CM30225 Parallel Computing Assessed Coursework Assignment 2

The document describes the design and method used to implement in parallel a simple relaxation technique using the C MPI standard. Furthermore, it assesses program tests and the benefits in efficiency when performing a parallel computation.

1 ALGORITHM APPROACH

The program is divided into four main stages which include array initialisation, processing, optional printing of the array and lastly deallocation of used resources.

Table 1: Example of an input table 8x8. The white cells indicate cells 'to compute' while the orange cells are constant.

🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧

Initialisation

Firstly, each process reads the size of the array, before it reads the specific array from a file or creates the fragment of the test array independently to other processes. To divide the array into similar sized segments, each process will receive a minimum of two rows to compute and the maximum difference in the row count is one. The full size is divided by the number of processes and then the division remainder is distributed between the process of rank 0 to the process of rank ‘reminder-1’. An example of such division within a table of size 8 in between two processes is demonstrated in tables one-three. Each process reads/creates the corresponding table fragment separately due to a large overhead while sending many messages to processes making this type of data distribution inefficient.

Table 2: Illustration of a segment of an input array 8x8 at rank 0 if divided between two processes. White - to compute, orange - constant, red - to send, green - to receive.

🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Table 3: Illustration of a segment of an input array 8x8 at rank 1 if divided between two processes. White - to compute, orange - constant, red - to send, green - to receive.

🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	⬜	⬜	⬜	⬜	⬜	⬜	🟧
🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧

Processes synchronisation

As visible in table two and three, each thread has rows marked as red, green and white.

Red rows are computed first and then send, in a non-blocking manner, to a process with a higher and lower rank. The destination rank is based on the row position. For example, if the red row is a beginning row (the computation top row) or the end row (the computation bottom row). Then, the white rows are computed. These are not sent between processes. Lastly, a process waits to receive its green row/rows. This is a red row from a process below/above allowing the next cycle of computation. The process is repeated until every process has reached precision.

Computed array presentation

The program can be run with an additional ‘’-print’’ flag to print the array after the computation. As each segment must be printed in the rank order, the process must be sequential. This is implemented by each process going through a “for loop”. If the current value is the process number, then the given process prints the array. Following this, each process waits at a blocking barrier before progressing to the next iteration of the loop. The process is shown in figure one.

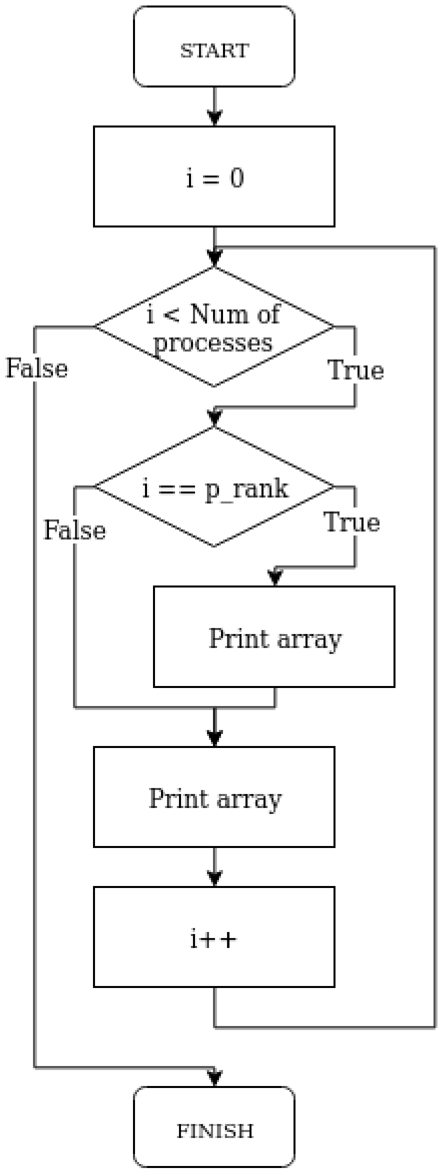


Figure 1: Print function flow diagram.

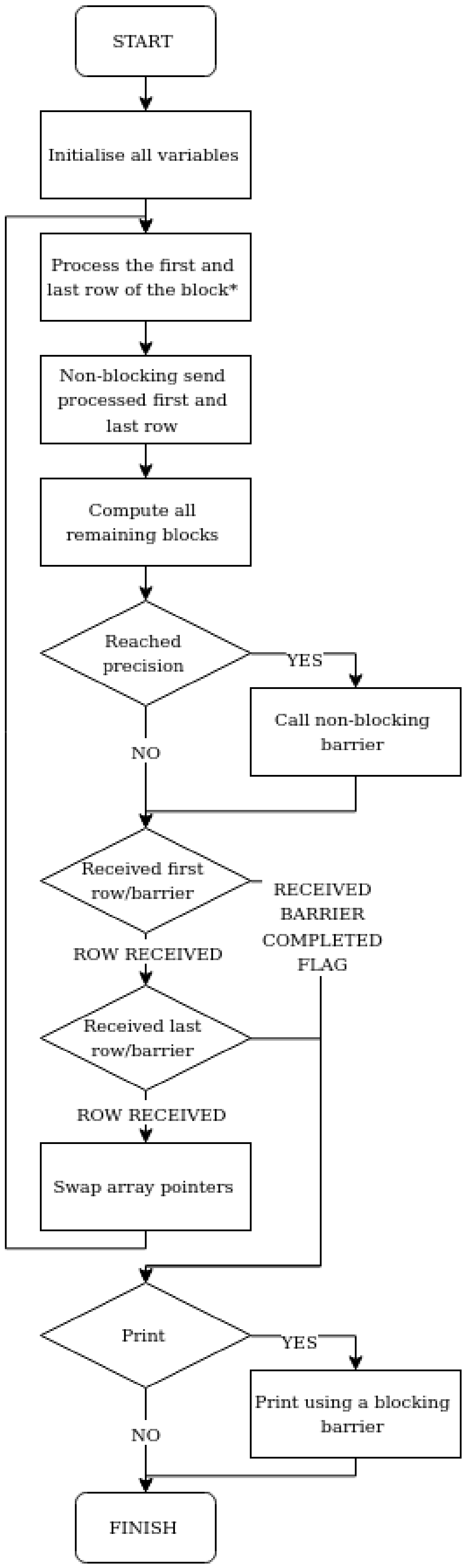
Deallocation

When a process has finished all operation, it deallocates the memory used and leaves the MPI routine.

This full process including initialisation, computing, printing and deallocation is demonstrated in figure two.

2 PROGRAM TESTING

Testing determines the quality of the software. The aim is to create an objective view of the risks of software implementation. There are several types of tests used to ensure the correct operation and output of the program.



START
Initialise all variables
Process the first and last row of the block*
Non-blocking send processed first and last row
Compute all remaining blocks
Reached precision
YES
Call non-blocking barrier
NO
Received first row/barrier
RECEIVED BARRIER COMPLETED FLAG
ROW RECEIVED
Received last row/barrier
ROW RECEIVED
Swap array pointers
Print
YES
Print using a blocking barrier
NO
FINISH

Figure 2: Diagram representing the program flow.

Compiler setup

During the compilation process, there are four main steps: pre-processing, compiling, assembling and linking. The program performing these tasks offers some error detection. This is often simple like incorrect type conversion but allows for elimination of a significant number of mistakes. Flags used during the compilation of the MPI relaxation program are as follow: -pedantic

-pedantic-errors -Wall -Wextra

-Wconversion -Werror. These confirm that the code conforms to C11 ISO standard. Additionally, this turns all other warning messages as well as an implicit conversion that may alter value. Lastly, all warnings are treated as errors stopping the compilation process.

Unit testing

A level of software testing where individual units/functions are tested. The purpose of these tests is to automate the validation of small parts of the system that work as designed. The program was designed to maximise utilisation of small, one-responsibility functions. Therefore, this enables each function to be tested independently from one another. There are 15 test cases testing 9 functions of 10 used in the program, meaning a high code test coverage. Figure three contains the output form tests. The only functions, not unit tested are those in the main body of the program as they used MPI protocols that are challenging to mock. Additionally, it is assumed that the MPI calls are already unit tested. Unit tests are available in files: ComputationTests.c, mainFunctions.c and ThreadTests.c while the list of functionalities tested are attached in appendix [1].

```
Finished. Computation Tests Success.

Finished. Thread Tests Success.

Precision must be positive integer larger than 0. // Expected

Finished. Main Tests Success.
```

Figure 3: Output from Unit Testing, Error line is expected as it tests for incorrect input variables

Memory leaks

A memory leak occurs when a piece of memory that was previously allocated by the program is not being deallocated before the program exits. This causes some of the systems still marking the memory as in-use despite the program exiting. Used programming language (C) due to explicit memory management, it is more difficult for programmes to implement correct memory management. Therefore, the program was tested using Valgrind, a memory management tool that is capable of spotting memory leaks. Unfortunately, as the MPI uses calls that are not directly in the program, when Valgrind is attached to one of the processes it will create false positives warnings. The partial output is visible in figure four. There were no memory leaks related to the program spotted by the Valgrind, the full log is attached in appendix [2].

```
==20452== 1,120 bytes in 1 blocks are definitely lost in loss record 186 of 200

==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)

==20452==    by 0x10AC46B6: ???

==20452==    by 0x4EC8172: mca_coll_base_comm_select (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)

==20452==    by 0x4E83C67: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)

==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)

==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
```

Figure 4 Partial output from memory leakage testing with Valgrind

Correctness testing

A strict compiler setup, unit testing and memory leakage tests provide strong indication that the program is working as desired. Furthermore, assuring the implemented logic and all processes within that program are producing outcome without any critical errors. However, these measures do not ensure that the program is working correctly as a whole and that the output of the program is correct. This is usually tested by running the program with known input and output and comparing if the output is the same as the desired. This is challenging to implement as an integration test in MPI due to each process being independent so manual testing was used. The program was run in different configurations in small array sizes (up to 100). Therefore, the number of cores used was below 49 as each process required at least two rows. Then a border was set to a value, for example, 5. As the program finished the process, all the cells should approach this value. Then a spreadsheet editor was used to compare the output values with the expected values. Table four, five and six show the output of the three different tests. The precision used was 10^{-9} while the border values were 5, 0 and -10 respectively. As the print option gives each value to 6 decimal places, significantly higher precision means all outputs should be equal to the border value.

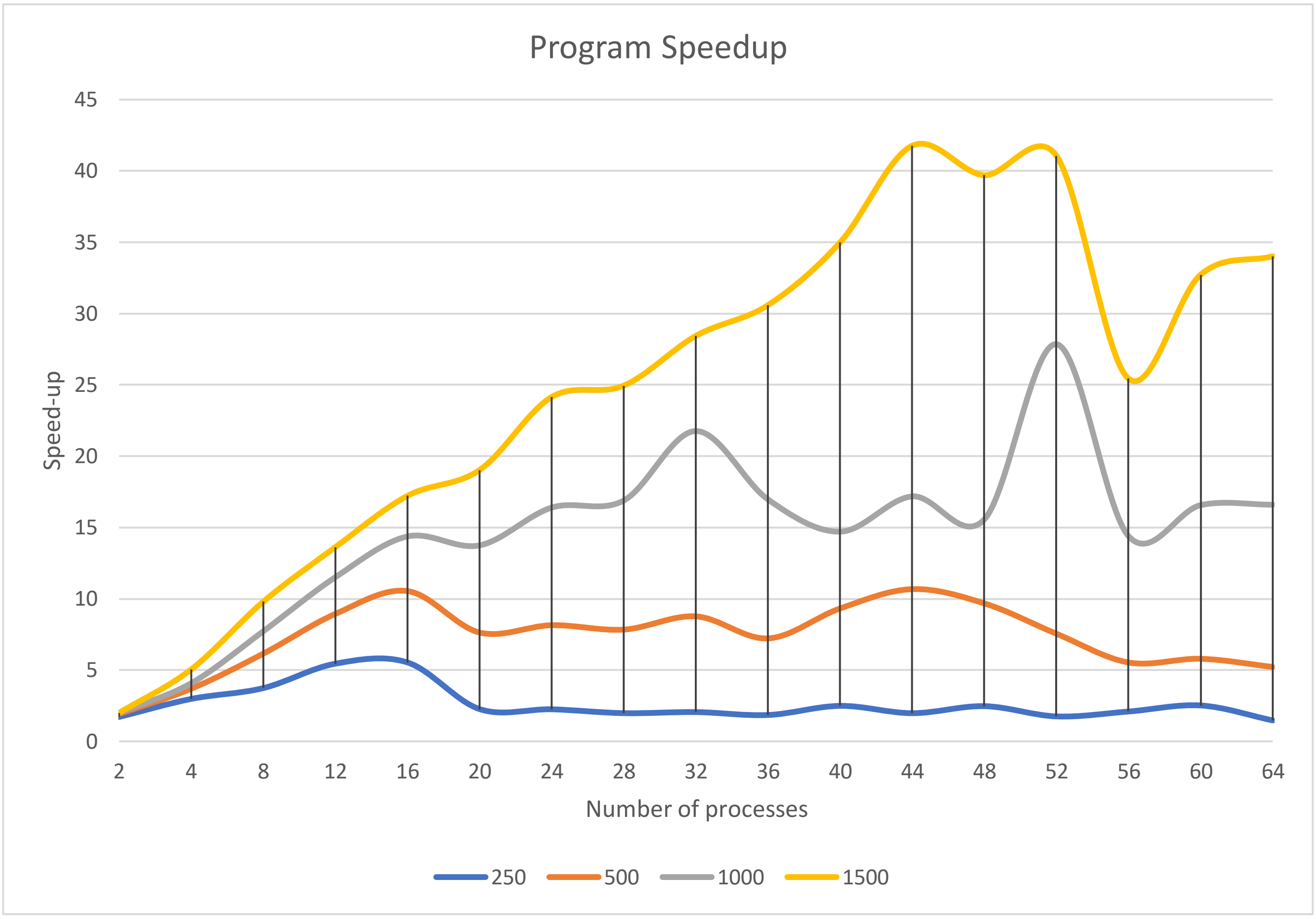


Figure 5: Gained speed-up with different size of the problem using a different number of processes

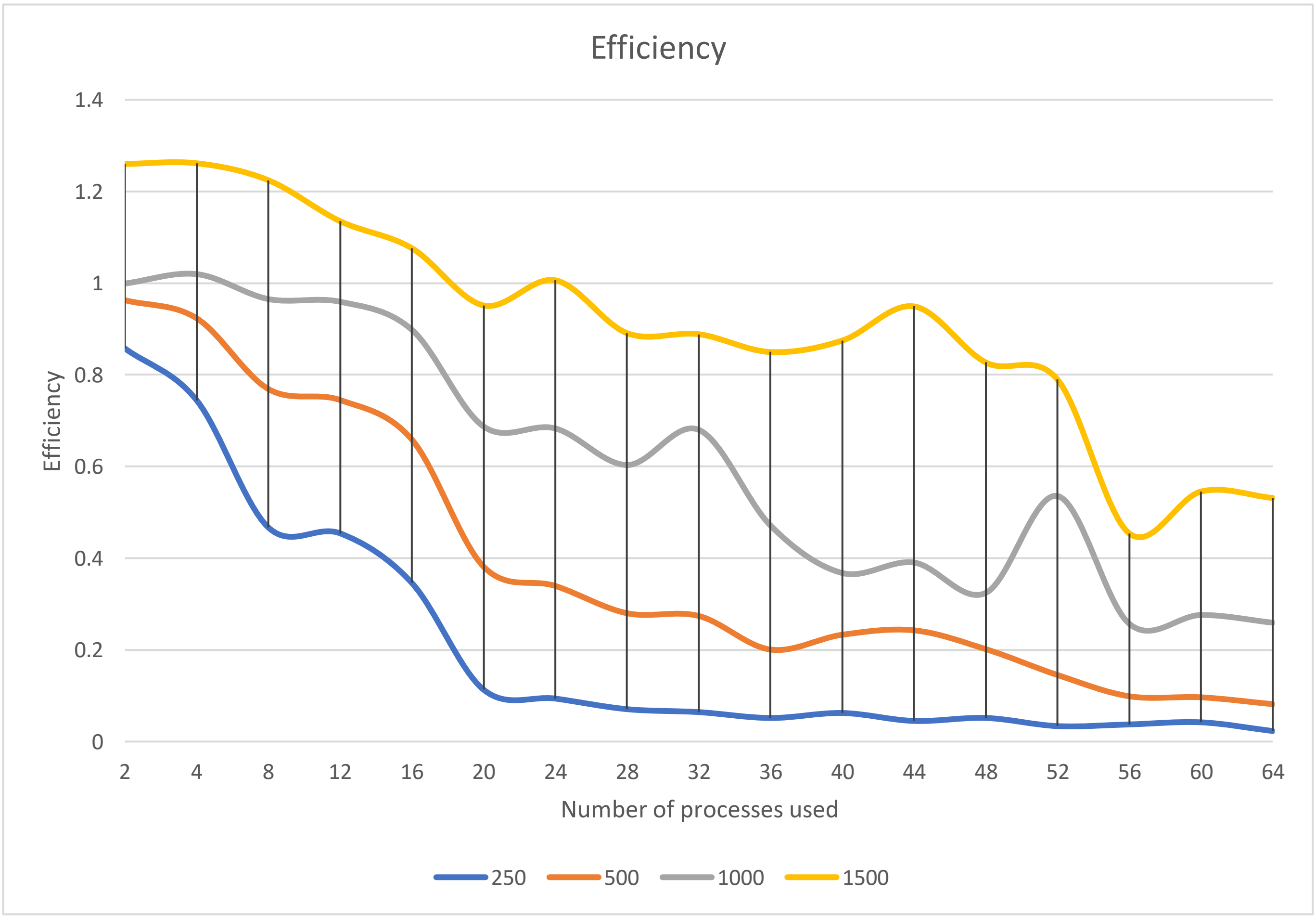


Figure 6: Gained speed-up with different size of the problem using different number of processes

3 SPEED AND EFFICIENCY

The speed and corresponding efficiency have been found by running different configurations of the program on the University Cluster Balena[3][4][5]. To limit the number of variables the starting values of the border were always equal to 5.000. Furthermore, 3 precision levels have been tested, 0.1, 0.001 and 0.00001. Lastly, the input table was in range 100 to 1500 due to the slowest one threaded operation in 0.00001 precision. Unfortunately, this was the limiting value in testing as the longest the program could run was 15 minutes. Lastly, for each precision and each array size, the program was run in 1, 2, 4, n+4 and 64 processes configuration.

Time was measured using a Linux program 'time' for the entire application including the initialisation, processing and memory deallocation.

Unfortunately, the first set of data found using the precision of 0.1 [3] was found to have very little information as the values were too close together due to low computation required in the relatively small array size to the precision. Therefore, it will be not used during within the remaining discussion.

Amdahl's Law

The law models the maximum speed-up limit assuming a fixed size of the problem. Figure 5 shows the graph for four different array sizes and the speed-up achieved. It can be noted that for each problem a different number of threads is optimal. When the number of threads is increased over that point the cost of communication becomes large enough to severely impact the performance of the system. The corresponding efficiency is visible in figure 6. It is also visible that as the number of threads increases the efficiency decreases, meaning the threads are idle and do not perform any operation, but are waiting for the messages with computed rows. Therefore, if the maximum size of the problem the program is designed to solve is up to 1500x1500 there is very little benefit of using all 64 processes but rather a higher efficiency and speed-up is achieved with only 44-52 processes.

Gustafson's Law

The law offers a different perspective on finding the maximum speed-up limit. As the problem increases when the number of processes is constant, the speed-up will tend to the number of processes as the parallel part of the program. This will be significantly larger

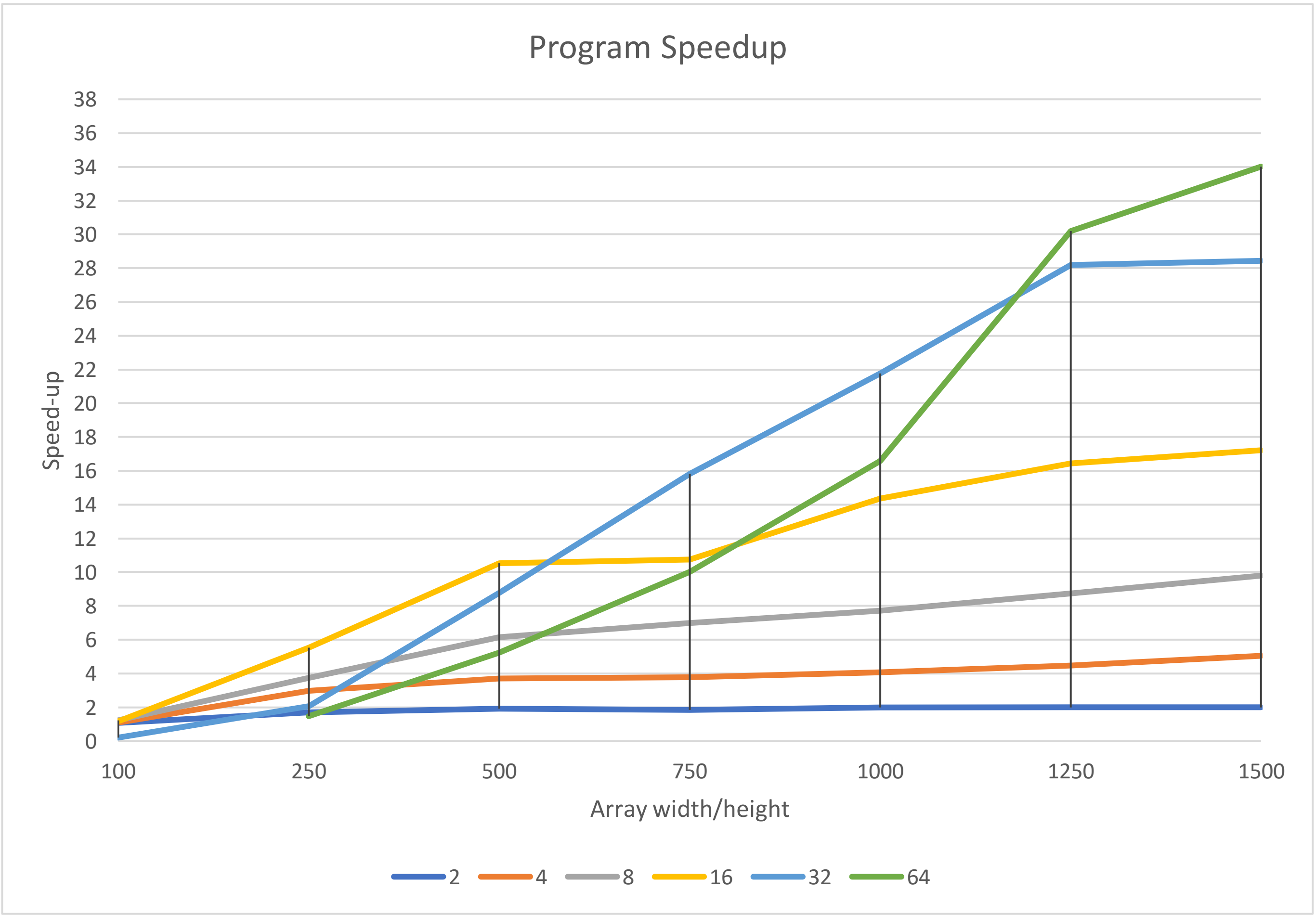


Figure 7: Speed-up scaling with increasing input array size.

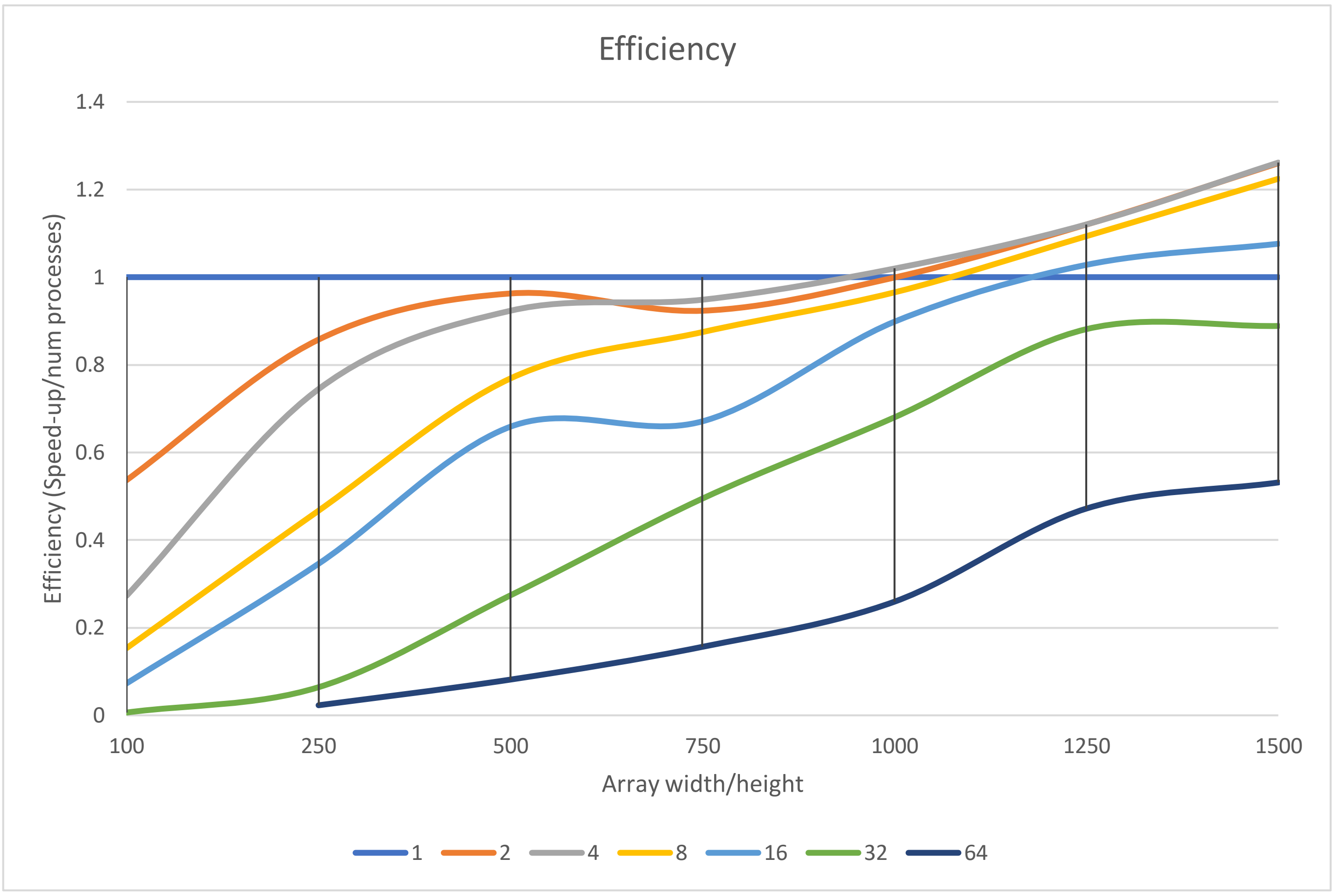


Figure 8: Efficiency of using different number of processes with increasing size of the input array.

than the sequential part. This effect is demonstrated in figure 7. It can be noted that as the size of the array increases, the values tend to be closer to the possible limit sometimes even making the speed-up larger than the number of processes used. There are two main explanations.

Firstly, it is very likely that the processes were interrupted for a moment, or the resources were allocated slightly differently, and this is a cause of the difference in the measurement.

A second possible explanation is that the sequential part of the program during the variable infantilization and array allocation the complexity of the task is not linear. Therefore, reducing this task will bring a greater speed-up. Consequently, smaller chunks of memory are allocated across 4 different nodes in the cluster.

Efficiency was also found as presented in figure 8. As shown, it was going towards one and exceeding it. Furthermore, 32 and 64 configurations did not manage to reach its full potential and the problem would need to exceed the size of 1500 to make better utilisation of these processes.

Nodes Distribution

It is important to note that with a constant and relatively small array size there might be small increase or even a decrease in speed-up when the number of processes increases over a single node. Adding a new node means a significant increase in the messaging overhead that is very visible on smaller arrays. For example, in figure 5 for an array size of 1000x1000, there is a decrease in speed-up between 32 and 36 cores when a new node was added. To overcome this sequential messaging constraint, the problem size should be increased.

4 CONCLUSION

MPI offers a flexible standard for implementing parallel programs in C/C++. Speed-up and efficiency testing, as discussed earlier, show that for larger problems the messaging system works very well despite being distributed over different machines. Furthermore, the program is flexible by providing the input arguments meaning it can be run with different configurations with code compiled in the same way. Unfortunately, MPI is difficult to debug and test for memory leakage. This might be a smaller issue for a larger project with easier accessibility to the hardware as it would be more beneficial to set up a remote debugging machine.

APPENDIX

[1] Names of unit tests

- computeAverageOfFour_inputTheSameDouble_expectTheSameOutput
- computeAverageOfFour_inputNegativeNumer_expectCorrectOutput
- computeAverageOfFour_allZeros_expectZero
- createArrayAnddealocateArray
- computeAveragesForARow_inputIncreasingArray_expectCorrectOutput
- computeAveragesForARow_inputArrayWithTheSameData_expectCorrectOutput
- computeAveragesForARow_inputTooSmallArray_expectReturnNoProcessing
- mainSetup_correctInput_correctOutput
- mainSetup_tooSmallArray_ErrorOutput
- mainSetup_tooZeroPrecison_ErrorOutput
- mainSetup_dontPrint_PrintIsFalse
- coefficients_correctIn_correctOut
- coefficients_moreThreadsThanRows_outZero
- coefficients_unevenThreadsToColumnsRetio
- initialiseEmptyThreadInfo_passEmpty_correctDataOut

[2] Output from Valgrind

```
==20452== Memcheck, a memory error detector
==20452== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20452== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==20452== Command: ./ParallelOpenMPI -p 0.001 -print -testArray 20 5
==20452==
==20452== Conditional jump or move depends on uninitialised value(s)
==20452==    at 0x57D4375: opal_value_unload (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x4E7F97A: ompi_proc_complete_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4E838A4: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452==
==20452== HEAP SUMMARY:
==20452==     in use at exit: 184,913 bytes in 685 blocks
==20452==   total heap usage: 20,531 allocs, 19,846 frees, 4,217,554 bytes allocated
==20452==
==20452== 10 bytes in 1 blocks are definitely lost in loss record 15 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x51CB9B9: strdup (strdup.c:42)
==20452==    by 0x57F2AA8: mca_base_var_enum_create_flag (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x58066AF: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F5004: mca_base_framework_register (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F5080: mca_base_framework_open (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F50F0: mca_base_framework_open (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x4E834A2: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 11 bytes in 1 blocks are definitely lost in loss record 18 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x7AA6DAD: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x7AA54FD: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x7ABB2EB: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x7ABA004: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x580D208: opal_libevent2022_event_base_loop (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x7AB7FCC: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x62436DA: start_thread (pthread_create.c:463)
==20452==    by 0x524F88E: clone (clone.S:95)
==20452==
==20452== 11 bytes in 1 blocks are definitely lost in loss record 19 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x51CB9B9: strdup (strdup.c:42)
==20452==    by 0xFFE4ED0: ???
==20452==    by 0x4010732: call_init (dl-init.c:72)
==20452==    by 0x4010732: _dl_init (dl-init.c:119)
==20452==    by 0x40151FE: dl_open_worker (dl-open.c:522)
```

```
==20452==    by 0x52952DE: _dl_catch_exception (dl-error-skeleton.c:196)
==20452==    by 0x40147C9: _dl_open (dl-open.c:605)
==20452==    by 0x645BF95: dlopen_doit (dlopen.c:66)
==20452==    by 0x52952DE: _dl_catch_exception (dl-error-skeleton.c:196)
==20452==    by 0x529536E: _dl_catch_error (dl-error-skeleton.c:215)
==20452==    by 0x645C734: _dlerror_run (dlerror.c:162)
==20452==    by 0x645C050: dlopen@@GLIBC_2.2.5 (dlopen.c:87)
==20452==
==20452== 17 bytes in 1 blocks are definitely lost in loss record 24 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x51CB9B9: strdup (strdup.c:42)
==20452==    by 0x57F2AA8: mca_base_var_enum_create_flag (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x58066C9: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F5004: mca_base_framework_register (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F5080: mca_base_framework_open (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F50F0: mca_base_framework_open (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x4E834A2: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 17 bytes in 1 blocks are definitely lost in loss record 25 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x51CB9B9: strdup (strdup.c:42)
==20452==    by 0x57F2AA8: mca_base_var_enum_create_flag (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x4E83876: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 40 bytes in 1 blocks are definitely lost in loss record 108 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0xA00068A: ???
==20452==    by 0x9FFF918: ???
==20452==    by 0x9F876C9: ???
==20452==    by 0x9F65CAA: ???
==20452==    by 0x9D2BB0D: ???
==20452==    by 0x60194C2: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6010910: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6009020: hwloc_topology_set_xmlbuffer (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x788663A: ???
==20452==    by 0x55331DD: orte_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x4E8327D: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==
==20452== 40 bytes in 1 blocks are definitely lost in loss record 109 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0xA00068A: ???
==20452==    by 0x9FA4059: ???
==20452==    by 0x9F876EA: ???
==20452==    by 0x9F65CAA: ???
==20452==    by 0x9D2BB0D: ???
==20452==    by 0x60194C2: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6010910: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6009020: hwloc_topology_set_xmlbuffer (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x788663A: ???
==20452==    by 0x55331DD: orte_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x4E8327D: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==
==20452== 48 bytes in 1 blocks are definitely lost in loss record 136 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x57E99CC: mca_base_component_repository_open (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57E8A09: mca_base_component_find (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F4BC9: mca_base_framework_components_register (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F5019: mca_base_framework_register (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F5080: mca_base_framework_open (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57CEB7A: opal_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x5533078: orte_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x4E8327D: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 56 bytes in 1 blocks are definitely lost in loss record 149 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0xD84D037: ???
==20452==    by 0xD227884: ???
==20452==    by 0xD632128: ???
==20452==    by 0x556F52C: orte_oob_base_select (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x5560C04: orte_ess_base_app_setup (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x7886A28: ???
==20452==    by 0x55331DD: orte_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
```

```
==20452==    by 0x4E8327D: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 68 bytes in 12 blocks are definitely lost in loss record 152 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x51CB9B9: strdup (strdup.c:42)
==20452==    by 0x1060D087: ???
==20452==    by 0x1060D822: ???
==20452==    by 0x1060E0E4: ???
==20452==    by 0x103FC4E7: ???
==20452==    by 0x4EDD4AF: ompi_mtl_base_select (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0xF5509C9: ???
==20452==    by 0x4EE59A4: mca_pml_base_select (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4E83544: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 88 bytes in 1 blocks are definitely lost in loss record 156 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x6A6F4A8: lt__malloc (in /usr/lib/x86_64-linux-gnu/libltdl.so.7.3.1)
==20452==    by 0x6A6F4DD: lt__zalloc (in /usr/lib/x86_64-linux-gnu/libltdl.so.7.3.1)
==20452==    by 0x6A71779: ??? (in /usr/lib/x86_64-linux-gnu/libltdl.so.7.3.1)
==20452==    by 0x6A72120: lt_dlopenadvise (in /usr/lib/x86_64-linux-gnu/libltdl.so.7.3.1)
==20452==    by 0x747CE85: ???
==20452==    by 0x600FAE0: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6008E86: hwloc_topology_init (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x7886610: ???
==20452==    by 0x55331DD: orte_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x4E8327D: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==
==20452== 104 bytes in 1 blocks are definitely lost in loss record 162 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0xA00073A: ???
==20452==    by 0xA06AA2C: ???
==20452==    by 0x624B826: __pthread_once_slow (pthread_once.c:116)
==20452==    by 0xA000AE0: ???
==20452==    by 0xA000028: ???
==20452==    by 0x9F876CE: ???
==20452==    by 0x9F65CAA: ???
==20452==    by 0x9D2BB0D: ???
==20452==    by 0x60194C2: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6010910: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6009020: hwloc_topology_set_xmlbuffer (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==
==20452== 371 (56 direct, 315 indirect) bytes in 1 blocks are definitely lost in loss record 172 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x7ACD6F5: PMIx_Store_internal (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x7A9114D: pmix1_store_local (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x5560CC2: orte_ess_base_app_setup (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x7886A28: ???
==20452==    by 0x55331DD: orte_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x4E8327D: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 376 (232 direct, 144 indirect) bytes in 1 blocks are definitely lost in loss record 173 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0xEB1F02F: ???
==20452==    by 0xF328C19: ???
==20452==    by 0x4E838F4: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 726 (720 direct, 6 indirect) bytes in 1 blocks are definitely lost in loss record 179 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0xE1022B4: ???
==20452==    by 0x556DC7C: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x556EE27: orte_oob_base_set_addr (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x580D208: opal_libevent2022_event_base_loop (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57D217D: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x62436DA: start_thread (pthread_create.c:463)
==20452==    by 0x524F88E: clone (clone.S:95)
==20452==
==20452== 826 (56 direct, 770 indirect) bytes in 1 blocks are definitely lost in loss record 181 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x7ABC820: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
```

```
==20452==    by 0x580D208: opal_libevent2022_event_base_loop (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x7AB7FCC: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x62436DA: start_thread (pthread_create.c:463)
==20452==    by 0x524F88E: clone (clone.S:95)
==20452==
==20452== 910 (400 direct, 510 indirect) bytes in 1 blocks are definitely lost in loss record 183 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x9F6043F: ???
==20452==    by 0x9F876F4: ???
==20452==    by 0x9F65CAA: ???
==20452==    by 0x9D2BB0D: ???
==20452==    by 0x60194C2: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6010910: ??? (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x6009020: hwloc_topology_set_xmlbuffer (in /usr/lib/x86_64-linux-gnu/libhwloc.so.5.7.6)
==20452==    by 0x788663A: ???
==20452==    by 0x55331DD: orte_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-rte.so.20.10.1)
==20452==    by 0x4E8327D: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==
==20452== 1,120 bytes in 1 blocks are definitely lost in loss record 185 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x10AC46B6: ???
==20452==    by 0x4EC8172: mca_coll_base_comm_select (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4E83C48: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 1,120 bytes in 1 blocks are definitely lost in loss record 186 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x10AC46B6: ???
==20452==    by 0x4EC8172: mca_coll_base_comm_select (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4E83C67: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== 4,152 bytes in 1 blocks are definitely lost in loss record 192 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x7ADA303: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x7ADBD3C: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x7AC0BBA: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x580D208: opal_libevent2022_event_base_loop (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x7AB7FCC: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x62436DA: start_thread (pthread_create.c:463)
==20452==    by 0x524F88E: clone (clone.S:95)
==20452==
==20452== 4,152 bytes in 1 blocks are definitely lost in loss record 193 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x7ADA4B6: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x7ADBD3C: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x7AC0BBA: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x580D208: opal_libevent2022_event_base_loop (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x7AB7FCC: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pmix_pmix112.so)
==20452==    by 0x62436DA: start_thread (pthread_create.c:463)
==20452==    by 0x524F88E: clone (clone.S:95)
==20452==
==20452== 16,568 bytes in 1 blocks are definitely lost in loss record 198 of 200
==20452==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==20452==    by 0x57C56AA: opal_free_list_grow_st (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57CB9CE: opal_rb_tree_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0xE9199AF: ???
==20452==    by 0xE91A1EC: ???
==20452==    by 0x57EA129: mca_base_framework_components_open (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x58211E1: ??? (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x57F50F0: mca_base_framework_open (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libopen-pal.so.20.10.1)
==20452==    by 0x4E83482: ompi_mpi_init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x4EA42AA: PMPI_Init (in /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so.20.10.1)
==20452==    by 0x10990C: main (ParallelOpenMPI.c:28)
==20452==
==20452== LEAK SUMMARY:
==20452==    definitely lost: 29,086 bytes in 33 blocks
==20452==    indirectly lost: 1,745 bytes in 55 blocks
==20452==      possibly lost: 0 bytes in 0 blocks
==20452==    still reachable: 154,082 bytes in 597 blocks
==20452==         suppressed: 0 bytes in 0 blocks
==20452== Reachable blocks (those to which a pointer was found) are not shown.
==20452== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==20452==
==20452== For counts of detected and suppressed errors, rerun with: -v
```

```
==20452== Use --track-origins=yes to see where uninitialised values come from
==20452== ERROR SUMMARY: 23 errors from 23 contexts (suppressed: 0 from 0)
```

[3] Table with collected runtime of the program with precision = 0.1

	100	250	500	750	1000	1250	1500
1	0.344	0.35	0.322	0.365	0.549	0.503	0.462
2	0.305	3.074	0.321	6.413	0.326	0.43	0.688
4	0.333	0.337	0.336	0.876	0.341	0.373	0.661
8	0.411	0.338	0.457	0.488	0.549	0.371	0.472
16	1.093	3.184	0.886	0.492	0.391	0.428	0.963
32	2.668	3.416	2.758	2.7	3.542	3.489	2.649
64	To Small Arr	2.679	2.812	2.643	9.172	2.956	2.803

[4] Table with collected runtime of the program with precision = 0.001

	100	250	500	750	1000	1250	1500
1	0.554	0.587	1.386	2.655	5.037	7.8	10.354
2	1.024	3.461	1.711	2.317	3.012	4.172	10.083
4	0.362	0.562	0.603	0.948	1.55	2.024	2.938
8	0.397	0.431	0.54	1.465	0.915	1.424	2.406
12	0.453	0.432	0.474	0.591	0.803	1.02	1.296
16	5.357	0.532	0.454	1.049	0.774	1.082	1.38
20	2.804	2.671	2.799	2.757	3.036	3.151	3.466
24	2.765	2.72	2.676	2.727	2.996	2.936	3.227
28	2.643	2.704	2.687	2.758	2.84	3.044	3.054
32	3.512	2.779	2.682	2.934	3.291	3.232	4.222
36	2.662	2.744	2.828	2.81	2.844	3.111	3.014
40	2.727	2.73	2.681	2.839	2.984	2.881	3.049
44	2.865	2.875	2.691	2.804	2.839	2.851	3.025
48	2.713	2.64	2.851	2.771	2.961	2.804	3.01
52	To Small Arr	2.676	2.793	2.776	2.891	2.844	3.227
56	To Small Arr	2.683	2.753	2.907	2.979	2.951	3.167
60	To Small Arr	2.705	2.726	2.969	3.018	3.119	3.236
64	To Small Arr	3.522	2.961	3.075	3.415	3.043	3.013

[5] Table with collected runtime of the program with precision = 0.00001

	100	250	500	750	1000	1250	1500
1	0.644	8.148	83.934	295.814	694.591	1069.248	1246.332
2	2.726	4.751	43.592	160.189	347.606	534.624	623.166
4	0.59	2.736	22.725	77.967	170.311	238.616	247.003
8	0.525	2.181	13.644	42.277	89.951	122.233	127.235
12	0.529	1.495	9.389	30.127	60.334	83.721	91.506
16	0.552	1.472	7.963	27.555	48.329	65.002	72.381
20	3.109	3.601	10.999	22.539	50.538	56.728	65.509
24	2.709	3.621	10.303	20.055	42.371	47.572	51.614
28	2.699	4.118	10.71	23.534	41.124	43.24	49.965
32	3.171	3.966	9.575	18.687	31.922	37.913	52.831
36	2.899	4.402	11.624	25.79	40.95	40.899	45.743
40	2.811	3.271	9.004	23.936	47.235	33.995	40.623
44	2.688	4.12	7.862	12.823	40.434	33.642	29.842
48	2.714	3.297	8.674	22.426	44.628	28.264	31.4
52	To Small Arr	4.652	11.119	18.659	24.941	26.878	30.358
56	To Small Arr	3.892	15.173	18.298	48.256	29.356	49.012
60	To Small Arr	3.235	14.49	21.307	41.936	34.471	38.087
64	To Small Arr	5.54	16.076	29.562	41.855	35.407	36.645