Lab 3:

Parallel Smith-Waterman with MPI

Jan Izquierdo
jan.izquierdo@
hpc1211

Laura Llorente
laura.llorente@
hpc1214

12/04/2024

Modified SW_mpi.c:

67:
```
#include <mpi.h>    // Use MPI ///ADDED mpi module
```
Here we include the mpi module to the program.

74:
```
#define min(a,b) (((a)<(b) ? (a) : (b))     /* Complete the value
returned when the condition evaluates to false.*/  ///if a is smaller
return a, else return b (b is smaller)
```
We define min to return the smallest value of the 2 inputs.

165:
```
 if ( rank==0 ) { /* Use process 0 as the master process
      and have it process the input */ ///check that rank is 0
  /**** Error handling for input file ****/
```

285-291:
```
 MPI_Bcast( &dim1, 1, MPI_INT, 0, MPI_COMM_WORLD); /* Broadcast dim1 */
///array, length of array, contents of array, root, comm
 MPI_Bcast( &dim2, 1, MPI_INT, 0, MPI_COMM_WORLD); /* Broadcast dim2 */
///
 MPI_Bcast( &BS, 1, MPI_INT, 0, MPI_COMM_WORLD); /* Broadcast BS */ ///
 MPI_Bcast( &DELTA, 1, MPI_INT, 0, MPI_COMM_WORLD); /* Broadcast DELTA
*/ ///
 MPI_Bcast( sim, AA*AA, MPI_INT, 0, MPI_COMM_WORLD ); /* Broadcast
matrix sim */ /// AA is size of matrix

 nrows = getRowCount(dim1 , rank, nprocs); ///getRowCount(int
rowsTotal, int mpiRank, int mpiSize) / rowtotal is dim1, because dim1
is the length of the 1rst sequence
```
We broadcast these variables to all processes from root, and calculate the number of rows per process in case of not being able to distribute the rows among processes evenly.

295:
```
  CHECK_NULL ((s11 = (short *) malloc ((short) * (dim1+1))));///create
array s11 the size of dim1+1(real seq length), as it will contain the
first sequence (dim1 can be exchanged by nrows) /clues from below
```
Create an array of the appropriate size to store the first sequence.

303:
```
 //MPI_Bcast( s1+1, dim1, MPI_SHORT, 0, MPI_COMM_WORLD );
 MPI_Scatter( s1+1, nrows, MPI_SHORT, s11+1, nrows, MPI_SHORT, 0,
MPI_COMM_WORLD); ///send buf, count, type, recv buf, count, type, root
(data origin), comm /copy s1 to s11
 /* s1+1: use pointer arithmetic to point to the second element in the
array
  *       since the 1st position is not used.
  *        _____________________      ____
  *       |       |      |                 |
  *       |s1[0]  |s1[1]|        ....      |
  *       |_______|______|_____       ____|
  *        ^       ^ ....
  *        |       |
  *        s1     s1+1
  *
  *       */
```
This line divides and sends the contents of s1 into s11 across all the processes.

315:

```
 MPI_Bcast( s2+1, dim2, MPI_SHORT, 0, MPI_COMM_WORLD); /// broadcasts
contents of array s2 skipping first position and distributes to all
processes (MPI_COMM_WORLD)
```
This line is used to broadcast s2 to all processes.

326:

```
 dim1local = (rank ? nrows : dim1); /* condition ? true_case :
false_case */ ///if rank!=0 dim1local=nrows(local row count), else:
dim1local= length of 1rst sequence(dim1)
```
Here we define dim1local to be nrows if the process is 0, dim1local will be dim1, else, the process will be nrows.

378-383:

```
 /* PARALLEL CODE */
 for (int jj = 1; jj <= dim2; jj += BS) {    /* Strip mining: Define
blocks of size BS in the columns */ ///jj+=BS because jumps of size BS

  if (rank != 0) MPI_Recv( &h[0][jj], min(BS, dim2-jj+1), MPI_INT,
rank-1, 0, MPI_COMM_WORLD, &status); /// we receive data from previous
process and store in ith matrix row → Size of the buffer = size of
smallest between BS or dim2-jj+1. Status is a predefined variable
(MPI_Status, line 153) /tag can be 1 instead of 0

  for (i = 1; i <= nrows ; i++) /// iterate in nrows range
```

```
   for (j = jj; j < min(jj+BS,dim2+1); j++) /* Strip mining: Traverse
BS columns within the current block */ /// column index beyond current
block → dim2 as limit of columns (avoid) /min of (next for loop jump
(jj loop) vs max size+1), ensures that we are not out of range
```

Here we receive the data sent by the previous process and use it to create the scoring matrix in the following lines

421:

```
  if (rank != nprocs-1 ) MPI_Send(&h[nrows][jj], min(BS, dim2-jj+1),
MPI_INT, rank + 1, 0, MPI_COMM_WORLD);
  /* Here we could use a non-blocking send. */ ///Tag could be 1 as
well
```

We check that we are not in the last process, if that is true, send the &h matrix on row [nrows] and column [jj] to the next process in order.

If we were to use a non-blocking send (nb_SW_mpi.c):

```
if (rank != nprocs-1 ) {
   MPI_Request request;
   MPI_Isend(&h[nrows][jj], min(BS, dim2-jj+1), MPI_INT, rank + 1, 0,
MPI_COMM_WORLD, &request);///every row is BS length, when last row, BS
is too long (you are already in the last BS range)
   MPI_Wait(&request, MPI_STATUS_IGNORE);
}
}
/* Here we could use a non-blocking send. */ ///MPI_Isend+MPI_wait+
request → MPI_Send data and count??
```

In this case we would use MPI_Isend and MPI_Wait to avoid blocking the process

432:

```
   MPI_Allreduce(localres, globalres, 1, MPI_2INT, MPI_MAXLOC,
MPI_COMM_WORLD); /// we store the output in globalres array and use
MAXLOC operation to find max value
```

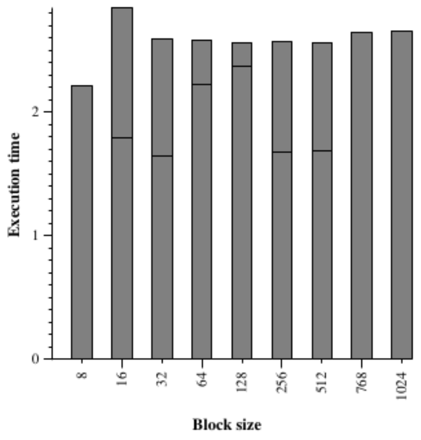
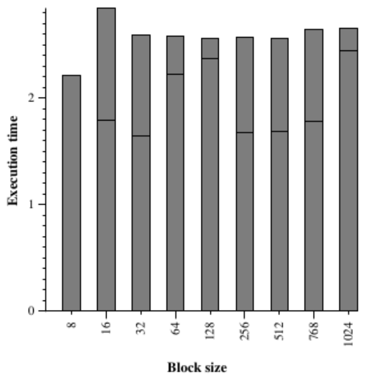
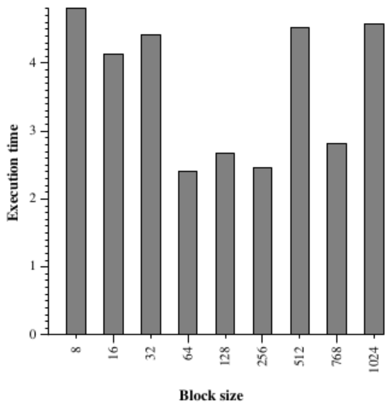
Allreduce is used to gather elements across all processes, then compute the maximum value across all processes and identify the process that contains it.

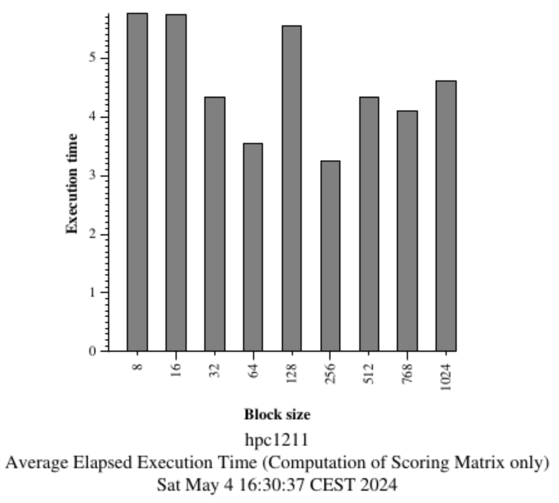
508:

```
 if (rank==0) free(s1); /* Only the master process allocated it */
///master process→ rank 0, if rank is 0, end
```

This line is used to end the program and free the memory when the last process (0) is reached.

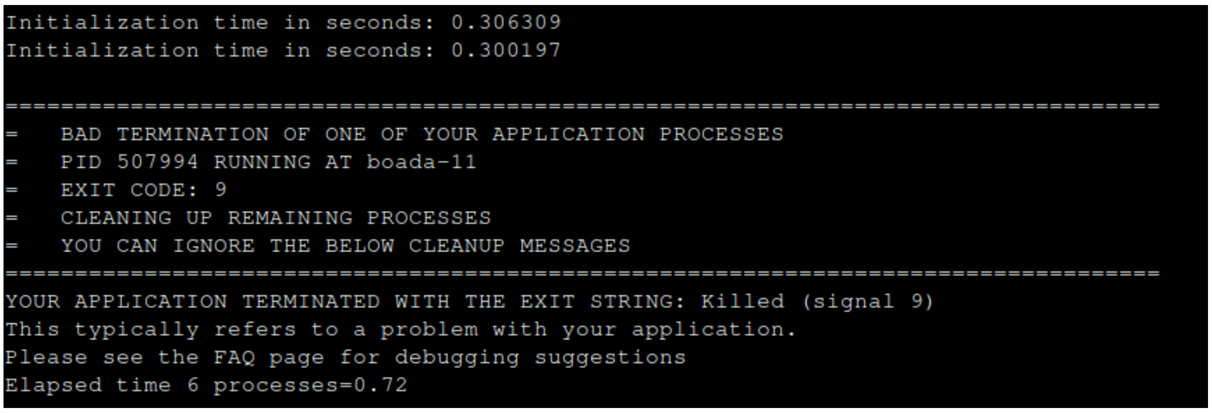
Blocksize SW_mpi.c:

5 MPI processes	 hpc1211 Average Elapsed Execution Time (Computation of Scoring Matrix only) Sat May 4 16:18:58 CEST 2024
10 MPI processes	 hpc1211 Average Elapsed Execution Time (Computation of Scoring Matrix only) Sat May 4 16:19:03 CEST 2024
20 MPI processes	 hpc1211 Average Elapsed Execution Time (Computation of Scoring Matrix only) Sat May 4 16:11:28 CEST 2024

30 MPI processes	

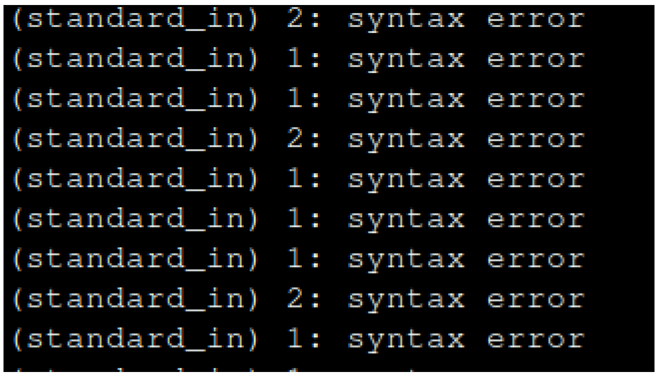
You can observe that the execution time increases with the more processors you have.
Also it can be observed that the blocking size with smaller execution time overall is block size 256, which makes it the optimal value as a blocking factor.

Strong SW_mpi.c:
There appears to be an error when executing this program.



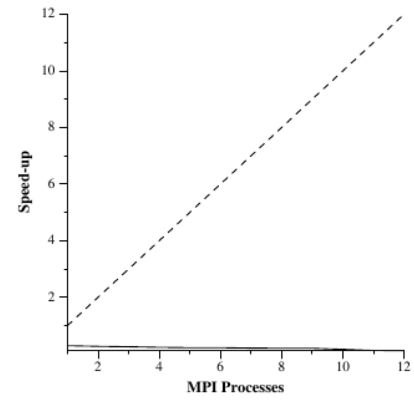
```
Initialization time in seconds: 0.306309
Initialization time in seconds: 0.300197

===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   PID 507994 RUNNING AT boada-11
=   EXIT CODE: 9
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Killed (signal 9)
This typically refers to a problem with your application.
Please see the FAQ page for debugging suggestions
Elapsed time 6 processes=0.72
```

According to the error exit of the sbatch submission it appears to be a syntax error.

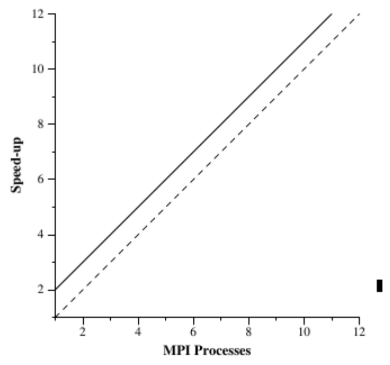


```
(standard_in) 2: syntax error
(standard_in) 1: syntax error
(standard_in) 1: syntax error
(standard_in) 2: syntax error
(standard_in) 1: syntax error
(standard_in) 1: syntax error
(standard_in) 1: syntax error
(standard_in) 2: syntax error
(standard_in) 1: syntax error
```

Even with this errors the code created the following 2 plots



hpc1211
Speed-up wrt sequential time (Complete Application)
Sat May 4 22:23:00 CEST 2024



hpc1211
Speed-up wrt sequential time (Computation of Scoring Matrix only)
Sat May 4 22:23:00 CEST 2024

The graph belonging to the computation of the scoring matrix presents no problem and seems to be plotted correctly, while the one belonging to the complete application seems to be wrongly plotted, this may be due to the errors mentioned above.