.c
```

Figure 35: source/src/Makefile.am

## SGE Scripts

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N ranks_run
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 32

mpirun -np 32 pac3_ranks
```

Figure 36: sge/ranks.sge

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N mpi_send_recv_run
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 2

mpirun -np 2 pac3_mpi_send_recv
```

Figure 37: sge/mpi\_send\_recv.sge

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N deadlock_run
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 2

mpirun -np 2 pac3_deadlock
```

Figure 38: sge/deadlock.sge

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N deadlock_1024_run
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 2

mpirun -np 2 pac3_deadlock_1024
```

Figure 39: sge/deadlock\_1024.sge

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N deadlock2_1024_run
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 2

mpirun -np 2 pac3_deadlock2_1024
```

Figure 40: sge/deadlock2\_1024.sge

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N deadlockv0_run
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 2

mpirun -np 2 pac3_deadlockv0
```

Figure 41: sge/deadlockv0.sge

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N deadlockv1_run
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 2

mpirun -np 2 pac3_deadlockv1
```

Figure 42: sge/deadlockv1.sge

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N deadlockv2_run
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 2

mpirun -np 2 pac3_deadlockv2
```

Figure 43: sge/deadlockb2.sge

```
#!/bin/bash
#£ -cwd
#£ -S /bin/bash
#£ -N deadlock_run.extrae
#£ -o £JOB_NAME.out.£JOB_ID
#£ -e £JOB_NAME.err.£JOB_ID
#£ -pe orte 2

export EXTRAE_HOME=/share/apps/extrae
source /share/apps/extrae/etc/extrae.sh
export EXTRAE_CONFIG_FILE=extrae.xml
export LD_PRELOAD=/share/apps/extrae/lib/libmpitrace.so
mpirun -np 2 pac3_deadlockv0
```

Figure 44: sge/trace/deadlockv0/deadlock.extrae.sge

```

#!/bin/bash
#L -cwd
#L -S /bin/bash
#L -N deadlock_run.extrae
#L -o $JOB_NAME.out.$JOB_ID
#L -e $JOB_NAME.err.$JOB_ID
#L -pe orte 2

export EXTRAE_HOME=/share/apps/extrae
source /share/apps/extrae/etc/extrae.sh
export EXTRAE_CONFIG_FILE=extrae.xml
export LD_PRELOAD=/share/apps/extrae/lib/libmpitrace.so
mpirun -np 2 pac3_deadlockv1

```

Figure 45: sge/trace/deadlockv1/deadlock.extrae.sge

```

#!/bin/bash
#L -cwd
#L -S /bin/bash
#L -N deadlock_run.extrae
#L -o $JOB_NAME.out.$JOB_ID
#L -e $JOB_NAME.err.$JOB_ID
#L -pe orte 2

export EXTRAE_HOME=/share/apps/extrae
source /share/apps/extrae/etc/extrae.sh
export EXTRAE_CONFIG_FILE=extrae.xml
export LD_PRELOAD=/share/apps/extrae/lib/libmpitrace.so
mpirun -np 2 pac3_deadlockv2

```

Figure 46: sge/trace/deadlockv2/deadlock.extrae.sge

```
#!/bin/bash
#L -cwd
#L -S /bin/bash
#L -N stencil_run
#L -o $JOB_NAME.out.$JOB_ID
#L -e $JOB_NAME.err.$JOB_ID
#L -pe orte 4

mpirun -np 4 pac3_stencil
```

Figure 47: sge/stencil.sge

```
#!/bin/bash
#L -cwd
#L -S /bin/bash
#L -N rma_run
#L -o $JOB_NAME.out.$JOB_ID
#L -e $JOB_NAME.err.$JOB_ID
#L -pe orte 2

mpirun -np 2 pac3_rma
```

Figure 48: sge/rma.sge