Università degli Studi di Salerno
DIEM, a.a 2021-22
Course: HIGH PERFORMANCE COMPUTING
Lecturer: Francesco Moscato

PARALLELIZATION AND PERFORMANCES EVALUATION OF ***COUNTING SORT*** ALGORITHM WITH **MPI**

Salvatore Grimaldi 0622701742 s.grimaldi29@studenti.unisa.it
Enrico Maria Di Mauro 0622701706 e.dimauro5@studenti.unisa.it
Allegra Cuzzocrea 0622701707 a.cuzzocrea2@studenti.unisa.it

Index

Introduction

The main purpose of this report is parallelizing and evaluating performances of Counting Sort Algorithm. MPI (Message Passing Interface) is the communication protocol for parallel programming used in this report.

In the following pages the problem faced is going to be described in a detailed way, paying great attention to the description of the case studies considered. The results are going to be analysed and explained in the light of the theoretical knowledge acquired during the High-Performance Computing course held by prof. Francesco Moscato at University of Salerno.

Problem description

The problem is how to parallelize and evaluate performances of **Counting Sort Algorithm**, by using **MPI**. Counting sort is an integer sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array. The count is stored in an auxiliary array that is then used to get the actual sorted array.

Counting sort can be used only to sort collections of objects whose keys are positive integers: this is the reason why it is described as an integer sorting algorithm. Counting sort is not a comparison sort, which means that it is not based on comparisons among the objects of the collection meant to be sorted. Counting sort running time is linear in the number of items and the difference between the maximum key value and the minimum key value, so it is only suitable for direct use in situations where the variation in keys is not significantly greater than the number of items. Moreover, to make things easier and faster to understand, it has been chosen to deal with a collection that is a simple array of integers.

Counting sort algorithm can be divided in few steps:

1. A for-loop determines the minimum (**min**) and the maximum (**max**) integers inside the unsorted array **a**.
2. An auxiliary array **c**, whose length is $\boldsymbol{max - min + 1}$ is allocated.
3. A for-loop initializes to 0 all **c** elements.
4. A for loop traverses **a** and at each iteration increments by 1 the **c** element placed in the position corresponding to the visited **a** element.
5. A for loop increments every **c** element by adding to it the sum of all its previous elements in **c**.
6. Two nested for-loops exploit **c** content in order to sort in place array **a**.

Parallelization is obtained thanks to MPI. It is a standardized and portable message-passing standard designed to function on parallel computing architectures. The MPI standard defines the syntax and semantics of library routines that are useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran (in this report we deal with C language).

There are several open-source MPI implementations, such as **MPICH** (the one used for this project), OpenMPI and Intel MPI. Message Passing Interface is based on a **Distributed Memory Model**, in which several nodes send and receive messages, so that a sort of memory abstraction is created.

MPI was born to allow parallel computing to improve performances. MPI is not a programming language, but a framework in which a fundamental role is played by the so-called middleware MPI, which manages synchronization among parallel processes and messages mechanism. MPI is mostly thought for physical **clusters** (e.g., raspberry clusters), but ensures good performances even when the program is run by a single machine, if the number of generated processes is not greater than the number of available cores.

One of the main MPI concepts is the meaning of **communicator**: a sort of container of processes, in which each of them is identified by a rank. Communicator size is given by the number of processes it manages and cannot be changed after Communicator creation.

Theoretical notes

Given that this report purpose is evaluating performances through speed-up and efficiency, it is fundamental to clarify these two words meanings.

- **Speed-up**: it is a number that measures the relative performance of two systems processing the same problem. More technically, it is the improvement in speed of execution of a task executed on two similar architectures with different resources. The notion of speed-up was established by Amdahl's law, which was particularly focused on parallel processing. For example, let's consider she same program executed on N processors and on 1 processor: if the program executed on only 1 processor is K times slower than the same program executed on N processors, we can say that the obtained speed-up is equal to K. In the most of cases, speed-up functions, plotted considering the same algorithm launched on a different number of processors, are not linear but present a particular trend that can be explained in the light of machine characteristics, algorithm complexity, parallelizable portions, framework used, etc.
- **Efficiency**: it is given by the execution time of the algorithm on 1 processor divided by P * the execution time of the same algorithm on P processors.

$$E = T_1/PT_p$$

Efficiency is a useful measure of parallel algorithm quality.

Experimental setup

Information provided below refers to hardware and software used during the evaluation of performances.

Hardware

CPU

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
stepping        : 12
microcode       : 0xffffffff
cpu MHz         : 1992.006
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid
sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi
ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
vmx flags       : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept
vpid unrestricted_guest ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
srbds
bogomips        : 3984.01
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
stepping        : 12
microcode       : 0xffffffff
cpu MHz         : 1992.006
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 1
initial apicid  : 1
```

```
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid
sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi
ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
vmx flags       : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept
vpid unrestricted_guest ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
srbds
bogomips        : 3984.01
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 2
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
stepping        : 12
microcode       : 0xffffffff
cpu MHz         : 1992.006
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 1
cpu cores       : 4
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid
sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi
ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
vmx flags       : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept
vpid unrestricted_guest ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
srbds
bogomips        : 3984.01
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
```

```
stepping        : 12
microcode       : 0xffffffff
cpu MHz         : 1992.006
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 1
cpu cores       : 4
apicid          : 3
initial apicid  : 3
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid
sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi
ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
vmx flags       : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept
vpid unrestricted_guest ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
srbds
bogomips        : 3984.01
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 4
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
stepping        : 12
microcode       : 0xffffffff
cpu MHz         : 1992.006
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 2
cpu cores       : 4
apicid          : 4
initial apicid  : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid
sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi
ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
vmx flags       : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept
vpid unrestricted_guest ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
srbds
bogomips        : 3984.01
```

```
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 5
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
stepping        : 12
microcode       : 0xffffffff
cpu MHz         : 1992.006
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 2
cpu cores       : 4
apicid          : 5
initial apicid  : 5
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid
sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi
ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
vmx flags       : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept
vpid unrestricted_guest ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
srbds
bogomips        : 3984.01
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 6
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
stepping        : 12
microcode       : 0xffffffff
cpu MHz         : 1992.006
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 3
cpu cores       : 4
apicid          : 6
initial apicid  : 6
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
```

```
constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid
sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi
ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
vmx flags       : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept
vpid unrestricted_guest ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
srbds
bogomips        : 3984.01
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 7
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
stepping        : 12
microcode       : 0xffffffff
cpu MHz         : 1992.006
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 3
cpu cores       : 4
apicid          : 7
initial apicid  : 7
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid
sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm
3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi
ept vpid ept_ad fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt xsaveopt xsavec xgetbv1 xsaves flush_l1d arch_capabilities
vmx flags       : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept
vpid unrestricted_guest ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
srbds
bogomips        : 3984.01
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

RAM

```
MemTotal:         3930548 kB
MemFree:          3242452 kB
MemAvailable:     3273480 kB
Buffers:            51864 kB
Cached:            167144 kB
SwapCached:             0 kB
Active:             98372 kB
Inactive:          323472 kB
Active(anon):         220 kB
Inactive(anon):    202908 kB
Active(file):       98152 kB
Inactive(file):    120564 kB
Unevictable:            0 kB
Mlocked:                0 kB
SwapTotal:        1048576 kB
SwapFree:         1048576 kB
Dirty:                 16 kB
Writeback:              0 kB
AnonPages:         202820 kB
Mapped:             64436 kB
Shmem:                284 kB
KReclaimable:       25488 kB
Slab:               56136 kB
SReclaimable:       25488 kB
SUnreclaim:         30648 kB
KernelStack:         4128 kB
PageTables:          8060 kB
NFS_Unstable:           0 kB
Bounce:                 0 kB
WritebackTmp:           0 kB
CommitLimit:      3013848 kB
Committed_AS:      893480 kB
VmallocTotal:34359738367 kB
VmallocUsed:        23996 kB
VmallocChunk:           0 kB
Percpu:              2624 kB
AnonHugePages:      24576 kB
ShmemHugePages:         0 kB
ShmemPmdMapped:         0 kB
FileHugePages:          0 kB
FilePmdMapped:          0 kB
HugePages_Total:        0
HugePages_Free:         0
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:        2048 kB
Hugetlb:                0 kB
DirectMap4k:        19456 kB
DirectMap2M:      3033088 kB
DirectMap1G:      9437184 kB
```

Software

```
Ubuntu      : 20.04.3
GCC         : 9.3.0
MPICH       : 3.3.2
```

Case studies and performances

In this report two different case studies are considered:

- Case Study n.1: it is characterized by the use of the typical MPI collective primitives. Moreover, files are NOT used.
- Case Study n.2: it is characterized by the use of files, which are particularly interesting in MPI.

In both the case studies the sizes considered are 1mln and 10mln, where sizes are the dimensions of the arrays meant to be sorted. Moreover, two ranges are taken in account: 1k, 100k. The range represents the maximum integer that can be found in the unsorted array.

The choice regarding the use of multiple sizes and ranges is determined by the intention of testing the counting sort algorithm in different conditions. One of the purposes is comparing algorithm performances, in terms of speed-up and efficiency, for different sizes and ranges.

It is important to notice that in the following case studies are represented also the speed-ups obtained thanks to the parallelization of unsorted array initialization.

Graphic and table legend

The graphic and table legend useful to read the results is reported below:

- **Version**: program version considered (serial or parallel). Each case study has a serial and a parallel version
- **Processes**: number of processes that are involved in program execution
- **TimeInit**: elapsed time for unsorted array initialization
- **TimeCount**: elapsed time for counting sort algorithm
- **Speedup_timeInit**: speed-up corresponding to the unsorted array initialization portion of the program
- **Efficiency_timeInit**: efficiency corresponding to the unsorted array initialization portion of the program
- **Speedup_timeCount**: speed-up corresponding to the counting sort portion of the program
- **Efficiency_timeCount**: efficiency corresponding to the counting sort portion of the program

A table and two graphs are provided for each measure. In particular, the first graph represents the trend of initialization speed-up, while the second one represents the trend of sorting speed-up.

It is relevant to observe that in all speed-up graphs the point corresponding to 1 process is obtained comparing the parallel version execution time (with number of processes equal to 1) to the sequential version execution time. All the other points, instead, are obtained comparing the parallel version execution time (with number of processes equal to N, depending on the point) to the parallel version execution time with 1 process.

Case Study n.1

To better understand the following results, describing the logic behind the program is very useful.

The sequential version is extremely easy: it takes two parameters as input, which represent size (unsorted array length) and range. Then it calls a function, whose name is *init()* and whose role is initializing randomically the unsorted array. In the end, another function, whose name is *countingSort(),* is called: it just sorts the previous created array using counting sort algorithm.

The parallel version is a bit more complex. There are always two parameters with the same meaning seen above, but before dealing with them it is necessary to initialize the MPI environment. The only process that allocates the entire unsorted array **full_array**, according to the size inserted as input, is the one with rank 0. Then every process, included the one with rank 0, calls a function named *init()*, which is meant to initialize the unsorted array. This purpose is reached thanks to the following logic: every process must allocate and randomically initialize an array called **piece_init_array**. Afterwards, all these arrays are put together and sent to the process with rank 0, which keeps them in **full_array**.

Finished the initialization, it is counting sort turn: every process calls a function named *countingSort()*. The unsorted array **full_array** previously created must be scattered among all the available processes, so that each of them can work on a portion called **piece_of_array.** The i-th process computes the minimum and the maximum of its **piece_of_array**. These results are local so a reduction is compulsory to get the global maximum (**max**) and the global minimum (**min**). After that, the i-th process must allocate an auxiliary array named **c_local**, whose length is equal to $\boldsymbol{max - min + 1}$.

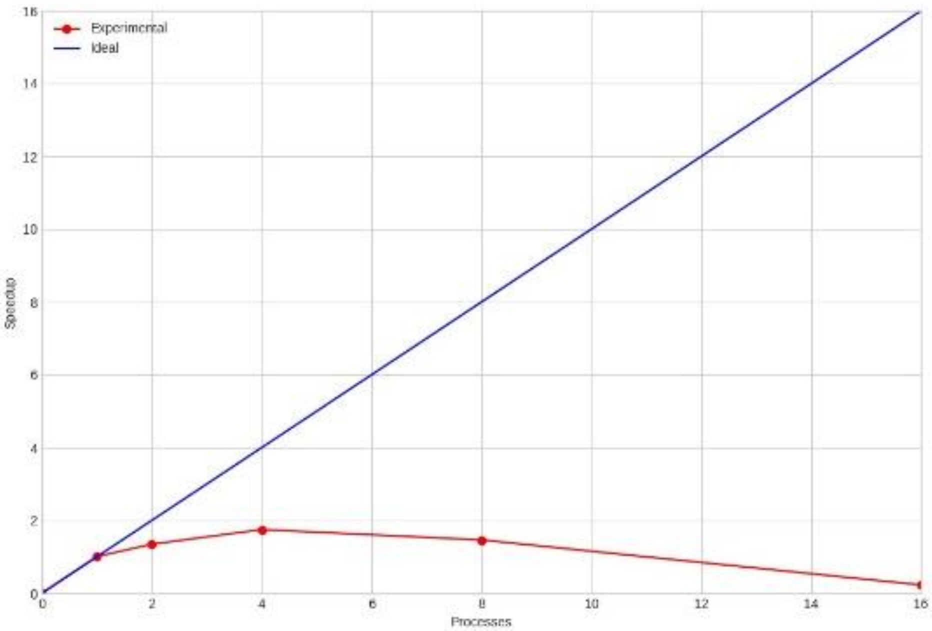
Three for-loops are executed: the first one initializes to 0 every **c_local** element; the second one increments by 1 the **c_local** element placed in the position corresponding to the visited **piece_of_array** element; the third one increments every **c_local** element by adding to it the sum of all its previous elements in **c_local**. After that, all the different **c_local** must be summed together through a reduction: the result is represented by another array, whose name is **c**, allocated only by rank 0 process, which can eventually complete counting sort, executing two for-loops that exploit **c** content to sort **full_array** in place.

It is important to observe that the job of measuring elapsed time is assigned to the process with rank 0. In fact it is the last one to finish *countingSort()*, and only when it terminates, **full_array** is actually sorted. TimeInit, instead, is basically the same for all the processes.

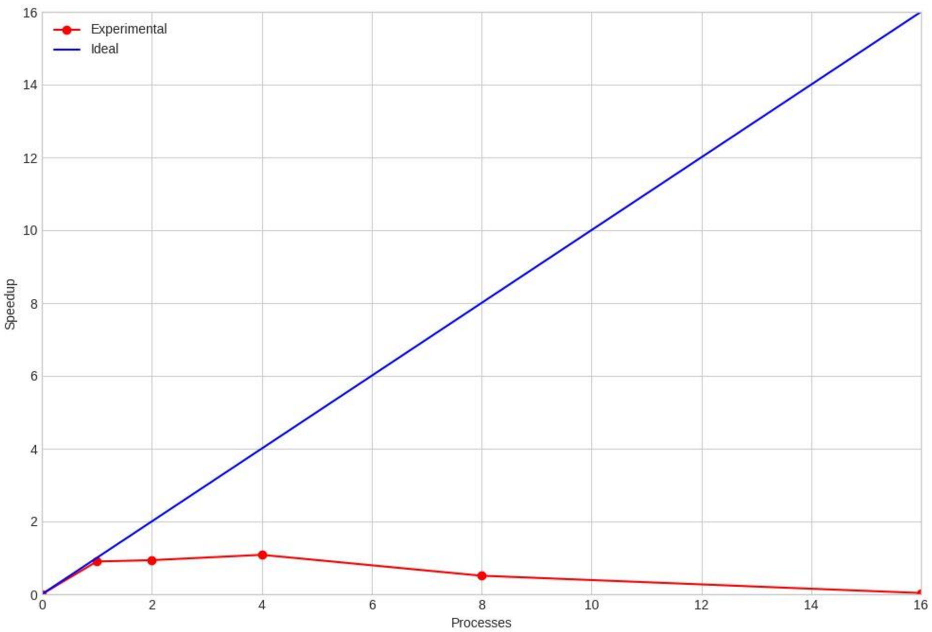
The following graphs are obtained considering not only different sizes and ranges, but also different gcc optimizations.

SIZE-1mln-RANGE-1k-OPT-O0

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.013959929	0.0088956	1	1	1	1
Parallel	1	0.013830929	0.00987975	1.009326923	1.009326923	0.900387156	0.900387156
Parallel	2	0.010272545	0.010548462	1.346397408	0.673198704	0.936605776	0.468302888
Parallel	4	0.0078986	0.009114545	1.751060767	0.437765192	1.083954219	0.270988555
Parallel	8	0.009442071	0.019371154	1.464819312	0.183102414	0.510023826	0.063752978
Parallel	16	0.060261083	0.286810769	0.22951676	0.014344797	0.034446928	0.002152933



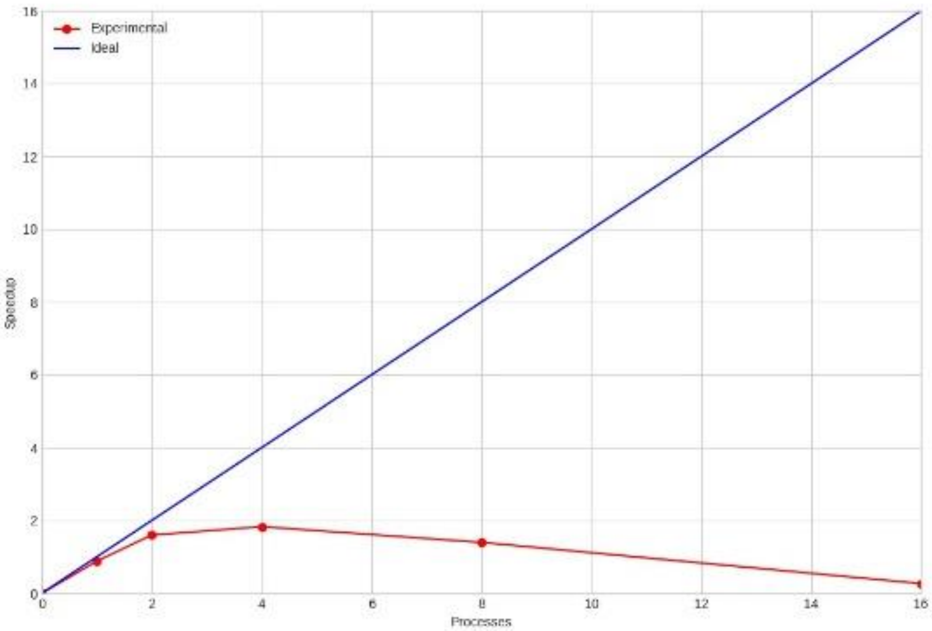
Speedup_timeInit



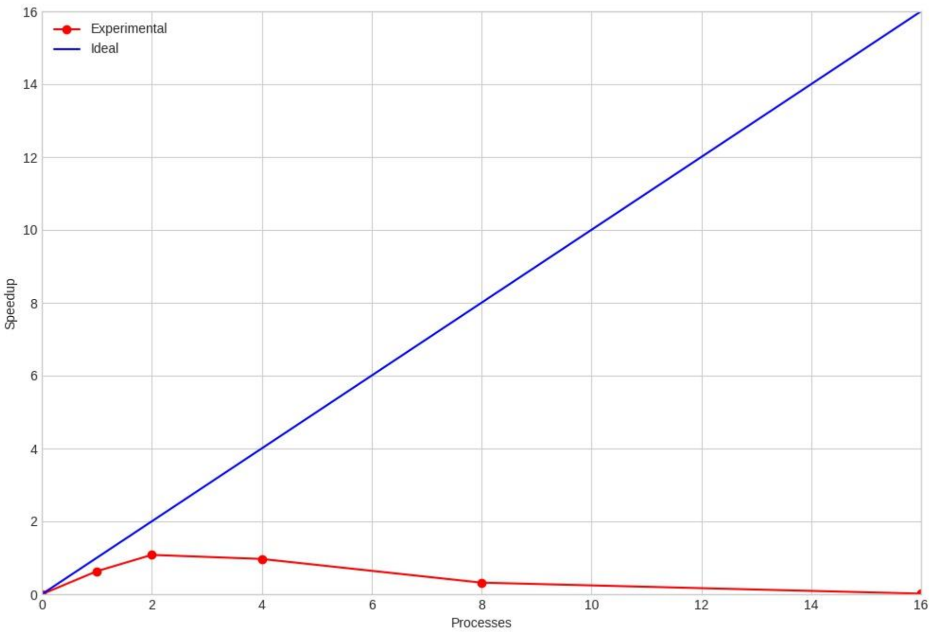
Speedup_timeCount

SIZE-1mln-RANGE-1k-OPT-O1

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.012185267	0.002657214	1	1	1	1
Parallel	1	0.013804765	0.004206556	0.882685575	0.882685575	0.631684106	0.631684106
Parallel	2	0.008634625	0.003893333	1.598768297	0.799384148	1.080450913	0.540225457
Parallel	4	0.007541917	0.004341714	1.830405362	0.45760134	0.968869732	0.242217433
Parallel	8	0.0098772	0.013204385	1.397639483	0.174704935	0.318572632	0.039821579
Parallel	16	0.051621071	0.250419929	0.267424994	0.016714062	0.016798006	0.001049875



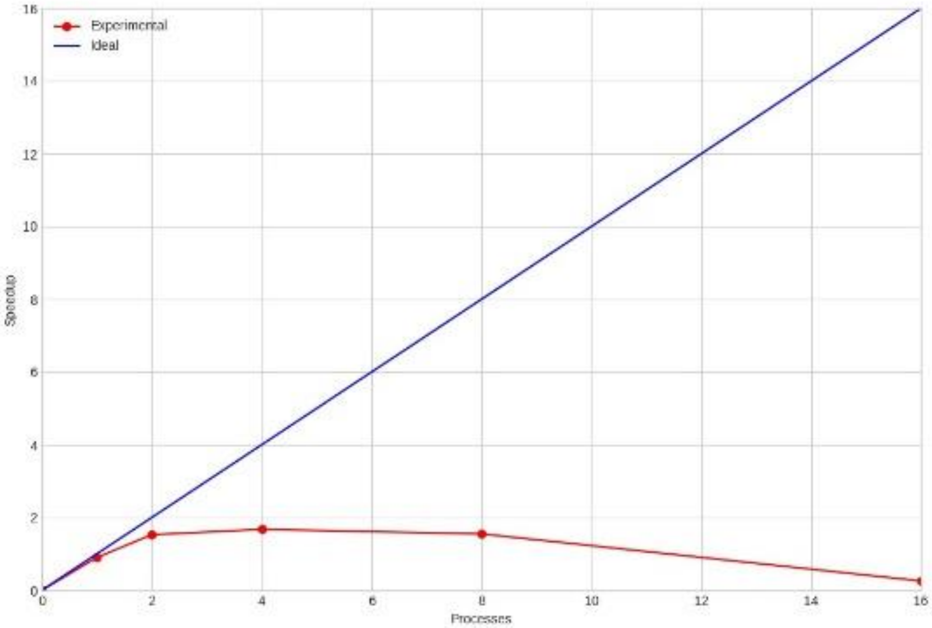
Speedup_timeInit



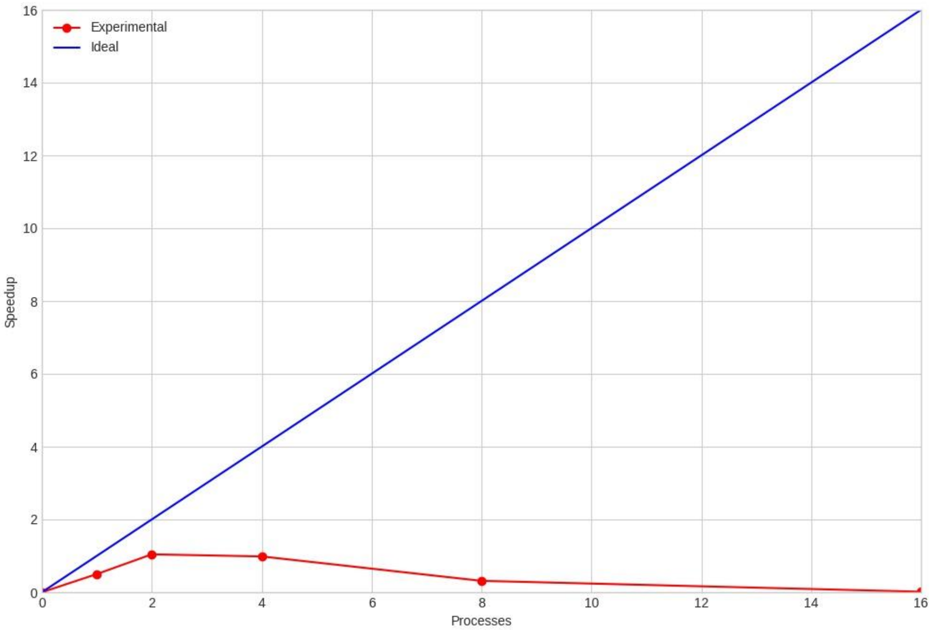
Speedup_timeCount

SIZE-1mln-RANGE-1k-OPT-O2

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.012237867	0.001918467	1	1	1	1
Parallel	1	0.013589588	0.003837077	0.900532559	0.900532559	0.499981289	0.499981289
Parallel	2	0.008890882	0.003681353	1.528485891	0.764242946	1.042300748	0.521150374
Parallel	4	0.008107824	0.003895533	1.676108044	0.419027011	0.98499399	0.246248498
Parallel	8	0.0087588	0.012218	1.551535397	0.193941925	0.314051148	0.039256393
Parallel	16	0.052776917	0.251612833	0.257491136	0.016093196	0.015249925	0.00095312



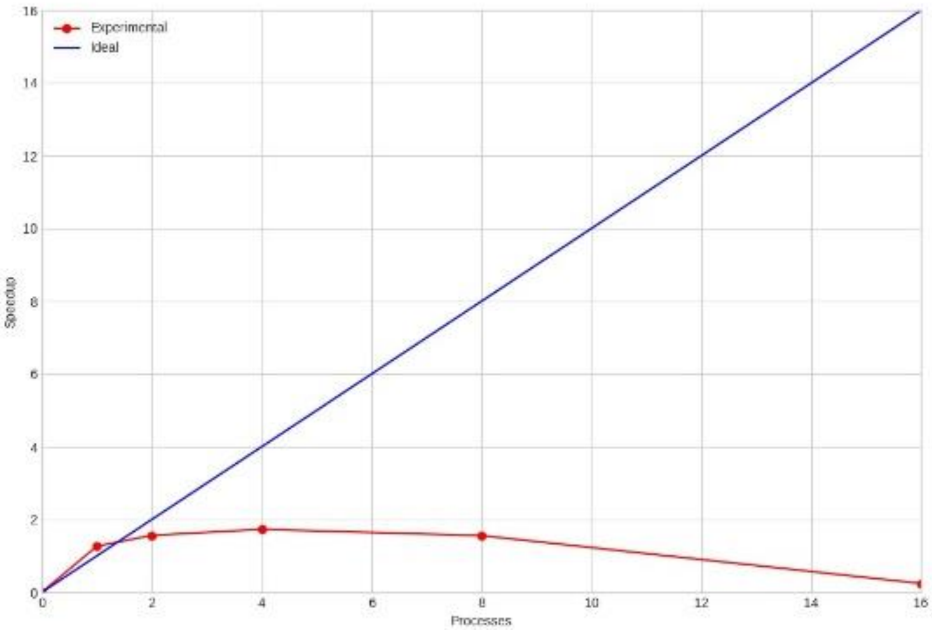
Speedup_timeInit



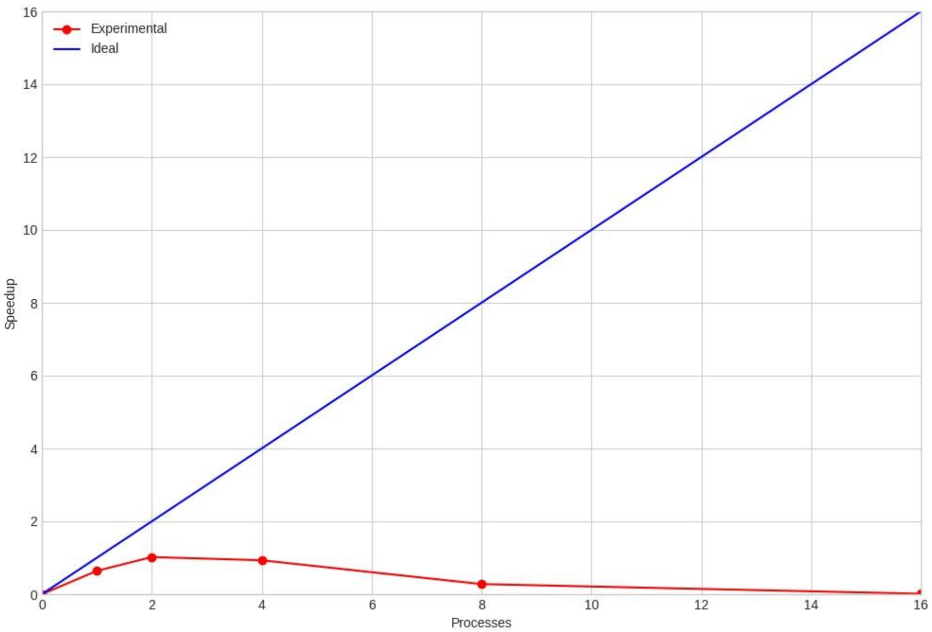
Speedup_timeCount

SIZE-1mln-RANGE-1k-OPT-O3

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.016688077	0.002383579	1	1	1	1
Parallel	1	0.013177533	0.003719882	1.266403696	1.266403696	0.640767293	0.640767293
Parallel	2	0.008417824	0.0036455	1.565432358	0.782716179	1.020403882	0.510201941
Parallel	4	0.007607929	0.003989154	1.732079003	0.433019751	0.932499095	0.233124774
Parallel	8	0.008448529	0.013336692	1.559742849	0.194967856	0.27892091	0.034865114
Parallel	16	0.054157818	0.265384077	0.243317286	0.01520733	0.014016976	0.000876061



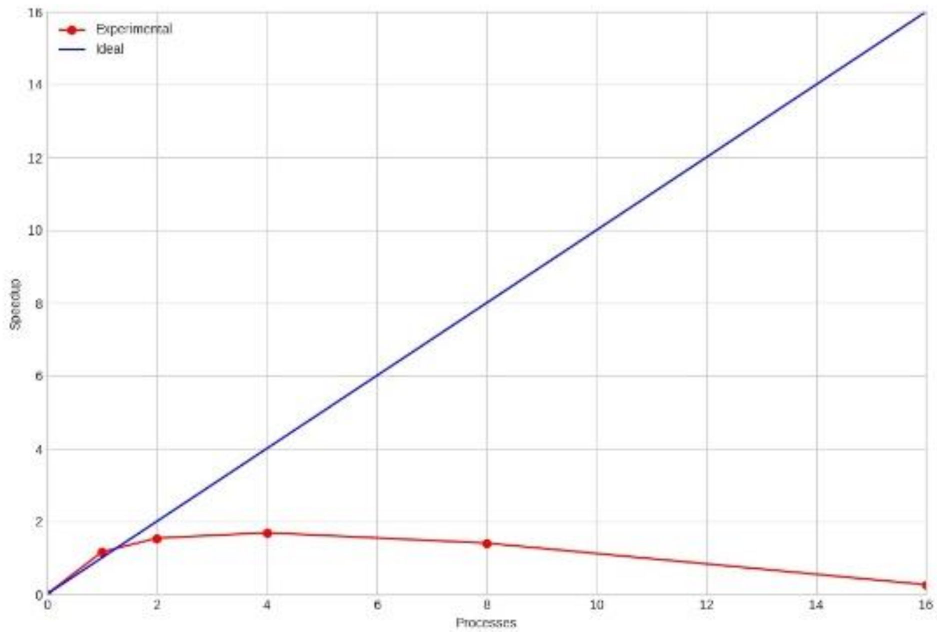
Speedup_timeInit



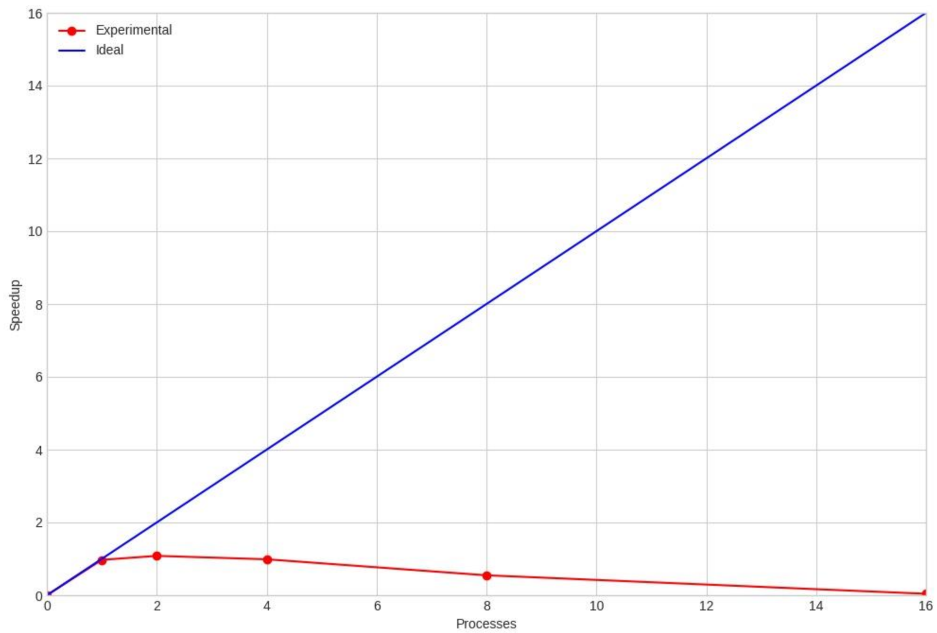
Speedup_timeCount

SIZE-1mln-RANGE-100k-OPT-O0

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.015675667	0.014094947	1	1	1	1
Parallel	1	0.013501	0.014491125	1.161074488	1.161074488	0.972660671	0.972660671
Parallel	2	0.008754824	0.013383769	1.542121318	0.771060659	1.082738708	0.541369354
Parallel	4	0.007978462	0.014661706	1.692180872	0.423045218	0.98836555	0.247091388
Parallel	8	0.009602385	0.026387105	1.406004919	0.175750615	0.549174487	0.068646811
Parallel	16	0.050908733	0.355363733	0.265200077	0.016575005	0.040778289	0.002548643



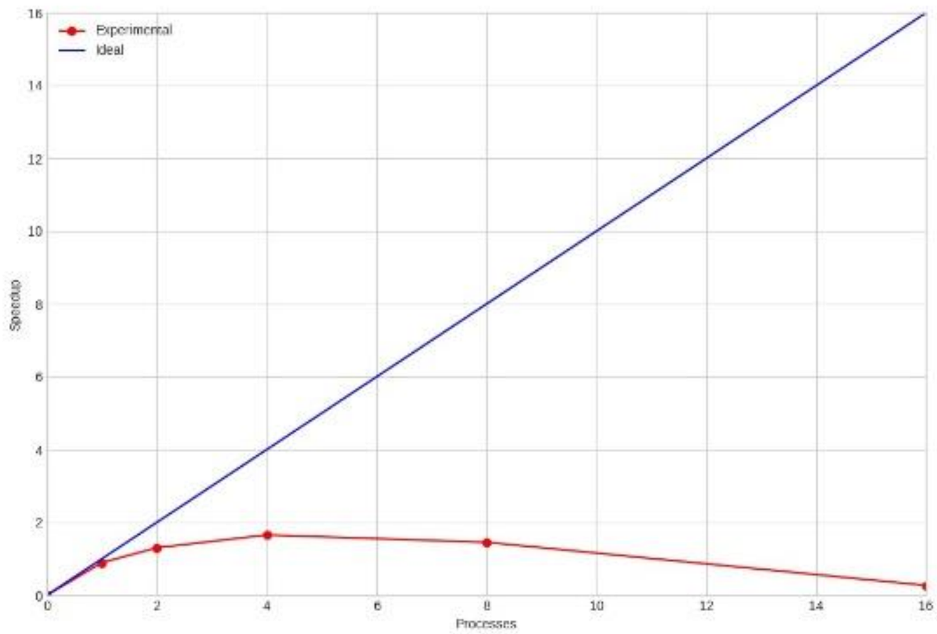
Speedup_timeInit



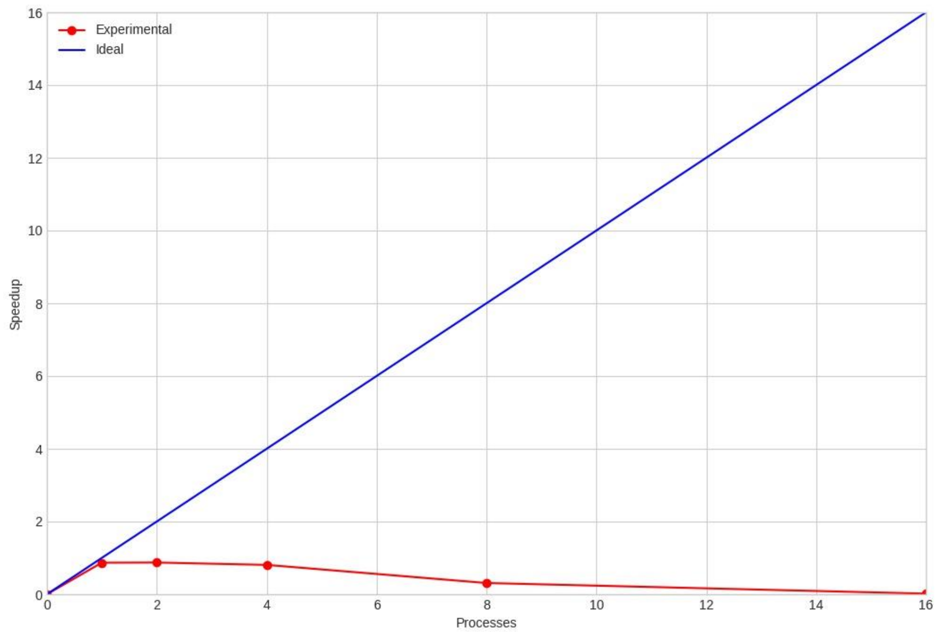
Speedup_timeCount

SIZE-1mln-RANGE-100k-OPT-O1

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.012082929	0.005404077	1	1	1	1
Parallel	1	0.013490867	0.006218273	0.895637684	0.895637684	0.869063993	0.869063993
Parallel	2	0.0103436	0.007112222	1.304271885	0.652135942	0.874307992	0.437153996
Parallel	4	0.008137941	0.007687333	1.657773923	0.414443481	0.808898542	0.202224636
Parallel	8	0.009266158	0.019997938	1.455928856	0.181991107	0.310945703	0.038868213
Parallel	16	0.049485364	0.339309462	0.272623371	0.017038961	0.018326258	0.001145391



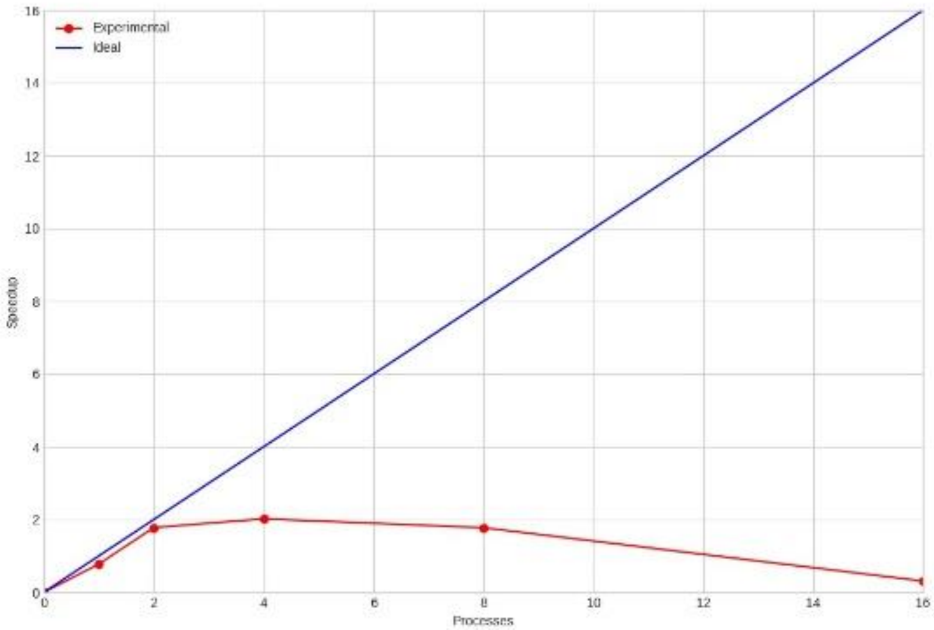
Speedup_timeInit



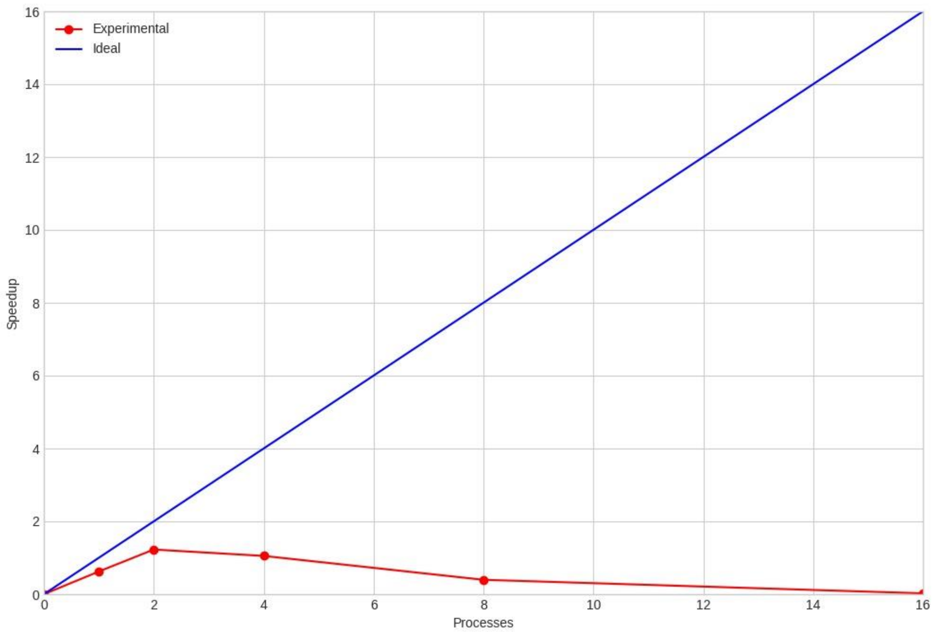
Speedup_timeCount

SIZE-1mln-RANGE-100k-OPT-O2

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.012393929	0.004717333	1	1	1	1
Parallel	1	0.015973786	0.007563857	0.775891751	0.775891751	0.623667693	0.623667693
Parallel	2	0.008979611	0.006161438	1.778895045	0.889447523	1.227612411	0.613806205
Parallel	4	0.007911643	0.007187231	2.019022598	0.504755649	1.052402154	0.263100539
Parallel	8	0.008991882	0.019057769	1.776467383	0.222058423	0.396891003	0.049611375
Parallel	16	0.051410667	0.303320846	0.310709562	0.019419348	0.024936819	0.001558551



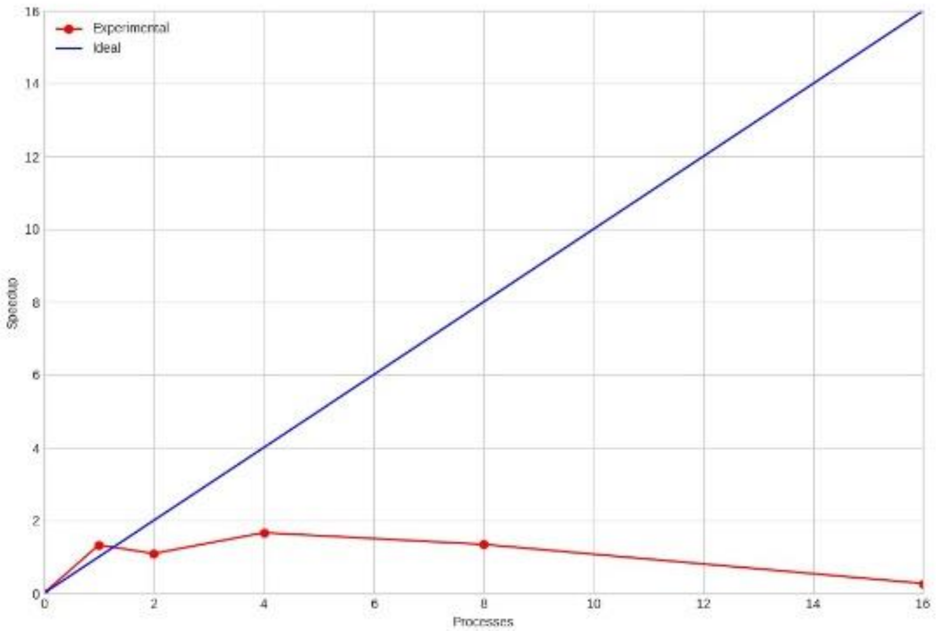
Speedup_timeInit



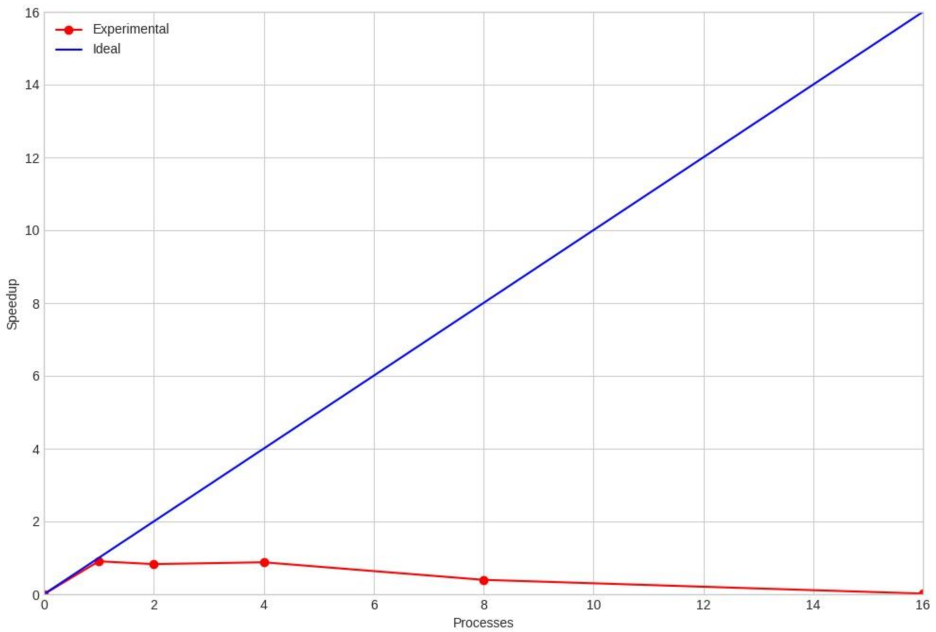
Speedup_timeCount

SIZE-1mln-RANGE-100k-OPT-O3

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.0170855	0.005821308	1	1	1	1
Parallel	1	0.012858308	0.006425647	1.32875184	1.32875184	0.90594887	0.90594887
Parallel	2	0.011783357	0.007752286	1.091226171	0.545613085	0.82887129	0.414435645
Parallel	4	0.007723583	0.007314385	1.664811155	0.416202789	0.878494555	0.219623639
Parallel	8	0.0095778	0.016183385	1.342511609	0.167813951	0.397052113	0.049631514
Parallel	16	0.0475958	0.341756	0.270156352	0.016884772	0.018801856	0.001175116



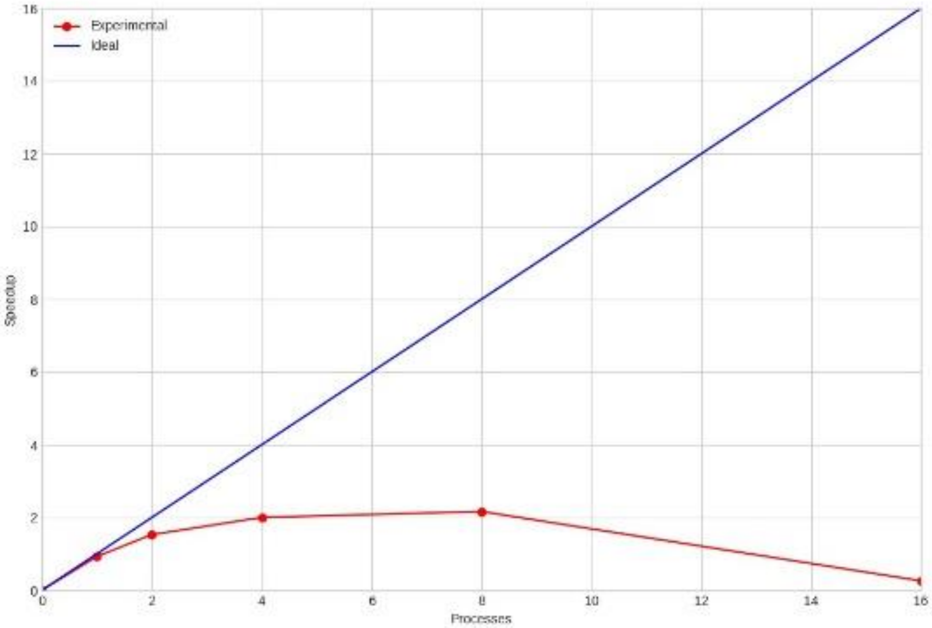
Speedup_timeInit



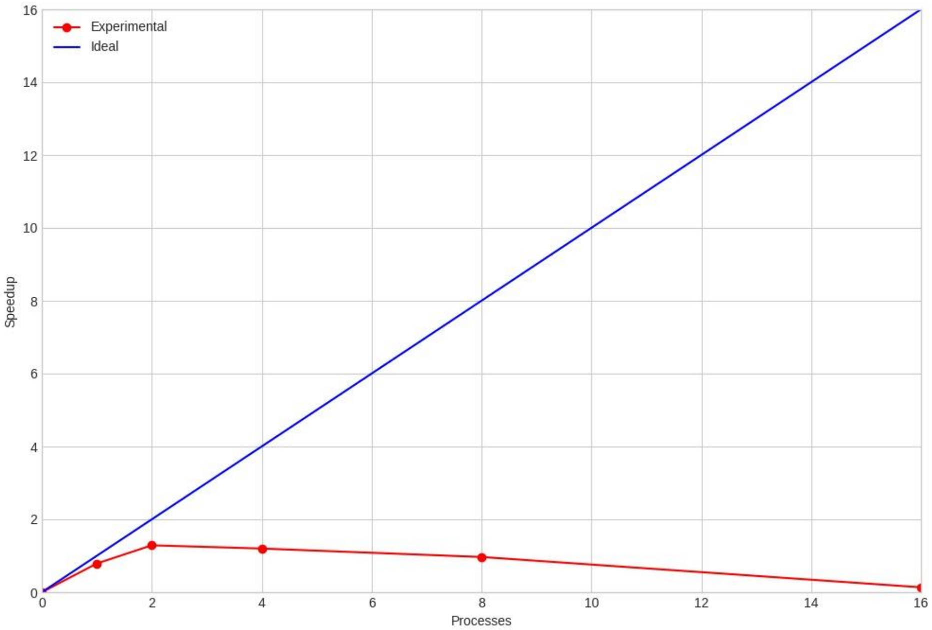
Speedup_timeCount

SIZE-10mln-RANGE-1k-OPT-O0

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.113145	0.076958059	1	1	1	1
Parallel	1	0.121964611	0.097649133	0.92768713	0.92768713	0.788107955	0.788107955
Parallel	2	0.079702778	0.07581	1.530242916	0.765121458	1.288077211	0.644038605
Parallel	4	0.060879533	0.081379	2.003376249	0.500844062	1.199930367	0.299982592
Parallel	8	0.056294667	0.100898273	2.16653936	0.27081742	0.967797869	0.120974734
Parallel	16	0.467889429	0.7167142	0.26066973	0.016291858	0.136245568	0.008515348



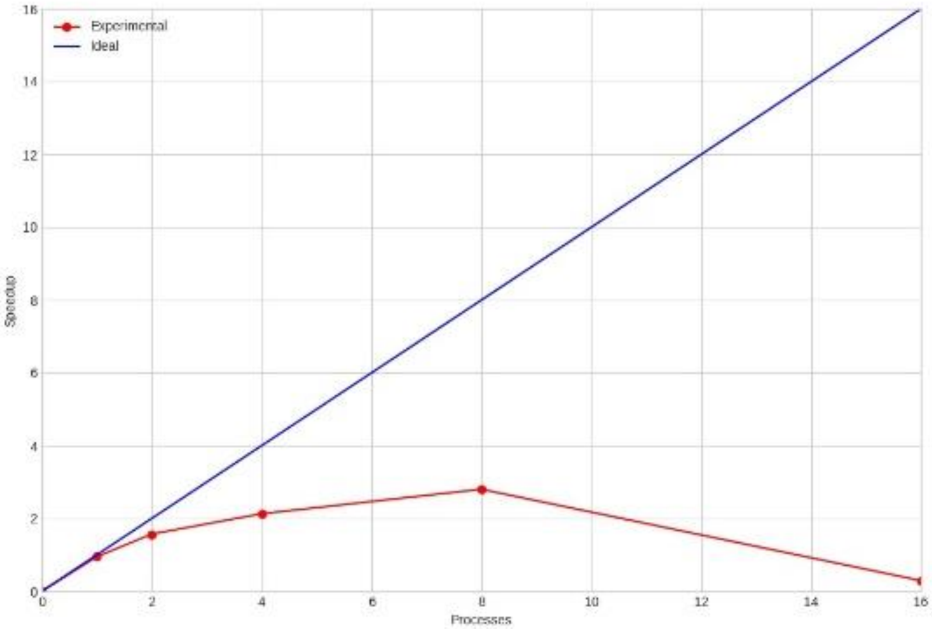
Speedup_timeInit



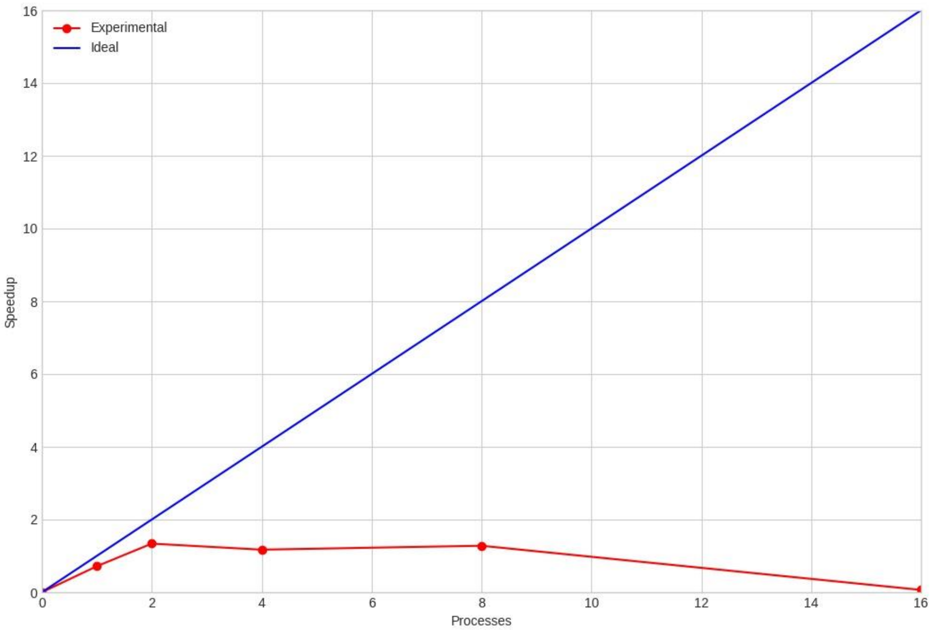
Speedup_timeCount

SIZE-10mln-RANGE-1k-OPT-O1

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.118283467	0.029825571	1	1	1	1
Parallel	1	0.124705125	0.041615786	0.948505257	0.948505257	0.716688894	0.716688894
Parallel	2	0.079373786	0.031089118	1.571112224	0.785556112	1.338596553	0.669298276
Parallel	4	0.058554143	0.035499067	2.129740423	0.532435106	1.172306475	0.293076619
Parallel	8	0.044501353	0.032509	2.802277161	0.350284645	1.280131216	0.160016402
Parallel	16	0.427509429	0.631099077	0.291701461	0.018231341	0.065941763	0.00412136



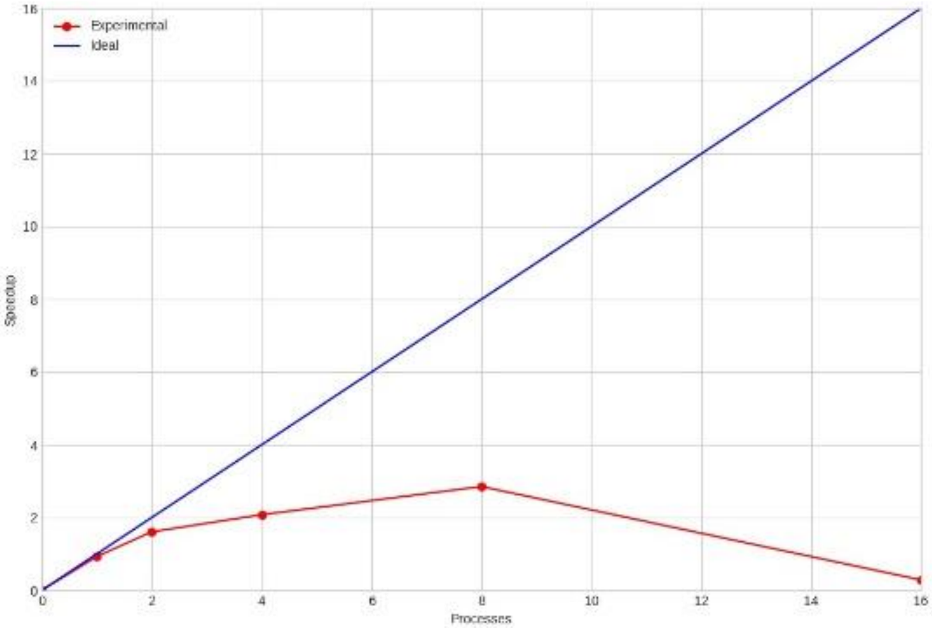
Speedup_timeInit



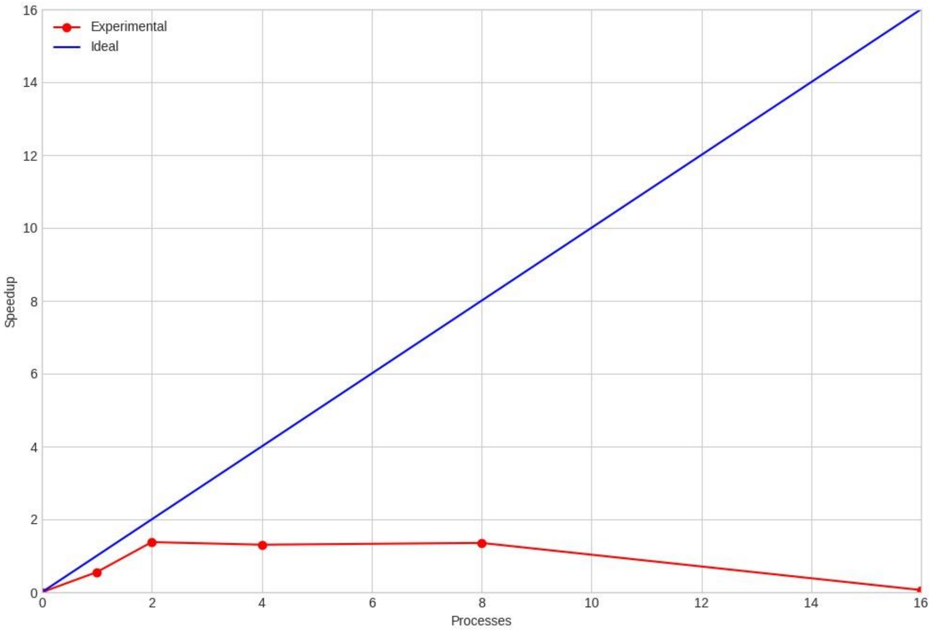
Speedup_timeCount

SIZE-10mln-RANGE-1k-OPT-O2

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.119294214	0.022436563	1	1	1	1
Parallel	1	0.128495364	0.040684667	0.928393141	0.928393141	0.551474655	0.551474655
Parallel	2	0.079925077	0.029563643	1.607697716	0.803848858	1.376172309	0.688086155
Parallel	4	0.061827333	0.031176286	2.078293802	0.51957345	1.304987613	0.326246903
Parallel	8	0.045032611	0.030045111	2.853384702	0.356673088	1.354119361	0.16926492
Parallel	16	0.4512788	0.6675646	0.284736096	0.017796006	0.060944913	0.003809057



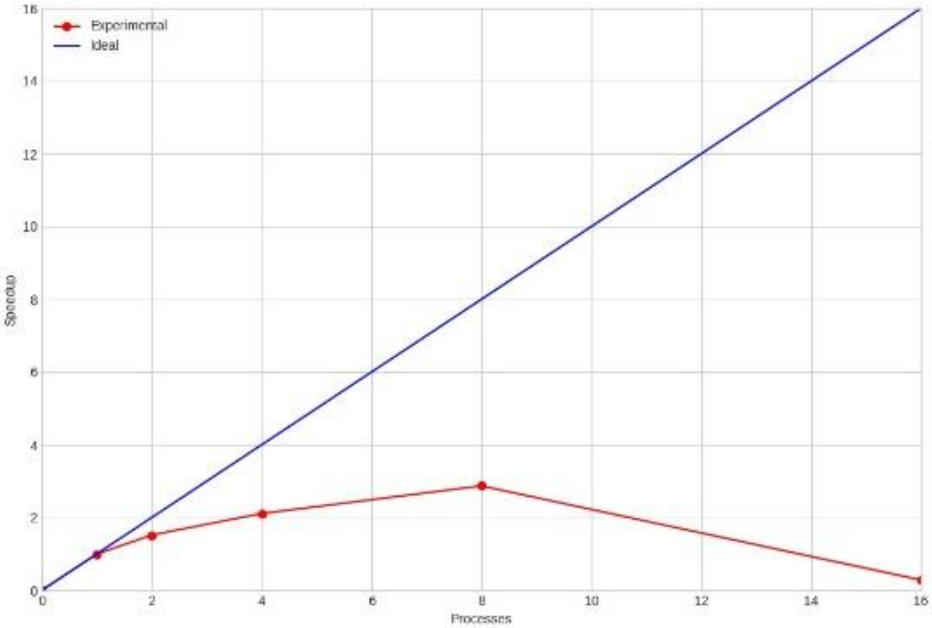
Speedup_timeInit



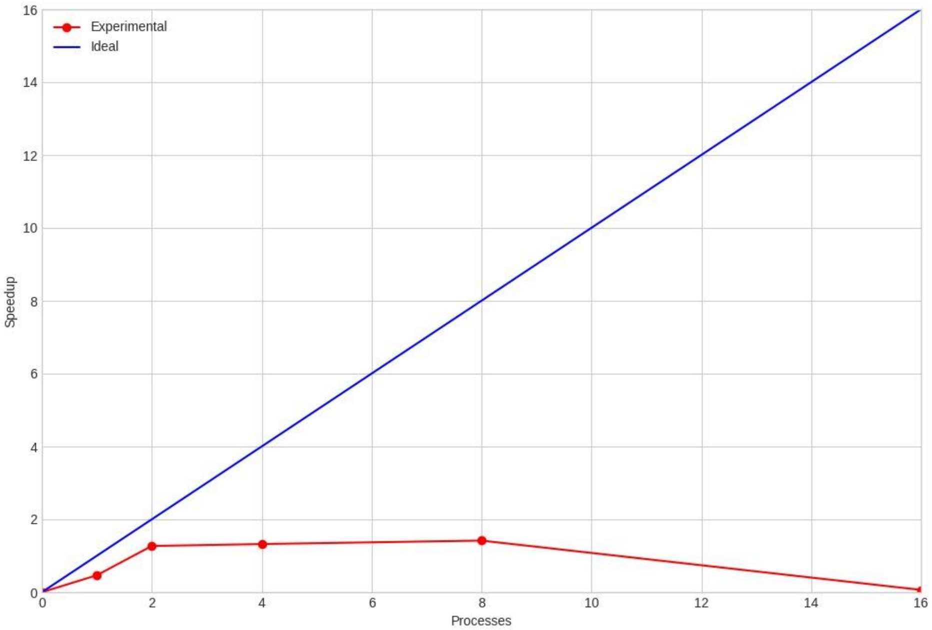
Speedup_timeCount

SIZE-10mln-RANGE-1k-OPT-O3

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.1186336	0.0187055	1	1	1	1
Parallel	1	0.119211176	0.040159333	0.995155014	0.995155014	0.465782134	0.465782134
Parallel	2	0.078471313	0.0316408	1.519168887	0.759584443	1.269226231	0.634613116
Parallel	4	0.056469813	0.030358	2.11106025	0.527765062	1.322858335	0.330714584
Parallel	8	0.0415316	0.028325	2.870372836	0.358796604	1.417805237	0.177225655
Parallel	16	0.421293	0.618553462	0.282965006	0.017685313	0.064924596	0.004057787



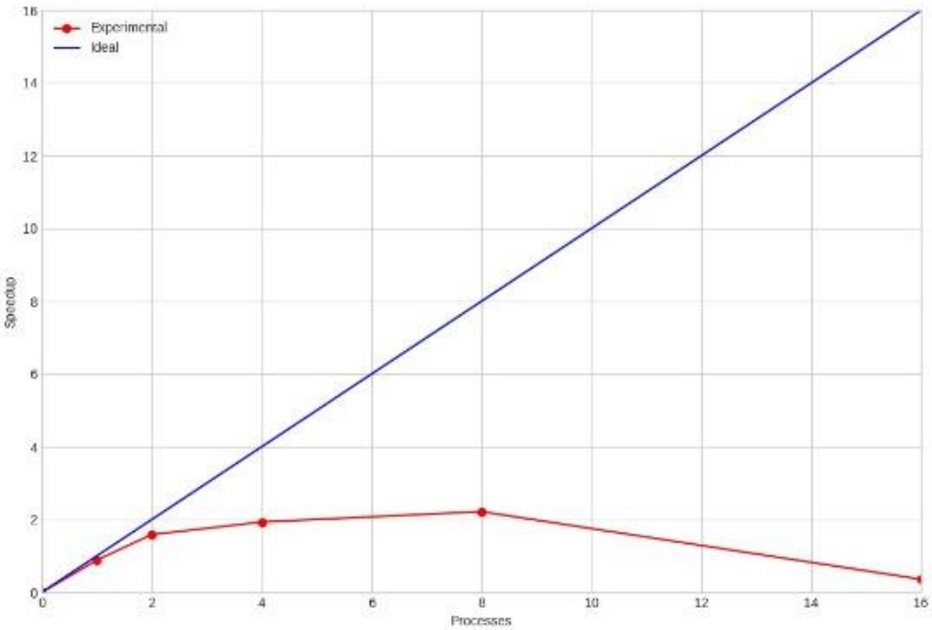
Speedup_timeInit



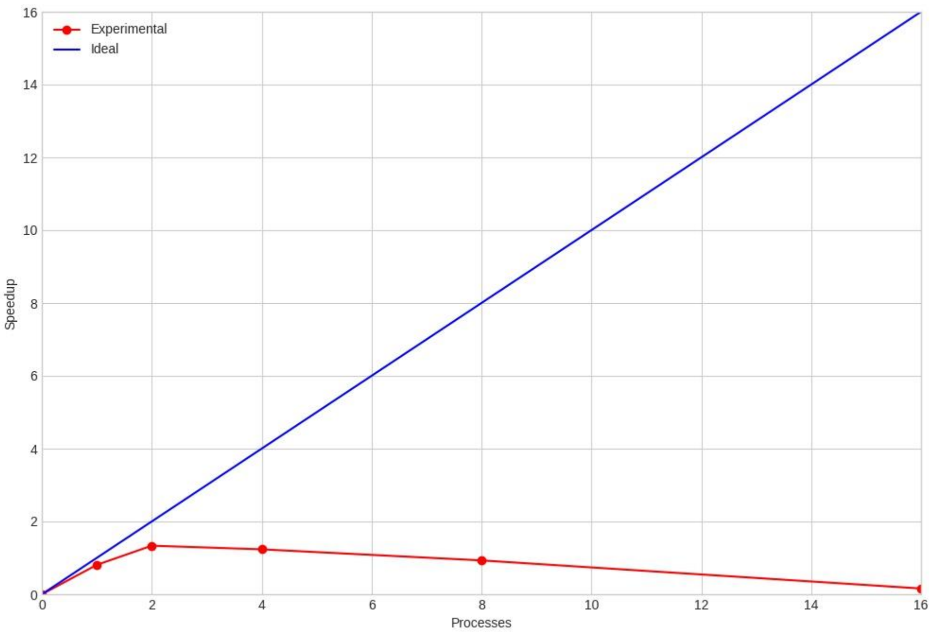
Speedup_timeCount

SIZE-10mln-RANGE-100k-OPT-O0

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.108544933	0.096833118	1	1	1	1
Parallel	1	0.122779294	0.119815889	0.884065462	0.884065462	0.808182609	0.808182609
Parallel	2	0.077380294	0.0897565	1.58669976	0.79334988	1.334899299	0.667449649
Parallel	4	0.063550917	0.097033353	1.931983055	0.482995764	1.234790773	0.308697693
Parallel	8	0.055339316	0.128677308	2.218663031	0.277332879	0.931134565	0.116391821
Parallel	16	0.342593571	0.751510929	0.358381781	0.022398861	0.159433329	0.009964583



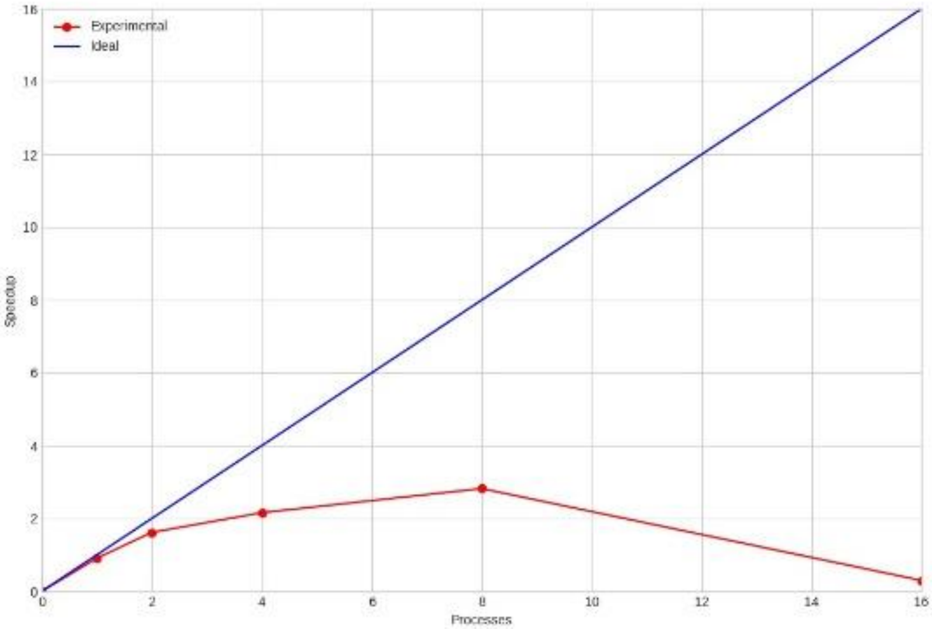
Speedup_timeInit



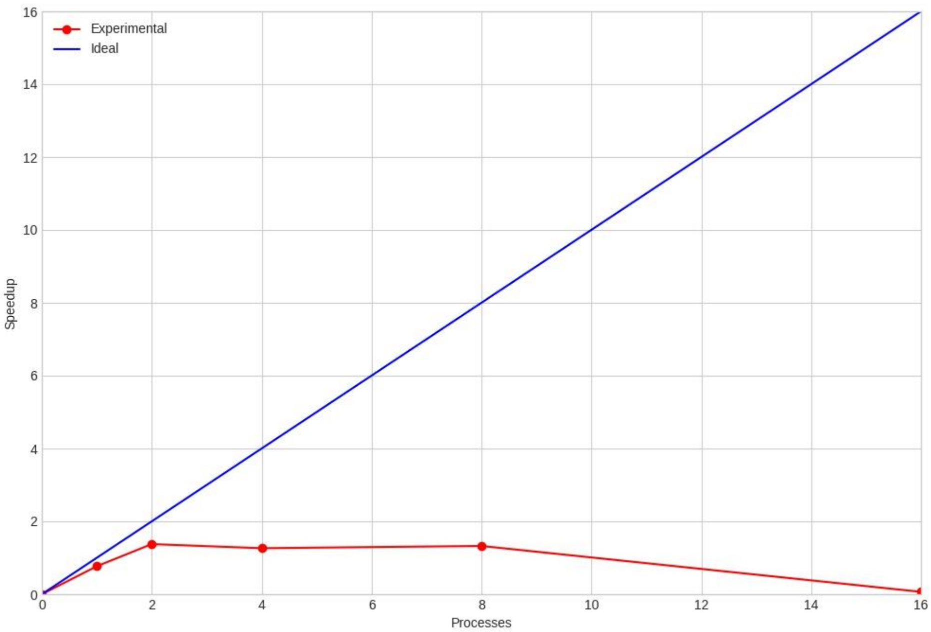
Speedup_timeCount

SIZE-10mln-RANGE-100k-OPT-O1

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.114144	0.039675071	1	1	1	1
Parallel	1	0.1256818	0.0516985	0.908198323	0.908198323	0.767431771	0.767431771
Parallel	2	0.07775	0.037563471	1.616486174	0.808243087	1.376297216	0.688148608
Parallel	4	0.058190385	0.040929917	2.159837933	0.539959483	1.263098101	0.315774525
Parallel	8	0.044548895	0.039023667	2.821210285	0.352651286	1.324798626	0.165599828
Parallel	16	0.422875313	0.784126571	0.297207702	0.018575481	0.06593132	0.004120708



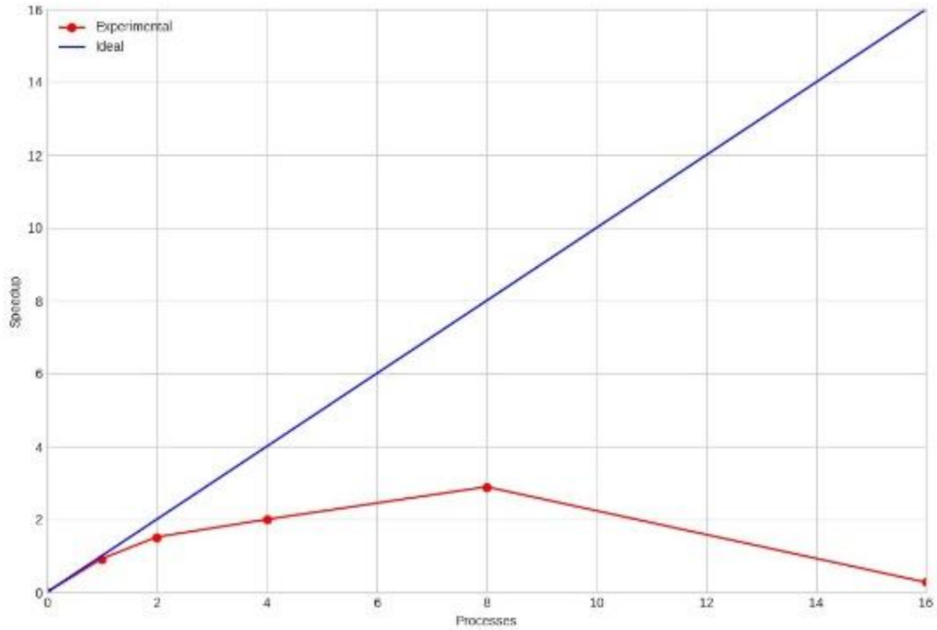
Speedup_timeInit



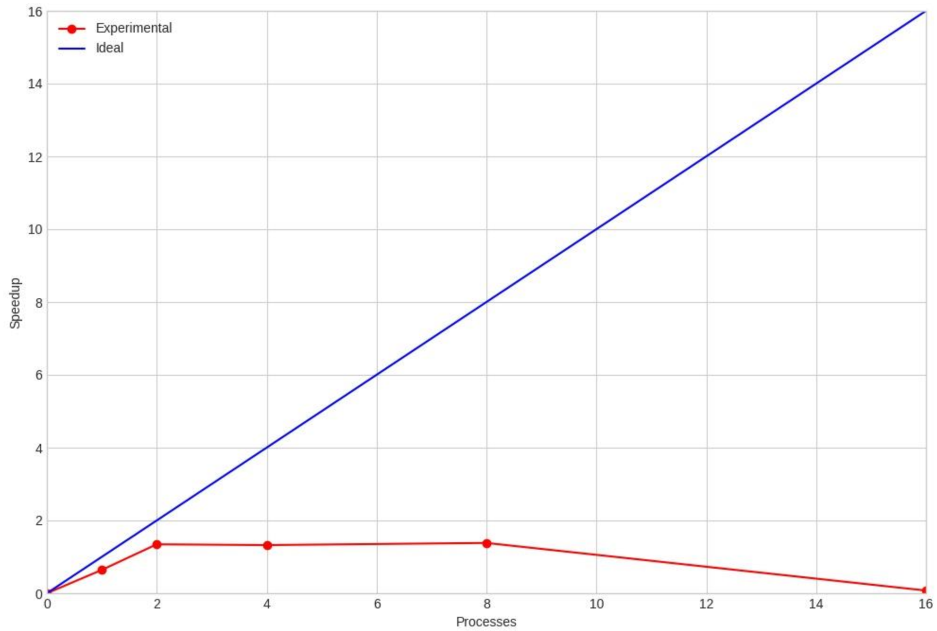
Speedup_timeCount

SIZE-10mln-RANGE-100k-OPT-O2

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.117415714	0.033428071	1	1	1	1
Parallel	1	0.12759	0.052106857	0.920257969	0.920257969	0.64152922	0.64152922
Parallel	2	0.084119813	0.038725765	1.51676515	0.758382575	1.345534621	0.672767311
Parallel	4	0.063804133	0.03936	1.999713707	0.499928427	1.323853078	0.330963269
Parallel	8	0.044101188	0.037673158	2.893119375	0.361639922	1.383129529	0.172891191
Parallel	16	0.467568769	0.689329	0.272879646	0.017054978	0.075590693	0.004724418



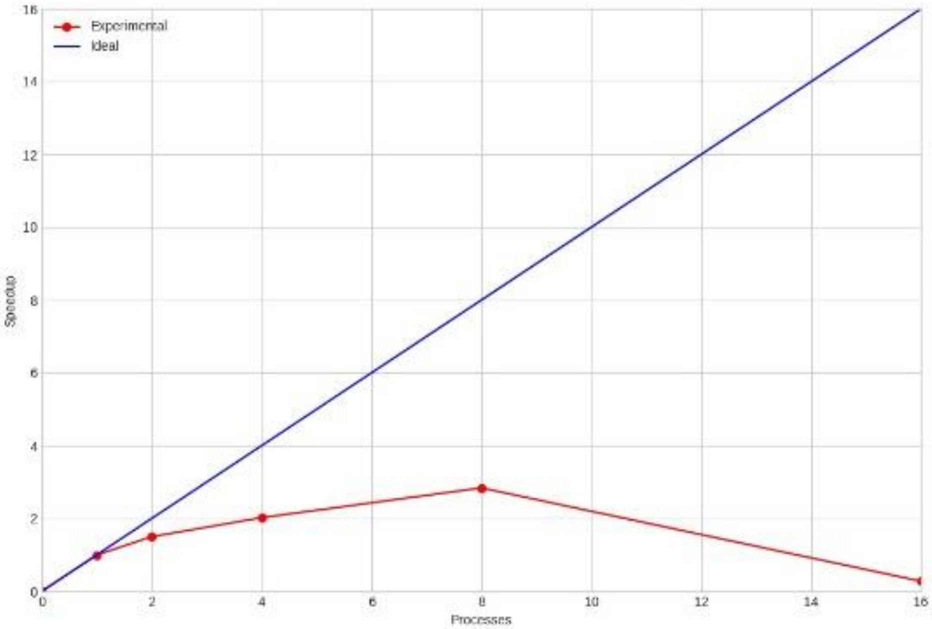
Speedup_timeInit



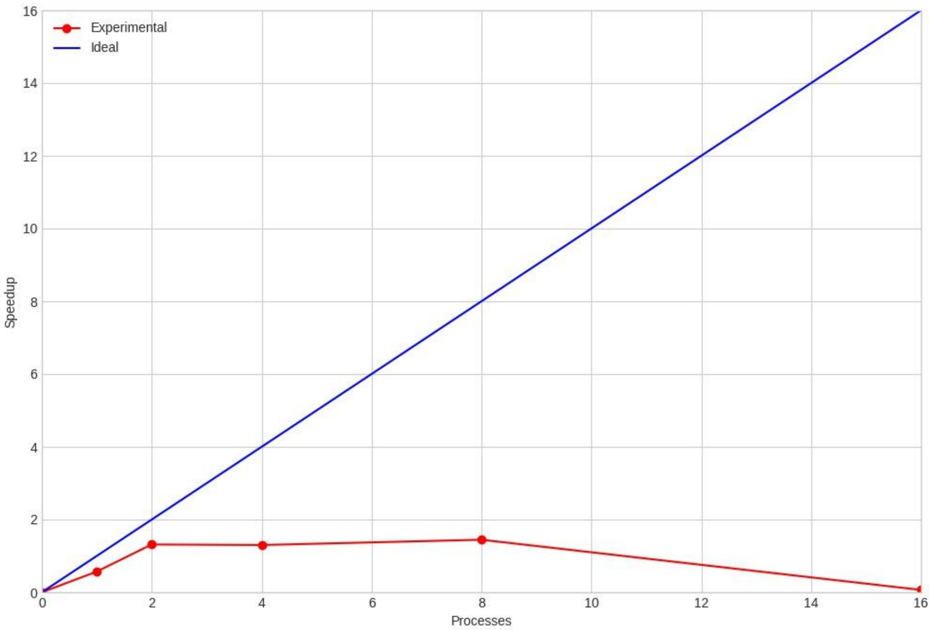
Speedup_timeCount

SIZE-10mln-RANGE-100k-OPT-O3

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.116830083	0.029244077	1	1	1	1
Parallel	1	0.117141375	0.051298588	0.997342598	0.997342598	0.570075667	0.570075667
Parallel	2	0.078114938	0.038999467	1.499602749	0.749801374	1.315366404	0.657683202
Parallel	4	0.057936769	0.039458214	2.021883107	0.505470777	1.30007374	0.325018435
Parallel	8	0.041271105	0.035486375	2.838338694	0.354792337	1.445585474	0.180698184
Parallel	16	0.424126214	0.7762586	0.276194612	0.017262163	0.066084406	0.004130275



Speedup_timeInit



Speedup_timeCount

Case Study n.2

To better understand the following results, describing the logic behind the program is very useful.

The sequential version is easy: it takes three parameters as input, which represent size (unsorted array length), range and the name of the file where to work. Then it calls a function, whose name is *init()* and whose role is initializing randomically the unsorted array and to print it on file. In the end, another function, whose name is *countingSort()*, is called: it reads the unsorted array from the file, sorts it using counting sort algorithm and prints it on the same file, overwriting its previous content.

The parallel version is a bit more complex. There are always three parameters with the same meaning seen above, but before dealing with them it is necessary to initialize the MPI environment. Then every process calls a function named *init()*, which is meant to initialize the unsorted array and to print it on file. This purpose is reached thanks to the following logic: every process must allocate and randomically initialize an array called **piece_init_array**. After that the file must be opened and the primitive *MPI_File_set_view()* must be called properly, so that each process can write its **piece_init_array** on file.

Finished the initialization, it is counting sort turn: every process calls a function named *countingSort()*. The full unsorted array previously written on file must be read in a distributed way by the processes: each of them reads a specific file portion (different from the ones read by the others) and saves its content into an array called **piece_of_array**. The i-th process computes the minimum and the maximum of its **piece_of_array**. These results are local so a reduction is compulsory to get the global maximum (**max**) and the global minimum (**min**). After that, the i-th process must allocate an auxiliary array named **c_local**, whose length is equal to $\boldsymbol{max - min + 1}$.

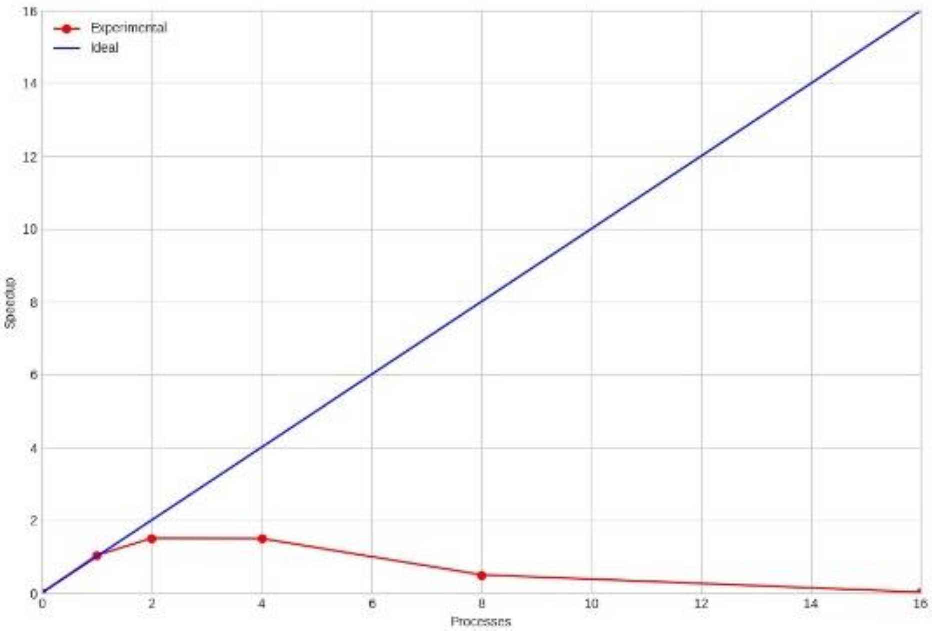
Three for-loops are executed: the first one initializes to 0 every **c_local** element; the second one increments by 1 the **c_local** element placed in the position corresponding to the visited **piece_of_array** element; the third one increments every **c_local** element by adding to it the sum of all its previous elements in **c_local**. After that, all the different **c_local** must be summed together through a reduction: the result is represented by another array, whose name is **c**, allocated only by rank 0 process. Afterwards, the process with rank 0 allocates an array **full_array** intended to contain the sorted array. Counting sort is completed executing two for-loops that exploit **c** content to fill **full_array**. In the end, **full_array** is printed on file, overwriting its previous content, by the process with rank 0.

Just like in Case Study n.1, it is important to observe that the job of measuring elapsed time is assigned to the process with rank 0.

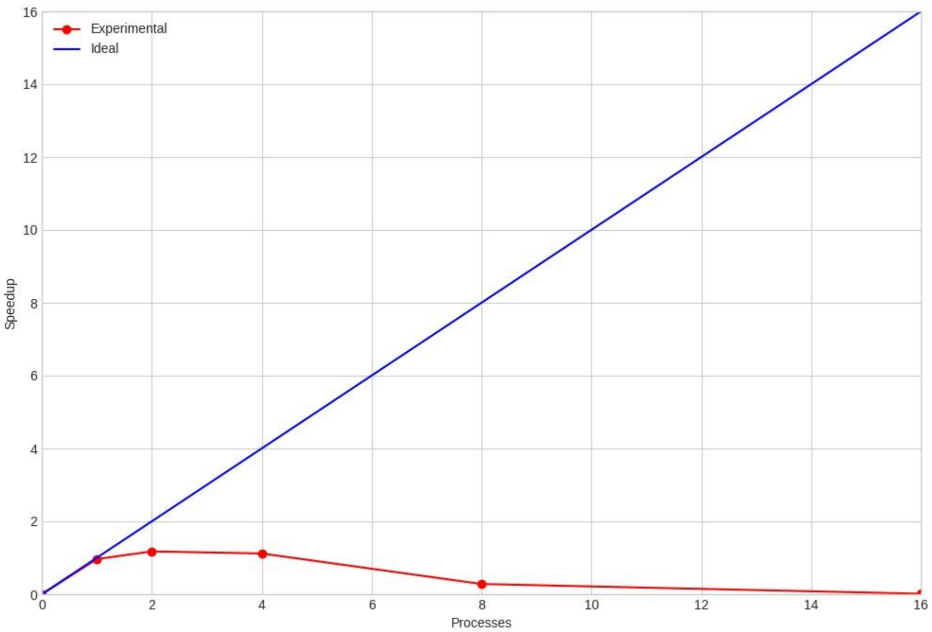
The following graphs are obtained considering not only different sizes and ranges, but also different gcc optimizations.

SIZE-1mln-RANGE-1k-OPT-O0

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.016469846	0.010816941	1	1	1	1
Parallel	1	0.015928111	0.011214789	1.034011255	1.034011255	0.964524675	0.964524675
Parallel	2	0.010580067	0.0095345	1.505483057	0.752741529	1.176232574	0.588116287
Parallel	4	0.010635667	0.010020941	1.497612854	0.374403214	1.119135346	0.279783837
Parallel	8	0.03165125	0.039823615	0.50323798	0.062904747	0.281611535	0.035201442
Parallel	16	0.680192308	0.805559786	0.02341707	0.001463567	0.013921735	0.000870108



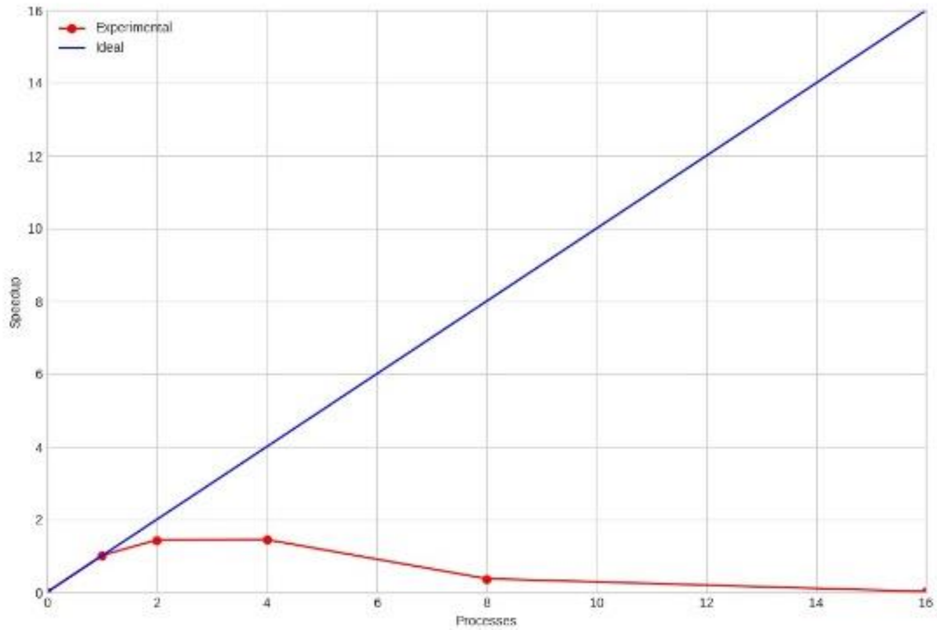
Speedup_timeInit



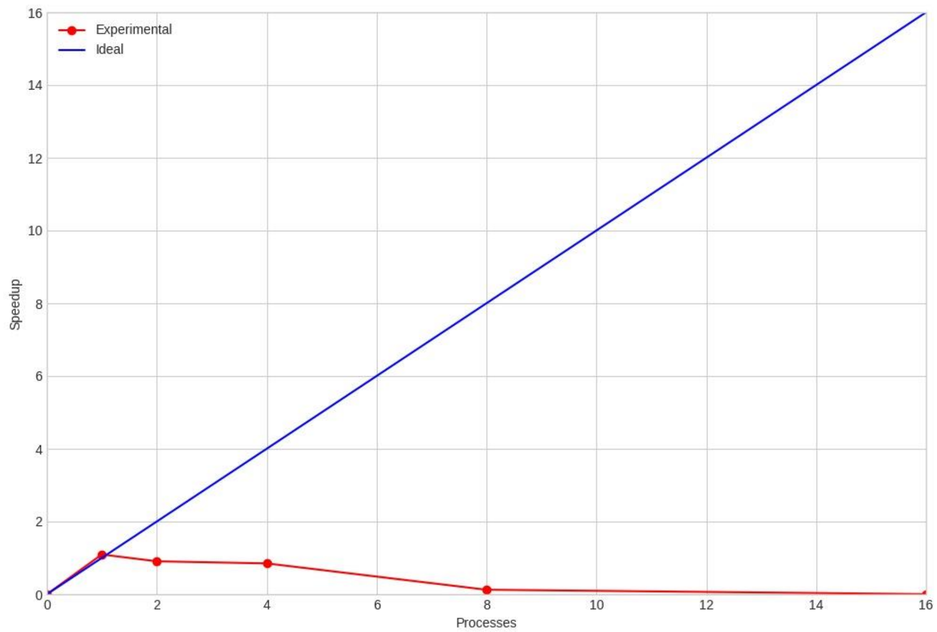
Speedup_timeCount

SIZE-1mln-RANGE-1k-OPT-O1

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.014310933	0.004776867	1	1	1	1
Parallel	1	0.014100071	0.004362286	1.014954669	1.014954669	1.095037551	1.095037551
Parallel	2	0.009808667	0.0048045	1.43751153	0.718755765	0.907958313	0.453979156
Parallel	4	0.009736071	0.005134375	1.448230072	0.362057518	0.849623511	0.212405878
Parallel	8	0.037487538	0.034594471	0.376126895	0.047015862	0.126097773	0.015762222
Parallel	16	0.704524143	0.912850231	0.02001361	0.001250851	0.004778753	0.000298672



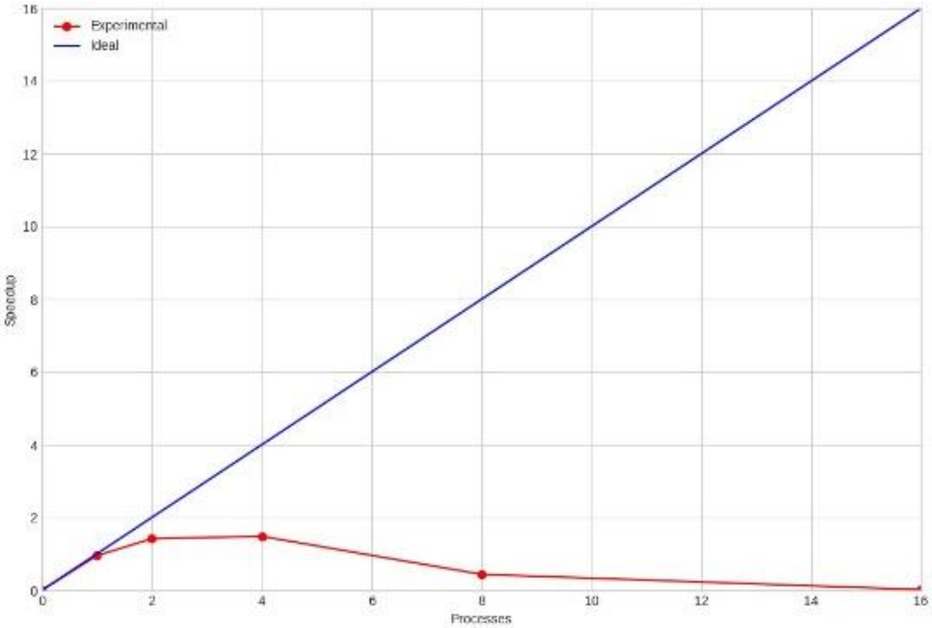
Speedup_timeInit



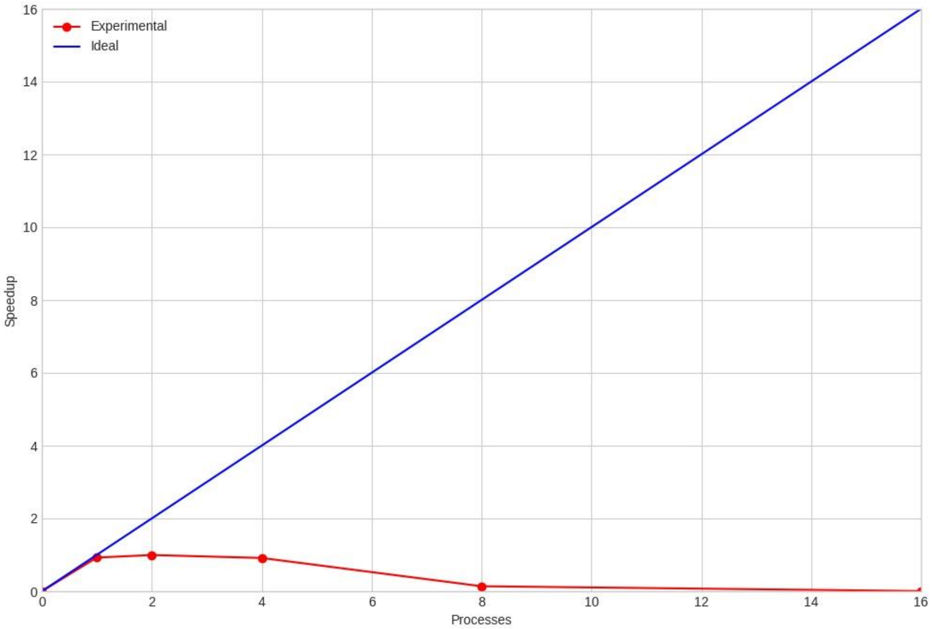
Speedup_timeCount

SIZE-1mln-RANGE-1k-OPT-O2

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.013857722	0.004604471	1	1	1	1
Parallel	1	0.014556143	0.004972316	0.952018839	0.952018839	0.926021352	0.926021352
Parallel	2	0.010263667	0.004992059	1.418220538	0.709110269	0.996045112	0.498022556
Parallel	4	0.00980225	0.005431786	1.48497976	0.37124494	0.915410889	0.228852722
Parallel	8	0.0331384	0.0358116	0.439253037	0.05490663	0.138846513	0.017355814
Parallel	16	0.761085125	0.881368538	0.019125512	0.001195345	0.005641585	0.000352599



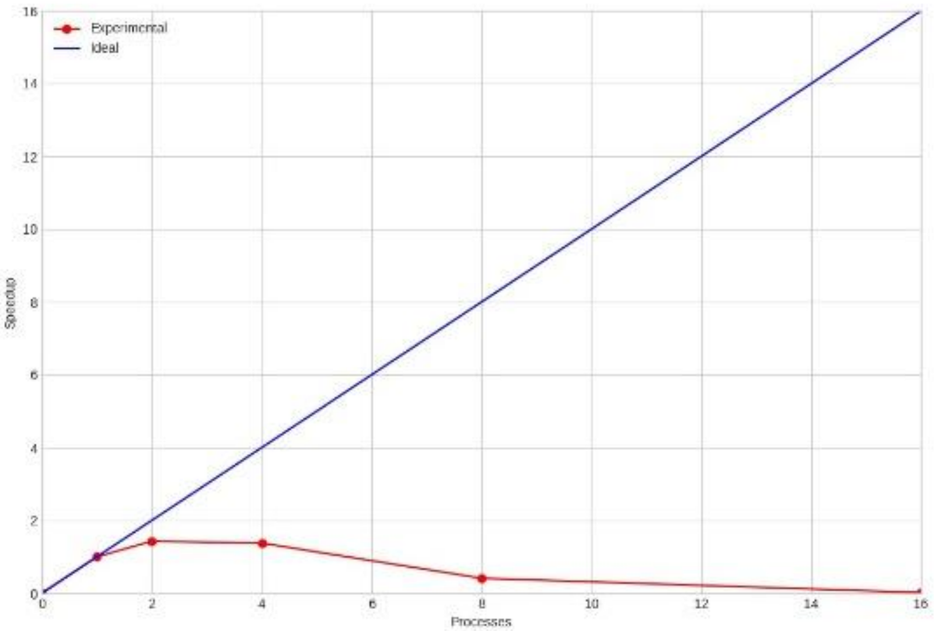
Speedup_timeInit



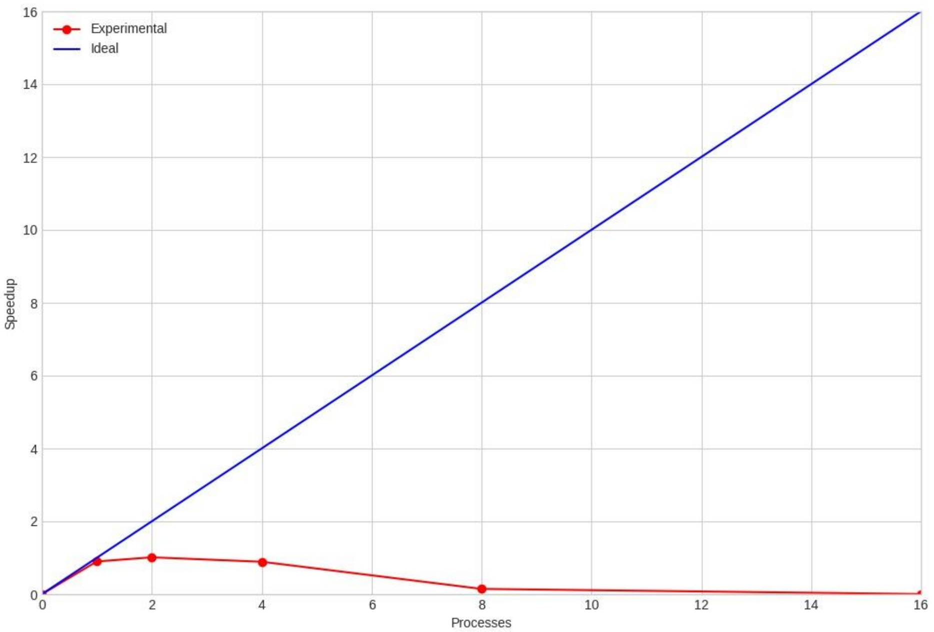
Speedup_timeCount

SIZE-1mln-RANGE-1k-OPT-O3

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.0142249	0.004237938	1	1	1	1
Parallel	1	0.014215667	0.004711769	1.000649518	1.000649518	0.899436558	0.899436558
Parallel	2	0.009938867	0.004650133	1.430310632	0.715155316	1.013254652	0.506627326
Parallel	4	0.010312923	0.005281938	1.378432338	0.344608084	0.892053197	0.223013299
Parallel	8	0.03437325	0.031976125	0.413567721	0.051695965	0.147352727	0.018419091
Parallel	16	0.704009733	0.878848462	0.020192429	0.001262027	0.005361299	0.000335081



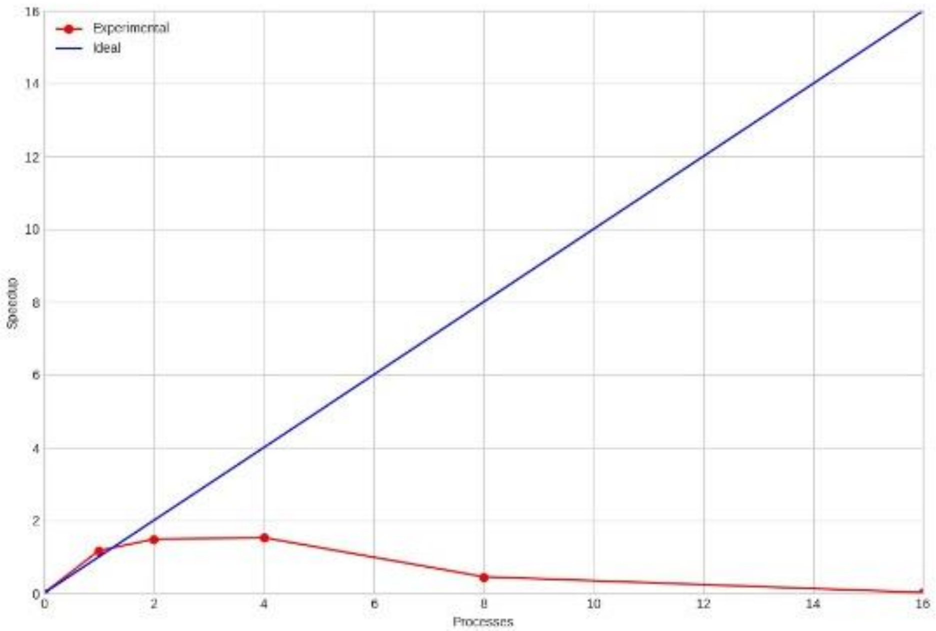
Speedup_timeInit



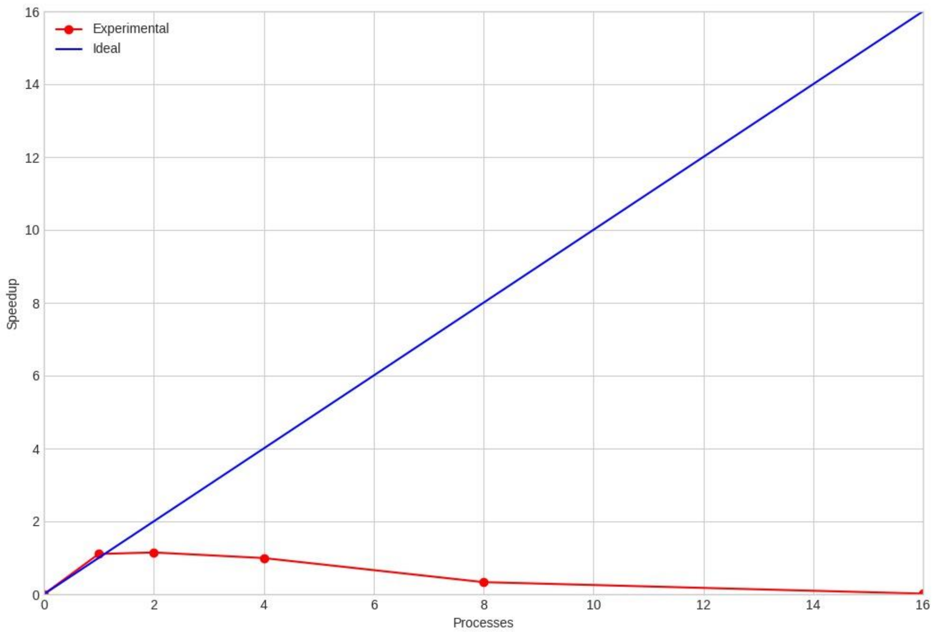
Speedup_timeCount

SIZE-1mln-RANGE-100k-OPT-O0

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.018609429	0.017380538	1	1	1	1
Parallel	1	0.015895	0.015707529	1.17077248	1.17077248	1.106510006	1.106510006
Parallel	2	0.010705889	0.013692533	1.484696896	0.742348448	1.14716021	0.573580105
Parallel	4	0.010393071	0.015806	1.529384274	0.382346068	0.99377005	0.248442513
Parallel	8	0.034995	0.047248077	0.454207744	0.056775968	0.332448016	0.041556002
Parallel	16	0.724912231	0.926211357	0.021926792	0.001370425	0.016958904	0.001059931



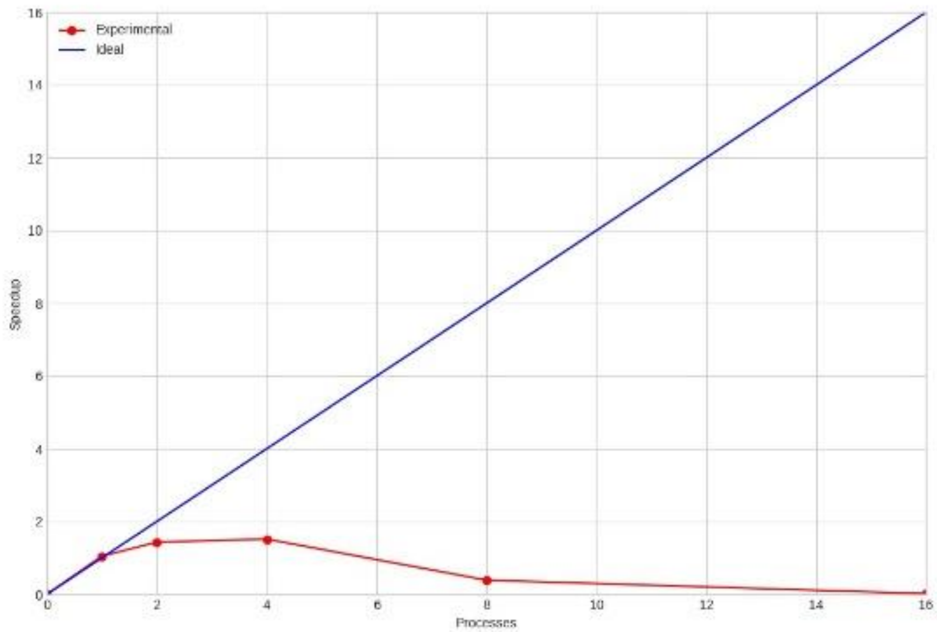
Speedup_timeInit



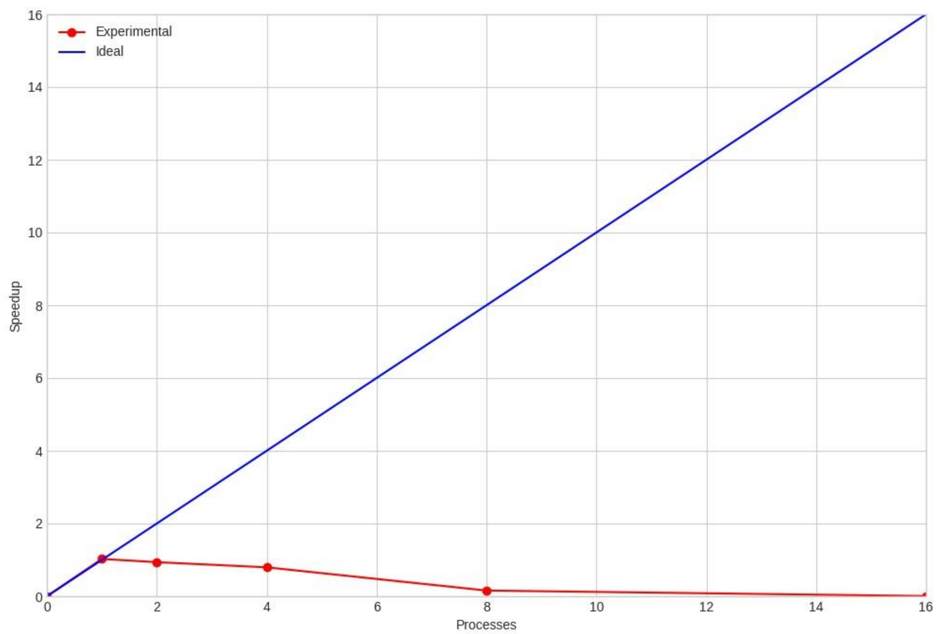
Speedup_timeCount

SIZE-1mln-RANGE-100k-OPT-O1

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.014784308	0.006963308	1	1	1	1
Parallel	1	0.0141105	0.006778438	1.047752219	1.047752219	1.027273275	1.027273275
Parallel	2	0.009846765	0.007216353	1.433008752	0.716504376	0.939316238	0.469658119
Parallel	4	0.009292214	0.008505778	1.518529337	0.379632334	0.796921537	0.199230384
Parallel	8	0.036223077	0.042600786	0.389544489	0.048693061	0.159115316	0.019889415
Parallel	16	0.747990167	0.969228615	0.018864553	0.001179035	0.006993642	0.000437103



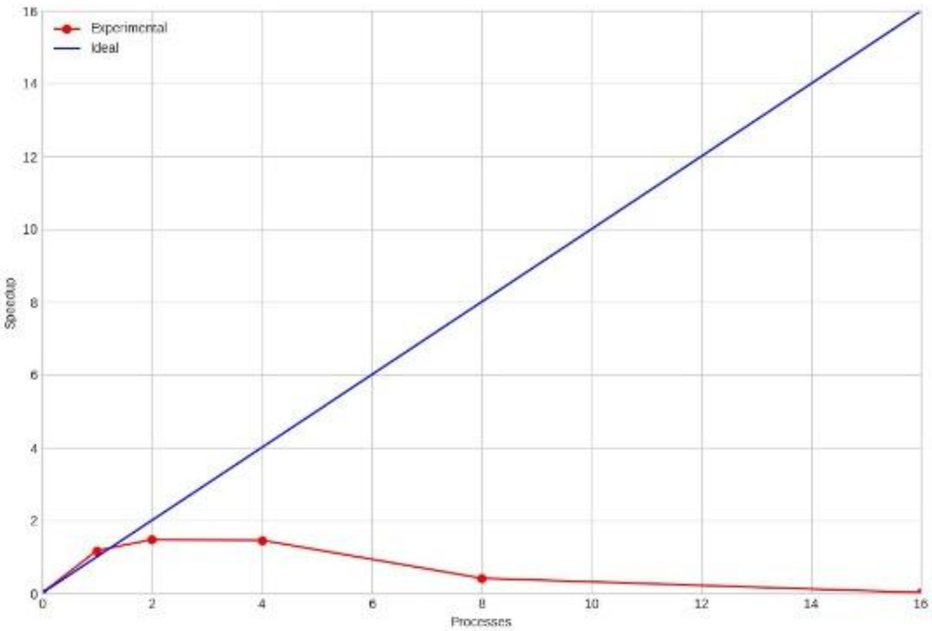
Speedup_timeInit



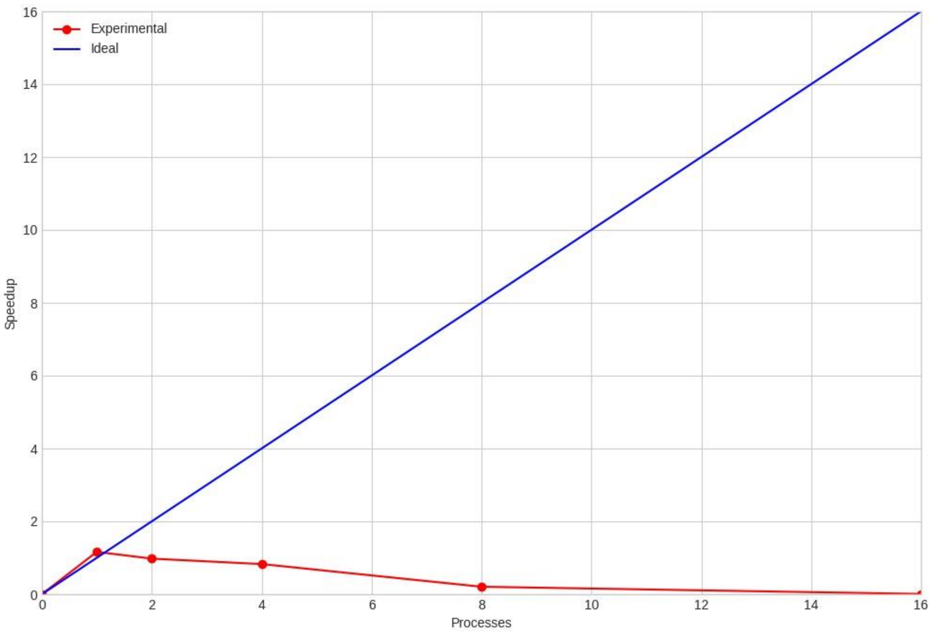
Speedup_timeCount

SIZE-1mln-RANGE-100k-OPT-O2

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.017100846	0.0087944	1	1	1	1
Parallel	1	0.014585	0.00756	1.172495451	1.172495451	1.163280423	1.163280423
Parallel	2	0.009901556	0.007731833	1.473000875	0.736500438	0.977775862	0.488887931
Parallel	4	0.010006929	0.009128071	1.457490168	0.364372542	0.828214378	0.207053594
Parallel	8	0.035227938	0.036601	0.414017994	0.051752249	0.206551734	0.025818967
Parallel	16	0.736563733	0.893832667	0.019801409	0.001237588	0.008457959	0.000528622



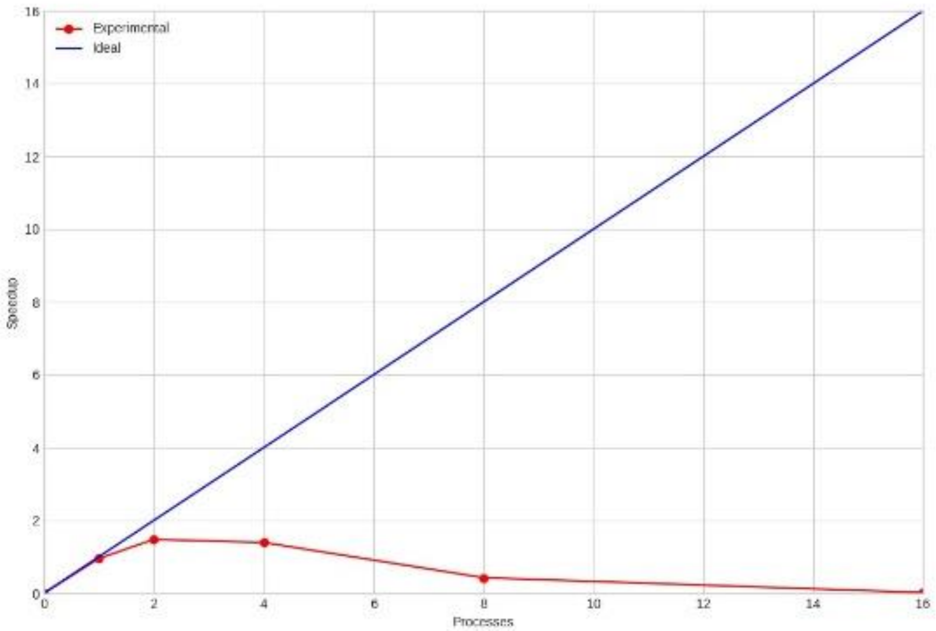
Speedup_timeInit



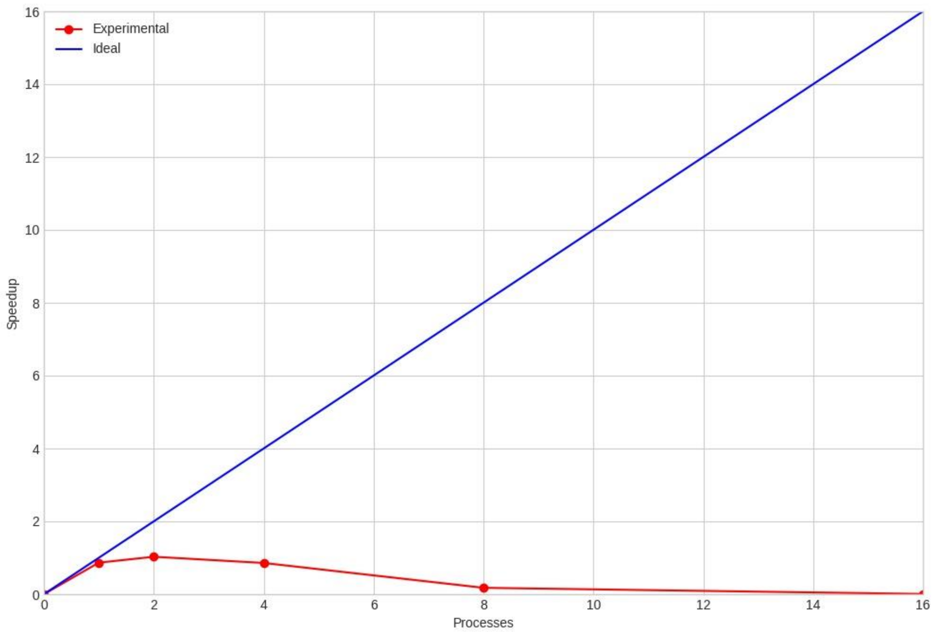
Speedup_timeCount

SIZE-1mln-RANGE-100k-OPT-O3

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.014114611	0.006521444	1	1	1	1
Parallel	1	0.01478825	0.007522125	0.954447694	0.954447694	0.866968369	0.866968369
Parallel	2	0.009999588	0.007308471	1.478885895	0.739442948	1.029233806	0.514616903
Parallel	4	0.010595643	0.008759	1.395691625	0.348922906	0.858788104	0.214697026
Parallel	8	0.034519692	0.042448875	0.4284004	0.05355005	0.177204343	0.022150543
Parallel	16	0.714214692	0.984639833	0.020705609	0.001294101	0.007639469	0.000477467



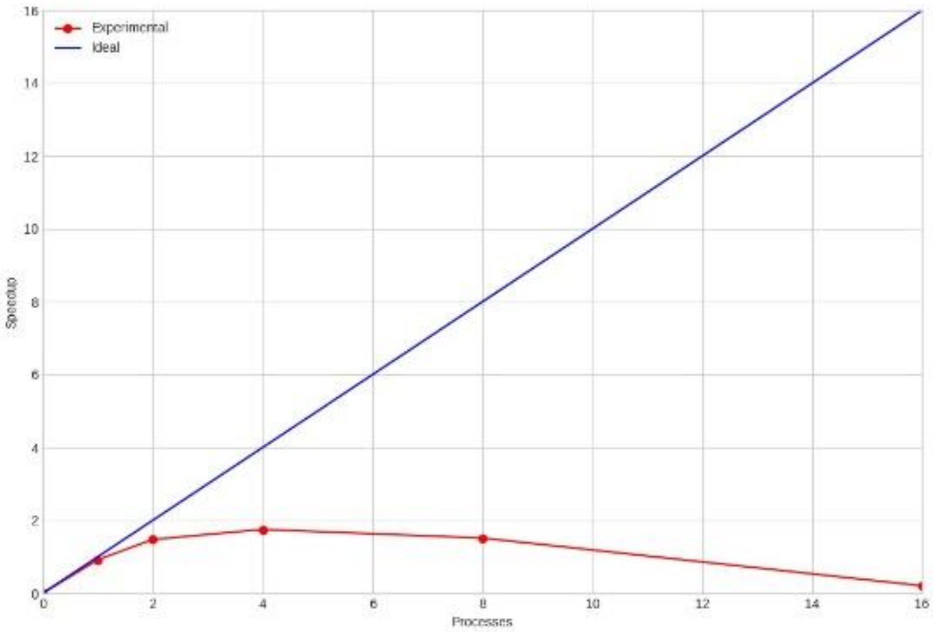
Speedup_timeInit



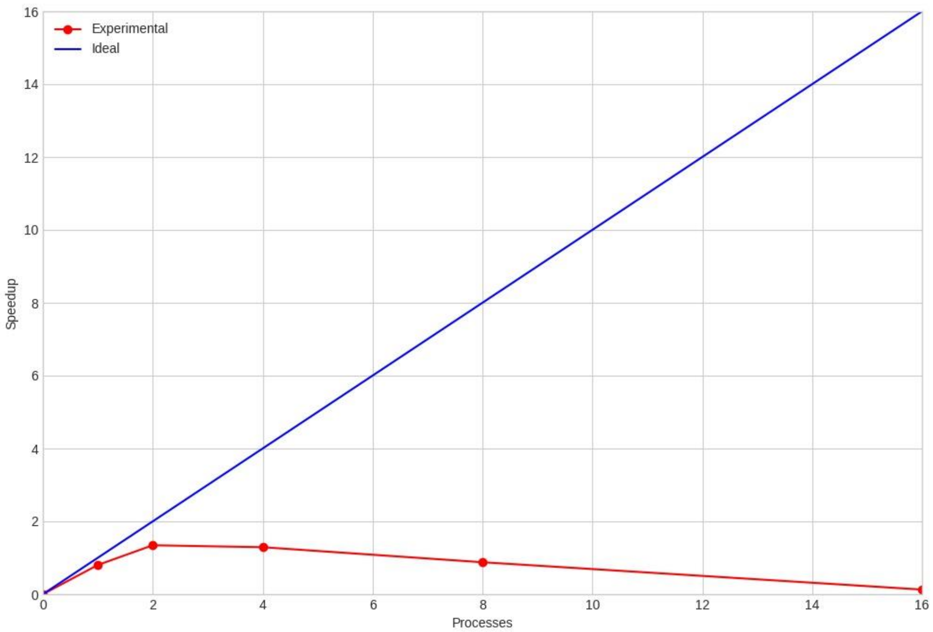
Speedup_timeCount

SIZE-10mln-RANGE-1k-OPT-O0

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.145362182	0.100373467	1	1	1	1
Parallel	1	0.158185688	0.125047867	0.918933844	0.918933844	0.80268036	0.80268036
Parallel	2	0.107274462	0.0930085	1.474588502	0.737294251	1.344477834	0.672238917
Parallel	4	0.090254833	0.096873471	1.752656137	0.438164034	1.290837067	0.322709267
Parallel	8	0.104405474	0.14228	1.515109141	0.189388643	0.878885765	0.109860721
Parallel	16	0.768422462	0.971588333	0.205857709	0.012866107	0.128704578	0.008044036



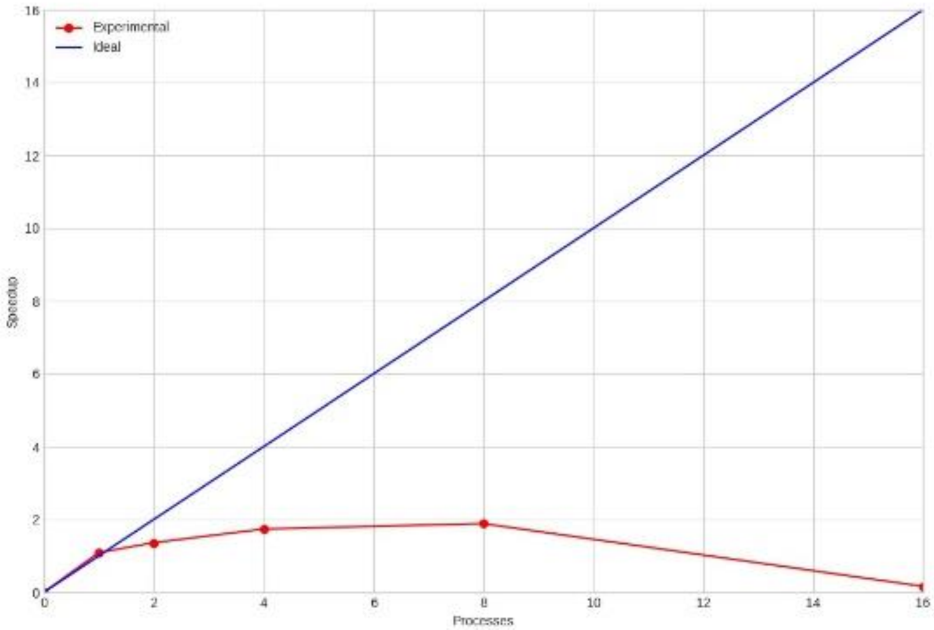
Speedup_timeInit



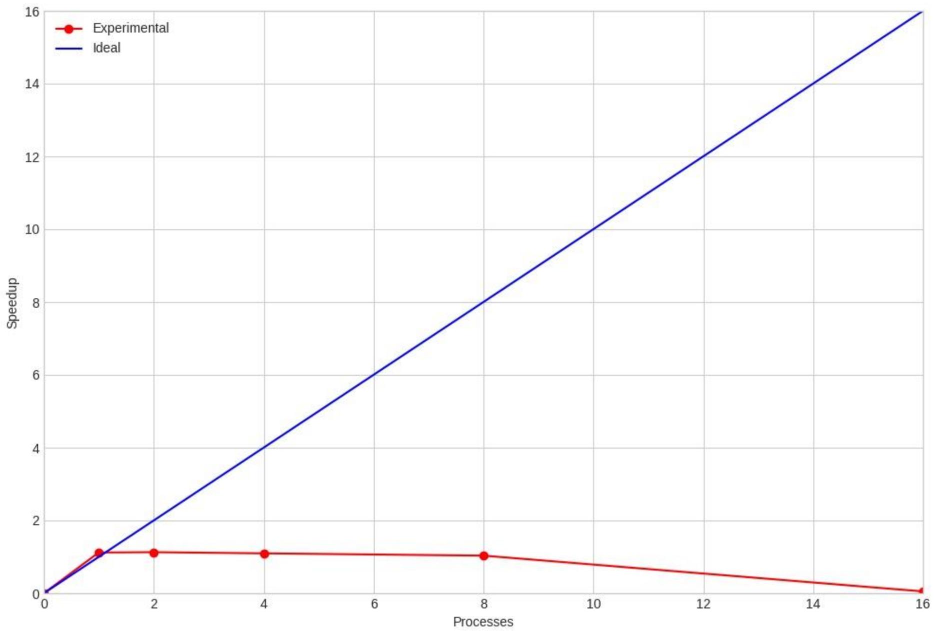
Speedup_timeCount

SIZE-10mln-RANGE-1k-OPT-O1

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.147400105	0.054926579	1	1	1	1
Parallel	1	0.135232222	0.049037167	1.089977691	1.089977691	1.12010099	1.12010099
Parallel	2	0.099264563	0.043406933	1.362341392	0.681170696	1.129708157	0.564854079
Parallel	4	0.077773643	0.044709556	1.73879244	0.43469811	1.096793875	0.274198469
Parallel	8	0.071692353	0.047419667	1.886285171	0.235785646	1.03411032	0.12926379
Parallel	16	0.833464692	0.95312	0.162253091	0.010140818	0.0514491	0.003215569



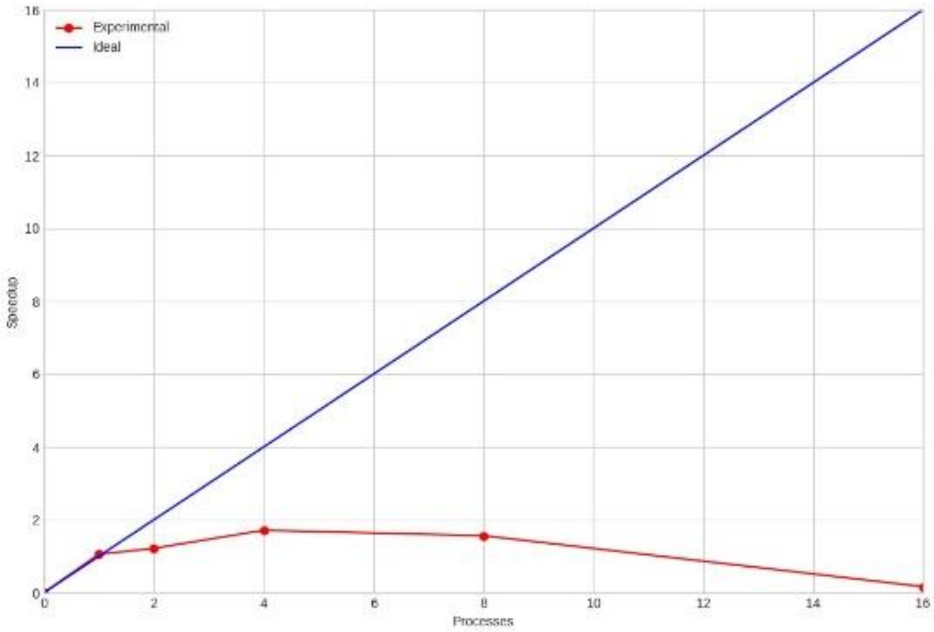
Speedup_timeInit



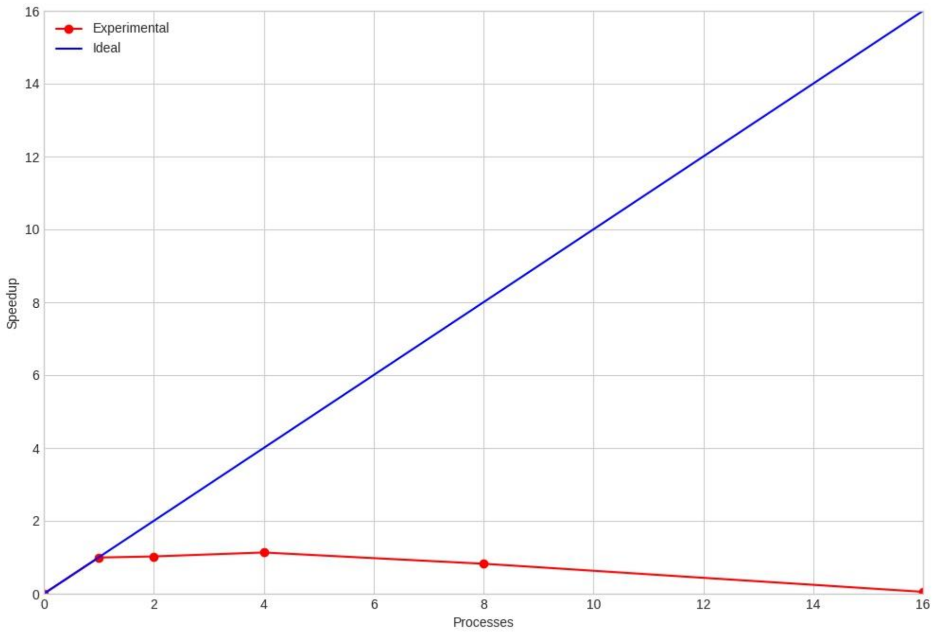
Speedup_timeCount

SIZE-10mln-RANGE-1k-OPT-O2

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.142755375	0.053797722	1	1	1	1
Parallel	1	0.135042556	0.054134188	1.057113992	1.057113992	0.993784606	0.993784606
Parallel	2	0.110964471	0.052874	1.216989139	0.60849457	1.023833784	0.511916892
Parallel	4	0.078709	0.0478116	1.715719366	0.428929841	1.132239613	0.283059903
Parallel	8	0.086187778	0.065716235	1.566841135	0.195855142	0.823756675	0.102969584
Parallel	16	0.803596	1.037303727	0.168047819	0.010502989	0.052187403	0.003261713



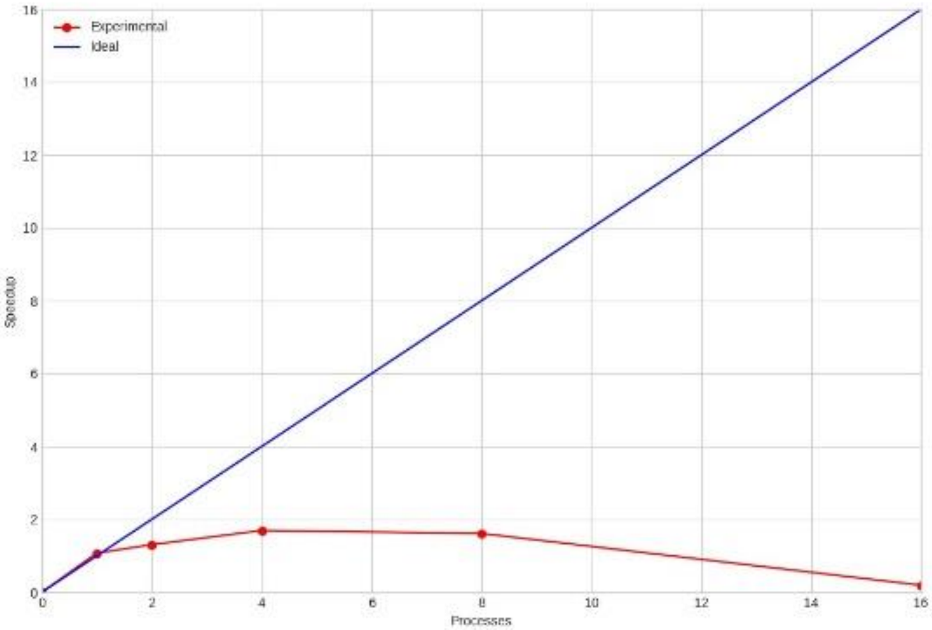
Speedup_timeInit



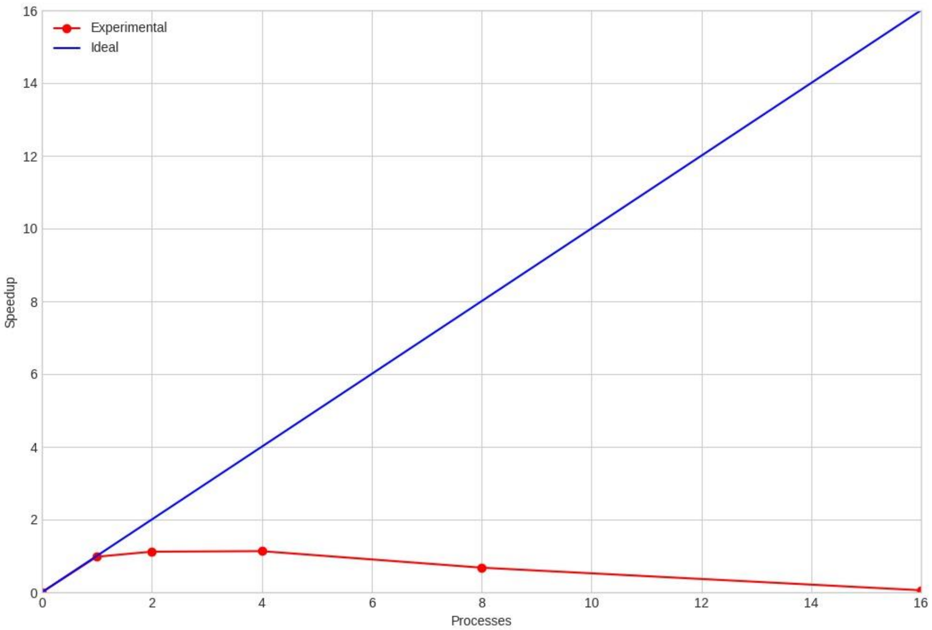
Speedup_timeCount

SIZE-10mln-RANGE-1k-OPT-O3

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.14796275	0.050122	1	1	1	1
Parallel	1	0.137556778	0.051276647	1.07564856	1.07564856	0.977482009	0.977482009
Parallel	2	0.105395231	0.045896882	1.305151825	0.652575912	1.117214164	0.558607082
Parallel	4	0.081086	0.0453094	1.696430676	0.424107669	1.13169998	0.282924995
Parallel	8	0.085241278	0.0756659	1.613734348	0.201716793	0.677671805	0.084708976
Parallel	16	0.709711615	0.912299571	0.193820666	0.012113792	0.056205931	0.003512871



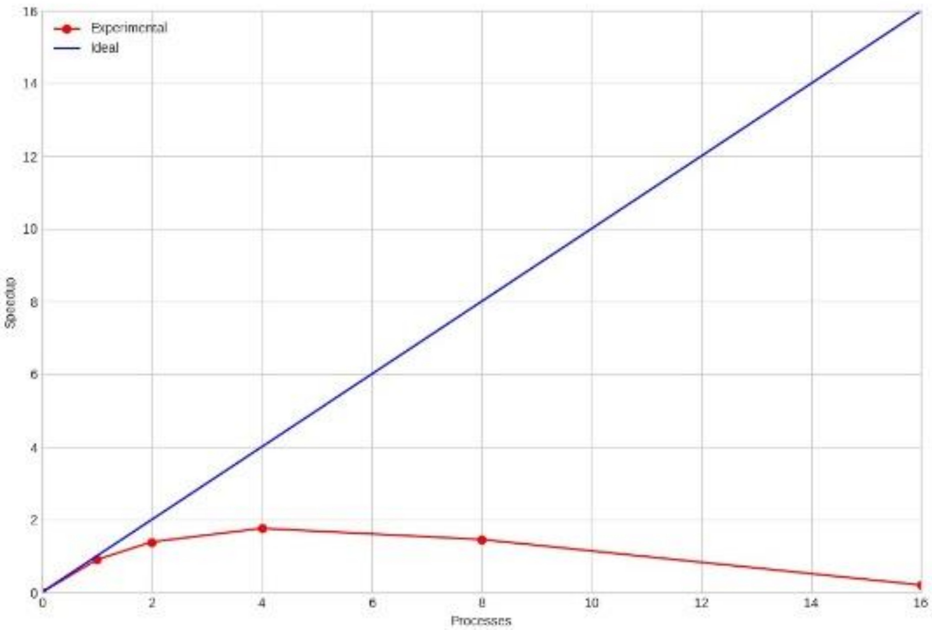
Speedup_timeInit



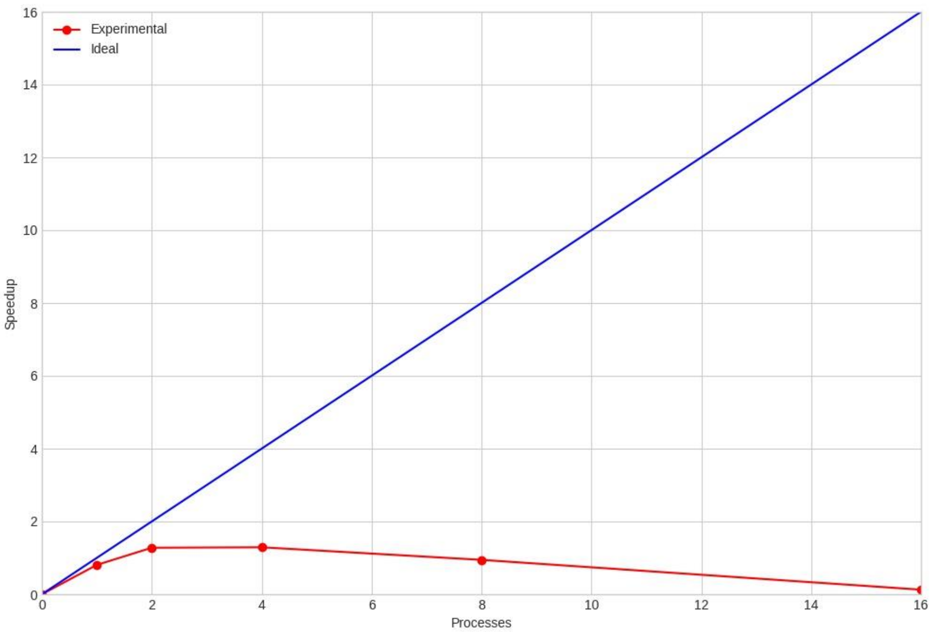
Speedup_timeCount

SIZE-10mln-RANGE-100k-OPT-O0

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.138033667	0.1186145	1	1	1	1
Parallel	1	0.153756412	0.1469855	0.897742508	0.897742508	0.806980961	0.806980961
Parallel	2	0.110834263	0.114937077	1.387264257	0.693632129	1.278834506	0.639417253
Parallel	4	0.087246	0.113838273	1.762331932	0.440582983	1.291178235	0.322794559
Parallel	8	0.105727647	0.155315824	1.454268737	0.181783592	0.946365262	0.118295658
Parallel	16	0.761761188	1.155428231	0.201843326	0.012615208	0.127213007	0.007950813



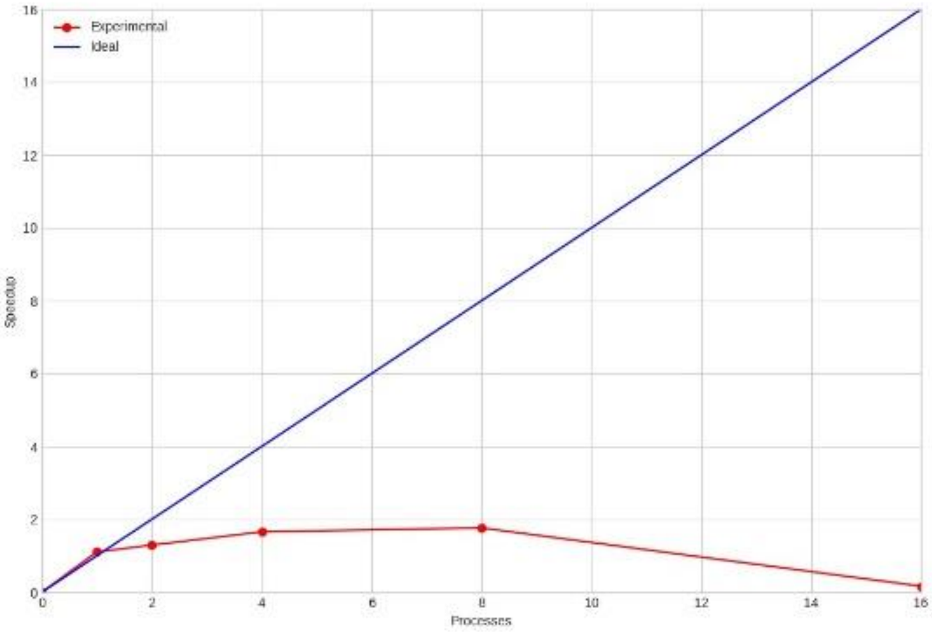
Speedup_timeInit



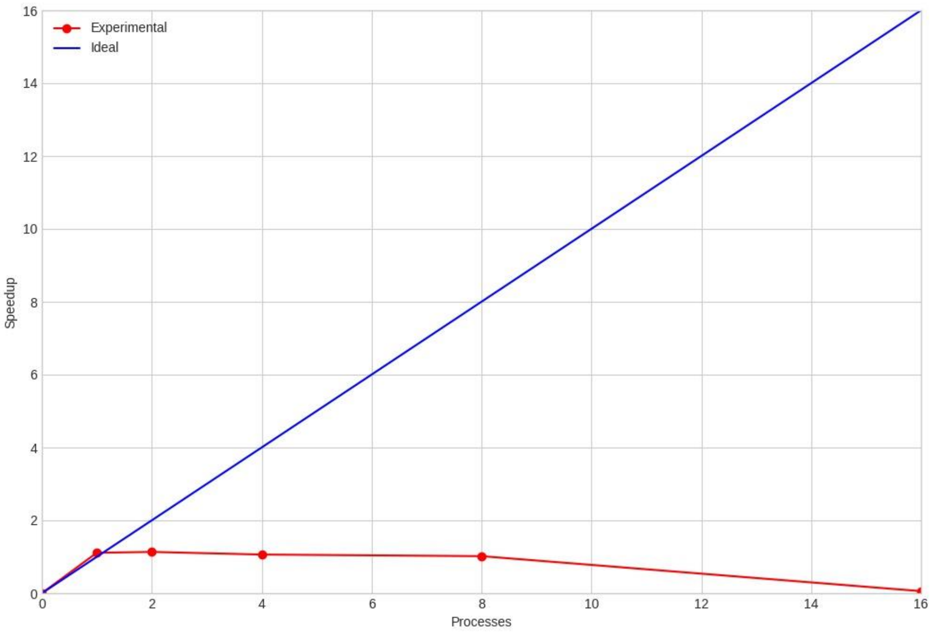
Speedup_timeCount

SIZE-10mln-RANGE-100k-OPT-O1

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.149719294	0.067644056	1	1	1	1
Parallel	1	0.1353196	0.060869063	1.106412479	1.106412479	1.111304377	1.111304377
Parallel	2	0.104616714	0.053646529	1.293479736	0.646739868	1.13463188	0.56731594
Parallel	4	0.081555588	0.057304143	1.659231488	0.414807872	1.062210505	0.265552626
Parallel	8	0.076565706	0.059778625	1.767365669	0.220920709	1.018241261	0.127280158
Parallel	16	0.812736769	1.050187286	0.166498681	0.010406168	0.057960198	0.003622512



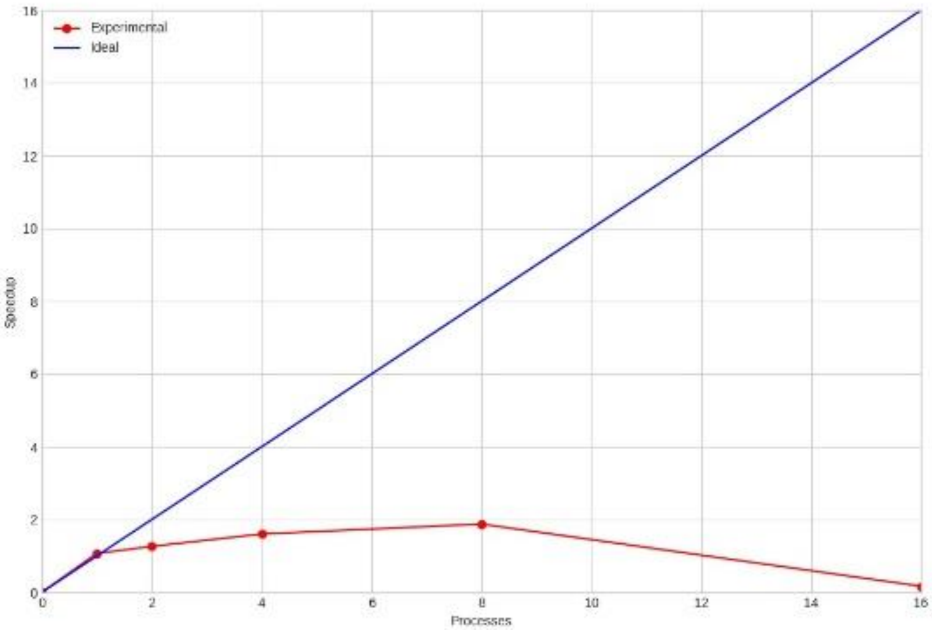
Speedup_timeInit



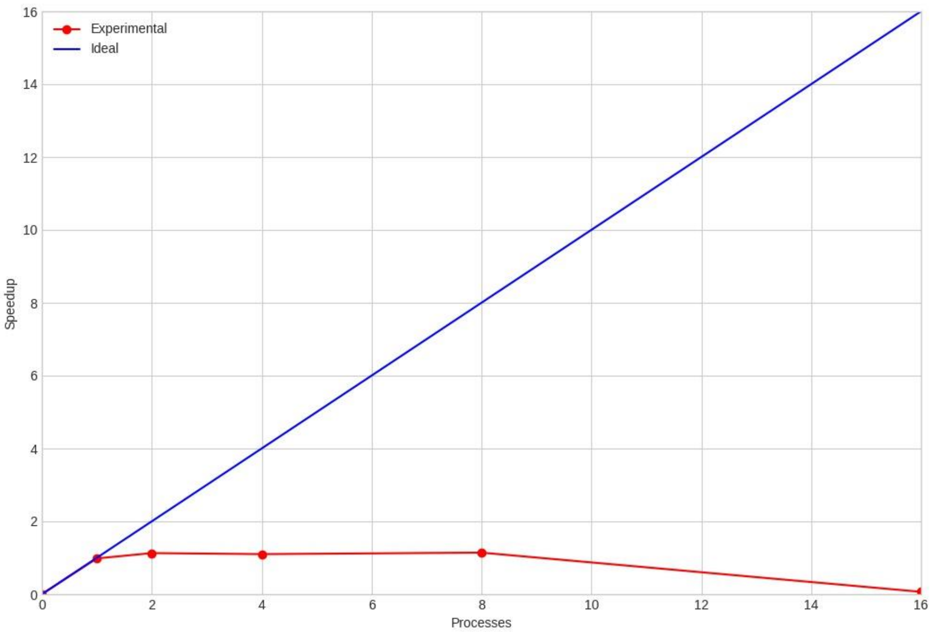
Speedup_timeCount

SIZE-10mln-RANGE-100k-OPT-O2

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.145158067	0.066433214	1	1	1	1
Parallel	1	0.135908313	0.067595889	1.068058782	1.068058782	0.982799626	0.982799626
Parallel	2	0.107703643	0.059828059	1.26187294	0.63093647	1.129835903	0.564917952
Parallel	4	0.084749231	0.061358857	1.603652461	0.400913115	1.101648434	0.275412109
Parallel	8	0.072430611	0.059167824	1.876393288	0.234549161	1.14244339	0.142805424
Parallel	16	0.812695929	1.013204917	0.167231443	0.010451965	0.066714924	0.004169683



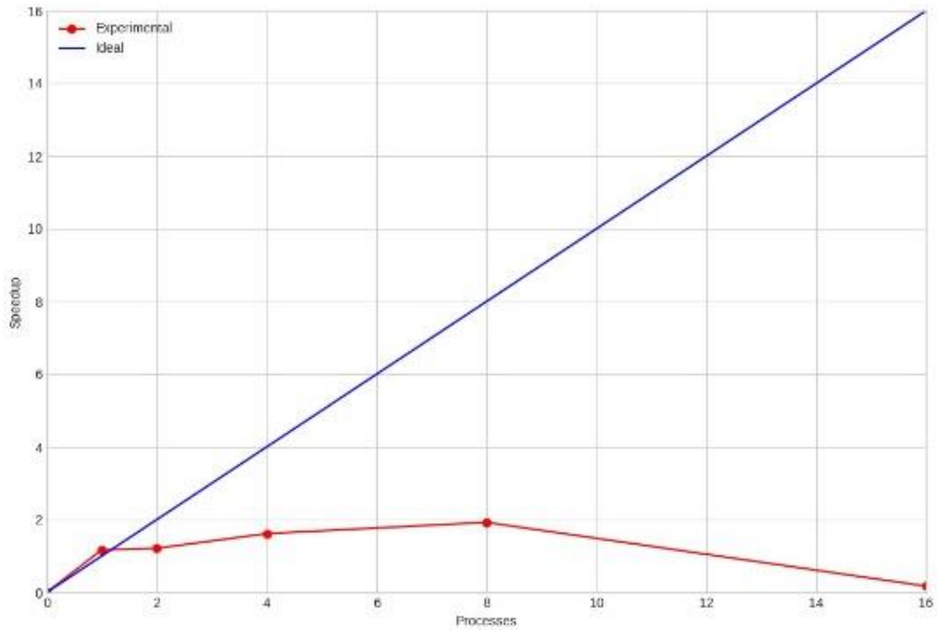
Speedup_timeInit



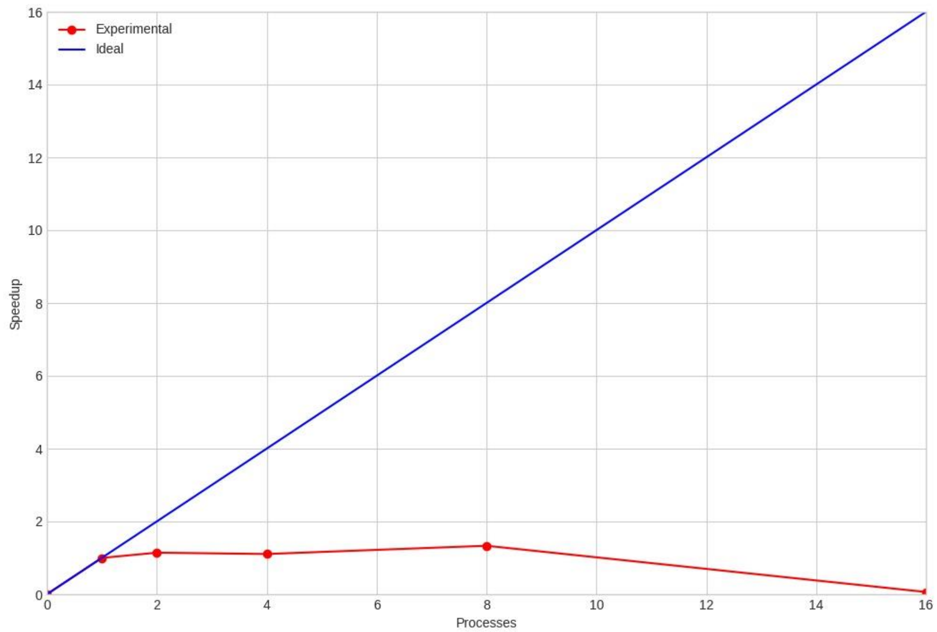
Speedup_timeCount

SIZE-10mln-RANGE-100k-OPT-O3

Version	Processes	TimeInit	TimeCount	Speedup_timeInit	Efficiency_timeInit	Speedup_timeCount	Efficiency_timeCount
Serial	1	0.157426938	0.064288588	1	1	1	1
Parallel	1	0.1344865	0.064310944	1.170577995	1.170577995	0.999652373	0.999652373
Parallel	2	0.111074786	0.056245143	1.210774337	0.605387168	1.143404411	0.571702206
Parallel	4	0.083170643	0.05809325	1.616994836	0.404248709	1.107029551	0.276757388
Parallel	8	0.069842571	0.048236222	1.925566274	0.240695784	1.333250024	0.166656253
Parallel	16	0.784640538	1.013485786	0.171398868	0.010712429	0.063455201	0.00396595



Speedup_timeInit



Speedup_timeCount

Considerations

Case study n.1

Considerations about counting sort

- For SIZE-1mln-RANGE-1k, regardless of gcc optimization, performances are not that good. In fact, data dimension is not very high, so MPI middleware overhead (mostly due to message passing) is not offset by parallel computation advantages.
- For SIZE-1mln-RANGE-100k performances keep being not exciting. O2 optimization is the best.
- For SIZE-10mln-RANGE-1k the results are much better thanks to the increasing of array size. Gcc optimizations give a fundamental contribution in terms of speed-up, guaranteeing mostly in O2 and O3 cases, significant improvements. O3 optimization reaches a peak of about 1.42 when the program is executed on 8 processes.
- SIZE-10mln-RANGE-100k is the configuration with which are obtained the best results without optimization (a speed-up of about 1.33 with 2 processes). The reason can be found obviously in the increasing of the amount of data: greater the size and the range are, greater is the advantage introduced by MPI parallelization. MPI overhead, in this case, is fully recovered and does not represent an important problem anymore. All the optimizations make improvements compared to O0 compilation, guaranteeing excellent performances until 8 processes practically in all the cases. The peak is reached with 8 processes and O3 optimization and is about 1.45.
- Fixing the range and increasing the size, TimeCount increases, as expected.
- Fixing the size and changing the range, TimeCount increases, as expected.

Considerations about initialization

- For SIZE-1mln-RANGE-1k performances are good but not amazing. Different optimizations can't make a great difference.
- For SIZE-1mln-RANGE-100k the results are practically the same, with the only difference that optimization O2 is clearly the best.
- For SIZE-10mln-RANGE-1k performances are much better. The reason is given by the increased dimension of the array that must be initialized. With all the different optimization options, the speed-up is greater than 2 when the program is launched on 4 and 8 processes. In particular, advanced optimizations guarantee better results: the peak is 2.87 and corresponds to the program compiled with O3 optimization and executed on 8 processes.
- For SIZE-10mln-RANGE-100k the results are practically the same, which means that, fixing array size, changing the range does not imply visible performances alterations in terms of speed-up. The peak is 2.89.
- Fixing the range and increasing the size, TimeInit increases, as expected.
- Fixing the size and changing the range, TimeInit does not change visibly, as expected.

Case study n.2

Considerations about counting sort

- For SIZE-1mln-RANGE-1k, performances are not very good. In fact, the not very high data dimension does not allow to offset MPI middleware overhead (mostly due to message passing).
  Regarding optimizations, better performances are obtained without optimization (O0) and they reach a speedup equal to 1.17 when the program is executed on 2 processes.
- For SIZE-1mln-RANGE-100k performances are quite similar to those with range 1k because of the same array size.
- For SIZE-10mln-RANGE-1k performances begin to be better thanks to the increasing of array size. In particular, O0 optimization guarantees a speedup of 1.34 when the program is executed on 2 processes. Besides, O2 and O3 optimizations slightly make performances worse still ensuring a speed-up pick of 1.13 when the number of processes is 4.
- SIZE-10mln-RANGE-100k is the configuration with which are obtained the best speed-up results (1.29 with 4 processes). This can be explained as Case Study n.1: greater the size and range are, greater the advantage introduced by MPI parallelization is. Differently from Case Study n.1, optimizations do not produce better performances than those produced by O0, except for the case obtained with O3 optimization and executed on 8 processes, which reaches a speed-up value of 1.33.

Considerations about initialization

- For SIZE-1mln-RANGE-1k performances are good but they are not the best obtained. They reach a speed-up value of 1.5. Optimizations do not improve results.
- For SIZE-1mln-RANGE-100k performances are similar to those of previous configuration. The maximum speed-up value is 1.52 obtained with O0. As previously, optimizations do not improve results.
- For SIZE-10mln-RANGE-1k performances get better: O0 produces a speed-up pick of 1.75 with 4 processes. Differently from Case Study n.1, optimizations do not produce better performances, except for the case obtained with O1 when the program is executed on 8 processes, which reaches a speed-up value of 1.88.
- For SIZE-10mln-RANGE-100k results are slightly better than those given by the previous configuration. O0 gives a speed-up pick of 1.76 with 4 processes. Optimizations improve performances only when the number of launched processes is 8. In particular, the maximum value, 1.92, is obtained by O3.

Further considerations

- Regarding performances, Case Study n.1 is better than Case Study n.2 for the following reason: it guarantees greater speed-ups and shorter elapsed time for both initialization and counting sort. In particular, the maximum speed-up in terms of counting sort reached by Case Study n.1 is 1.45, while 1.34 is the maximum value guaranteed by Case Study n.2. The difference in terms of initialization is even bigger: *speedup_timeInit* peak for Case Study n.1 is 2.89, while the one reached by Case Study n.2 is 1.92.
- Performances with 16 processes in both Case Studies are very bad because our system has only 4 physical cores and 8 threads.
- The principal reason why Case Study n.1 is faster than Case Study n.2 is because it does not present any overhead due to file management.

API – Case study n.1

Public Docs also available here

Public functions mainS.c

Type	Name
void	**init**(int n, int range, int *full_array) *This is the function that initializes randomly the array 'full_array'.*
void	**countingSort**(int n, int *full_array) *This is the function that sorts the array 'full_array' using Counting Sort Algorithm.*

Public functions countingsort.c

Type	Name
void	**init**(int n, int n_ranks, int rank, int range, int *full_array) *This is the function that initializes randomly the array 'full_array' distributing the computation among the processes.*
void	**countingSort**(int n, int n_ranks, int rank, int *full_array) *This is the function that sorts the array 'full_array' using Counting Sort Algorithm and distributing the computation among the processes.*

Public functions mainS.c documentation

function init

```
void init(
        int n,
        int range,
        int *full_array
)
```

Parameters:

- n number of array elements.
- range maximum acceptable integer.
- full_array pointer to the array.

function countingSort

```
void countingSort(
        int n,
        int *full_array
)
```

Parameters:

- n number of array elements.
- full_array pointer to the unsorted array.

Public functions countingsort.c documentation

function init

```
void init(
        int n,
        int n_ranks,
        int rank,
        int range,
        int *full_array
)
```

Parameters:

- n number of array elements.
- n_ranks number of ranks.
- rank rank of the current process.
- range maximum acceptable integer.
- full_array pointer to the array.

function countingSort

```
void countingSort(
        int n,
        int n_ranks,
        int rank,
        int *full_array
)
```

Parameters:

- n number of array elements.
- n_ranks number of ranks.
- rank rank of the current process.
- full_array pointer to the unsorted array.

API – Case study n.2

Public Docs also available here

Public functions mainS.c

Type	Name
void	**init**(int n, int range, char *file_name) *This is the function that writes in the file 'file_name' 'n' integers.*
void	**countingSort**(int n, char *file_name) *This is the function that sorts the integers in 'file_name' using Counting Sort Algorithm.*
void	**readingFile**(int n, char *file_name) *This is the function that reads the integers in 'file_name' and prints them on stdout.*

Public functions countingsort.c

Type	Name
void	**init**(int n, int n_ranks, int rank, int range, char *file_name) *This is the function that writes in the file 'file_name' 'n' integers distributing the computation among the processes.*
void	**countingSort**(int n, int n_ranks, int rank, char *file_name) *This is the function that sorts the integers in 'file_name' using Counting Sort Algorithm and distributing the computation among the processes.*
void	**readingFile**(int n, int rank, char *file_name) *This is the function that reads the integers in 'file_name' and prints them on stdout.*

Public functions mainS.c documentation

function init

```
void init(
        int n,
        int range,
        char *file_name
)
```

Parameters:

- n — number of array elements.
- range — maximum acceptable integer.
- file_name — file name.

function countingSort

```
void countingSort(
        int n,
        char *file_name
)
```

Parameters:

- n number of array elements.
- file_name file name.

function readingFile

```
void readingFile(
        int n
        char *file_name
)
```

Parameters:

- n number of array elements.
- file_name file name.

Public functions countingsort.c documentation

function init

```
void init(
        int n,
        int n_ranks,
        int rank,
        int range,
        char *file_name
)
```

Parameters:
- n number of array elements.
- n_ranks number of ranks.
- rank rank of the current process.
- range maximum acceptable integer.
- file_name file name.

function countingSort

```
void countingSort(
        int n,
        int n_ranks,
        int rank,
        char *file_name
)
```

Parameters:

- n number of array elements.
- n_ranks number of ranks.
- rank rank of the current process.
- file_name file name.

function readingFile

```
void readingFile(
        int n,
        int rank,
        char *file_name
)
```

Parameters:

- n number of array elements.
- rank rank of the current process.
- file_name file name.

How to run

Since 2 versions of Counting Sort are provided, you must use command `cd version1` or `cd version2` in order to choose the version to run and:

1. Create a build directory and launch cmake
   `mkdir build`
   `cd build`
   `cmake ..`
2. Generate executables with `make`
3. To generate measures, run `make generate_output`
   Attention*: it takes a lot of time. This is the reason why our measures are already included, so you should skip this step.*
4. To extract mean times and speedup curves from them run `make extract_measures`

Results can be found in the `measure/YYYY-MM-DD.hh:mm:ss` directory, divided by problem size and the gcc optimization option used.