

SLURM COMMANDS :

1)

a) `-n` := ntasks = it request an allocation of resources and submits a batch script.

b) `-N` := nodes. It request that a minimumm of minnodes nodes be allocated to this job.

c) `--job-name` = specify a name for the job allocation.

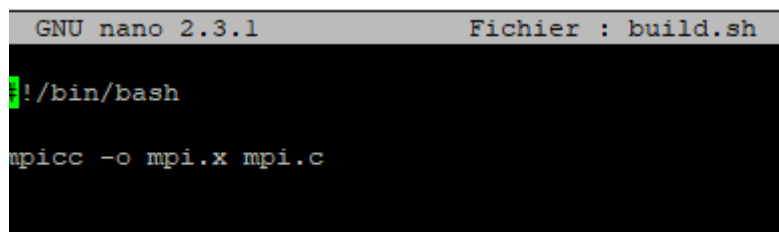
d) `--output` = instruct slurm to connect the batch script's standard output directly to the file name specified in the « filename patter ».

e) `--ntasks-per-node` = request that ntasks be nvoked on each node.

f) `--cpus-per-task` = advise slurm controller that ensuing job steps will require ncpus number of processors per task without this option, the controller will just try to allocate one processor per task.

g) `-p` := partition = request a specific partition for the resources allocation. If not specified, the default behavior is to allow the slurm controller to select the default partition as desginted by the system administrator.

2)



```
GNU nano 2.3.1      Fichier : build.sh
#!/bin/bash
mpicc -o mpi.x mpi.c
```

```

GNU nano 2.3.1 Fichier : mpi-simple

#!/bin/bash
#SBATCH --job-name mpi-simple
#SBATCH --output mpi-simple-%j.out
#SBATCH -n 1

echo "Running on: $SLURM_NODELIST"
echo "SLURM_NTASKS=$SLURM_NTASKS"
echo "SLURM_NTASKS_PER_NODE=$SLURM_NTASKS_PER_NODE"
echo "SLURM_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK"
echo "SLURM_NNODES=$SLURM_NNODES"
echo "SLURM_CPUS_ON_NODE=$SLURM_CPUS_ON_NODE"

SLURM_LIB=$( echo $LD_LIBRARY_PATH | sed -e 's:/\n/g' | grep slurm | grep "lib64$" )
if [ -z "${SLURM_LIB}" ]; then
    echo "SLURM_LIB cannot be set from LD_LIBRARY_PATH, trying from PATH"
    SLURM_LIB=$( which sinfo | sed -e 's$bin/sinfo$lib64$' )
    if [ -z "${SLURM_LIB}" ]; then
        echo "SLURM_LIB cannot be set from PATH, aborting. Check your SLURM environment."
        exit 255
    fi
fi

echo "SLURM_LIB set to ${SLURM_LIB}"
export I_MPI_PMI_LIBRARY=$SLURM_LIB/libpmi.so
export I_MPI_DEBUG=1
export I_MPI_FABRICS=tmi
export TMI_CONFIG=${HOME}/etc/tmi.conf

srun ./mpi.x

```

```

GNU nano 2.3.1 Fichier : mpi-logique.job

#!/bin/bash
#SBATCH --job-name mpi-logique
#SBATCH --output mpi-logique-%j.out
#SBATCH -n 1
#SBATCH -N 4
#SBATCH --ntasks-per-node 6

echo "Running on: $SLURM_NODELIST"
echo "SLURM_NTASKS=$SLURM_NTASKS"
echo "SLURM_NTASKS_PER_NODE=$SLURM_NTASKS_PER_NODE"
echo "SLURM_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK"
echo "SLURM_NNODES=$SLURM_NNODES"
echo "SLURM_CPUS_ON_NODE=$SLURM_CPUS_ON_NODE"

SLURM_LIB=$( echo $LD_LIBRARY_PATH | sed -e 's:/\n/g' | grep slurm | grep "lib64$" )
if [ -z "${SLURM_LIB}" ]; then
    echo "SLURM_LIB cannot be set from LD_LIBRARY_PATH, trying from PATH"
    SLURM_LIB=$( which sinfo | sed -e 's$bin/sinfo$lib64$' )
    if [ -z "${SLURM_LIB}" ]; then
        echo "SLURM_LIB cannot be set from PATH, aborting. Check your SLURM environment."
        exit 255
    fi
fi

echo "SLURM_LIB set to ${SLURM_LIB}"
export I_MPI_PMI_LIBRARY=$SLURM_LIB/libpmi.so
export I_MPI_DEBUG=1
export I_MPI_FABRICS=tmi
export TMI_CONFIG=${HOME}/etc/tmi.conf

srun ./mpi.x

```

3)

```

Running on: localhost
SLURM_NTASKS=1
SLURM_NTASKS_PER_NODE=
SLURM_CPUS_PER_TASK=
SLURM_NNODES=1
SLURM_CPUS_ON_NODE=32
SLURM_LIB cannot be set from LD_LIBRARY_PATH, trying from PATH
SLURM_LIB set to /var/lib/napd/nap/lib64
Format is :
[RANK/NUMPROCS - CPUID/NAME]
[00/01 - [09/localhost.localdomain] - Hello !

```

In parallelisation/slurm/openmp-mpi:

1)

```

GNU nano 2.3.1 Fichier : build.sh

#!/bin/bash

mpicc -openmp -o ompi.x ompi.c

```

```

GNU nano 2.3.1 Fichier : ompi-357.out

sam. avril 10 13:49:10 CEST 2021
SLURM_JOBID=357
Running on: localhost
SLURM_NTASKS=1
SLURM_NTASKS_PER_NODE=1
SLURM_CPUS_PER_TASK=1
SLURM_NNODES=1
SLURM_CPUS_ON_NODE=32
SLURM_LIB cannot be set from LD_LIBRARY_PATH, trying from PATH
SLURM_LIB set to /var/lib/napd/nap/lib64
thread 00/02 on cpu 00 MPI rank 00 numranks 01 name localhost.localdomain
thread 01/02 on cpu 01 MPI rank 00 numranks 01 name localhost.localdomain

```

In parallelisation/slurm/openmp:

1)

```

GNU nano 2.3.1 Fichier : build.sh

#!/bin/bash

gcc -fopenmp -openmp -o openmp.x openmp.c

```

```
GNU nano 2.3.1 Fichier : openmp.job

#!/bin/bash
#SBATCH --job-name openmp
#SBATCH --output openmp-%j.out
#SBATCH -n 1
##SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=2

# OpenMP runtime settings
export OMP_NUM_THREADS=$SLURM_CPUS_ON_NODE
mpirun ./openmp.x
```

2)

```
GNU nano 2.3.1 Fichier : openmp-360.out

Current thread/processor = 1/0
Thread 0 getting environment info...
Number of processors = 1
Number of threads = 32
Max threads = 32
In parallel? = 1
Dynamic threads enabled? = 0
Nested parallelism supported? = 0
Current thread/processor = 0/0
Current thread/processor = 2/0
Current thread/processor = 3/0
Current thread/processor = 4/0
Current thread/processor = 5/0
Current thread/processor = 6/0
Current thread/processor = 7/0
Current thread/processor = 8/0
Current thread/processor = 9/0
Current thread/processor = 10/0
Current thread/processor = 11/0
Current thread/processor = 12/0
Current thread/processor = 13/0
Current thread/processor = 14/0
Current thread/processor = 15/0
Current thread/processor = 16/0
Current thread/processor = 17/0
Current thread/processor = 18/0
Current thread/processor = 19/0
Current thread/processor = 20/0
Current thread/processor = 21/0
Current thread/processor = 22/0
Current thread/processor = 23/0
Current thread/processor = 24/0
Current thread/processor = 25/0
Current thread/processor = 26/0
Current thread/processor = 27/0
Current thread/processor = 28/0
Current thread/processor = 29/0
Current thread/processor = 30/0
Current thread/processor = 31/0
```

OpenMP

In parallelisation/openmp:

1)

```
GNU nano 2.3.1 Fichier : job openmp.sh

#!/bin/bash
#SBATCH --job-name job
#SBATCH --output job.out
#SBATCH --nodes 1
#SBATCH --ntasks 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4

# Alternative
##SBATCH --ntasks 4
##SBATCH --cpus-per-task=1

# OpenMP runtime settings
export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Alternative
#export OMP_NUM_THREADS=$SLURM_NTASKS

echo "Running on: $SLURM_NODELIST"
echo "SLURM_NTASKS=$SLURM_NTASKS"
echo "SLURM_NTASKS_PER_NODE=$SLURM_NTASKS_PER_NODE"
echo "SLURM_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK"
echo "SLURM_CPUS_ON_NODE=$SLURM_CPUS_ON_NODE"

mpirun ./a.out
```

2)

La variable d'environnement `OMP_DYNAMIC` contrôle l'ajustement dynamique du nombre de threads à utiliser pour exécuter des régions parallèles en définissant la valeur initiale de l'ICV `dyn-var`. Si la variable d'environnement est définie sur `false`, l'ajustement dynamique du nombre de threads est désactivé.

La variable d'environnement `OMP_NUM_THREADS` définit le nombre de threads à utiliser pour les régions parallèles en définissant la valeur initiale de l'ICV `nthreads-var`.

Echo est une commande UNIX qui permet d'afficher une chaîne de caractères passée en paramètre sur le terminal. Cette commande est fréquemment utilisée dans les scripts shell et les programmes batchs pour indiquer textuellement un état du programme à l'écran ou dans un fichier.

3)

- For static : When no chunk-size is specified, OpenMP divides iterations into chunks that are approximately equal in size and it distributes at most one chunk to each thread.

- For dynamic : chunksize default is one

- For guided : chunksize default is one

4)

With static and chunk = 5 :

```
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o saxpy-parallel.x saxpy-parallel.c
[thibault.lejoux@localhost openmp]$ ls
hello.c  integrate.c  job_openmp.sh  openmp_getenvinfo.c  race.c  saxpy.c  saxpy-parallel.c  saxpy-parallel.x  sharing.c  simd.c  test.mod
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 0/32 ; Variable i: 0
Rang Thread: 0/32 ; Variable i: 1
Rang Thread: 0/32 ; Variable i: 2
Rang Thread: 0/32 ; Variable i: 3
Rang Thread: 0/32 ; Variable i: 4
Rang Thread: 1/32 ; Variable i: 5
Rang Thread: 1/32 ; Variable i: 6
Rang Thread: 1/32 ; Variable i: 7
Rang Thread: 1/32 ; Variable i: 8
Rang Thread: 1/32 ; Variable i: 9
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.010s
user    0m0.156s
sys     0m0.015s
[thibault.lejoux@localhost openmp]$ nano saxpy-parallel.c
```

With chunk equal 5, there are only two threads. The first threads execute the operation 0,1,2,3,4 and the second it execute the threads 5,6,7,8,9.

With static and chunk by default :

```
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 0/32 ; Variable i: 0
Rang Thread: 7/32 ; Variable i: 7
Rang Thread: 9/32 ; Variable i: 9
Rang Thread: 1/32 ; Variable i: 1
Rang Thread: 3/32 ; Variable i: 3
Rang Thread: 6/32 ; Variable i: 6
Rang Thread: 8/32 ; Variable i: 8
Rang Thread: 5/32 ; Variable i: 5
Rang Thread: 2/32 ; Variable i: 2
Rang Thread: 4/32 ; Variable i: 4
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.008s
user    0m0.133s
sys     0m0.004s
```

By default, there are 10 threads which are executing one operation.

Static with chunksize equal 7 :

```
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 0/32 ; Variable i: 0
Rang Thread: 0/32 ; Variable i: 1
Rang Thread: 0/32 ; Variable i: 2
Rang Thread: 0/32 ; Variable i: 3
Rang Thread: 0/32 ; Variable i: 4
Rang Thread: 0/32 ; Variable i: 5
Rang Thread: 0/32 ; Variable i: 6
Rang Thread: 1/32 ; Variable i: 7
Rang Thread: 1/32 ; Variable i: 8
Rang Thread: 1/32 ; Variable i: 9
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.008s
user    0m0.127s
sys     0m0.003s
```

First thread do seven firsts operation and the second threads do the 3 last operation

In static, with a chunk equal 2 :

```
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 0/32 ; Variable i: 0
Rang Thread: 0/32 ; Variable i: 1
Rang Thread: 1/32 ; Variable i: 2
Rang Thread: 1/32 ; Variable i: 3
Rang Thread: 3/32 ; Variable i: 6
Rang Thread: 3/32 ; Variable i: 7
Rang Thread: 2/32 ; Variable i: 4
Rang Thread: 2/32 ; Variable i: 5
Rang Thread: 4/32 ; Variable i: 8
Rang Thread: 4/32 ; Variable i: 9
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.007s
user    0m0.127s
sys     0m0.002s
```

For the static we could say that the execution time decrease a little when there are multiple threads.

Dynamic with default chunk size :

```
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 21/32 ; Variable i: 5
Rang Thread: 14/32 ; Variable i: 0
Rang Thread: 9/32 ; Variable i: 7
Rang Thread: 0/32 ; Variable i: 9
Rang Thread: 11/32 ; Variable i: 8
Rang Thread: 31/32 ; Variable i: 1
Rang Thread: 18/32 ; Variable i: 6
Rang Thread: 19/32 ; Variable i: 3
Rang Thread: 29/32 ; Variable i: 4
Rang Thread: 25/32 ; Variable i: 2
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.008s
user    0m0.125s
sys     0m0.003s
```

As we see, the default chunk size is one, the repartition on the thread executing operation as no pattern, it is dynamic.

With duck size equal 3 :

```
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 0/32 ; Variable i: 0
Rang Thread: 0/32 ; Variable i: 1
Rang Thread: 0/32 ; Variable i: 2
Rang Thread: 27/32 ; Variable i: 3
Rang Thread: 27/32 ; Variable i: 4
Rang Thread: 27/32 ; Variable i: 5
Rang Thread: 31/32 ; Variable i: 6
Rang Thread: 31/32 ; Variable i: 7
Rang Thread: 31/32 ; Variable i: 8
Rang Thread: 24/32 ; Variable i: 9
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.008s
user    0m0.132s
sys     0m0.003s
```


It choose dynamicaly threads to execute 3 operations. There are less threads than before and it seems to be a little less fast.

With chunk equal 5 :

```
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 27/32 ; Variable i: 5
Rang Thread: 27/32 ; Variable i: 6
Rang Thread: 27/32 ; Variable i: 7
Rang Thread: 27/32 ; Variable i: 8
Rang Thread: 27/32 ; Variable i: 9
Rang Thread: 26/32 ; Variable i: 0
Rang Thread: 26/32 ; Variable i: 1
Rang Thread: 26/32 ; Variable i: 2
Rang Thread: 26/32 ; Variable i: 3
Rang Thread: 26/32 ; Variable i: 4
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.010s
user    0m0.157s
sys     0m0.010s
```

It choose 2 threads and execute 5 operations for each threads. The exécution time is longer, probably to the fact that we have less threads.

For the guided schedule :

With default chunk size :

```
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 7/32 ; Variable i: 4
Rang Thread: 13/32 ; Variable i: 9
Rang Thread: 22/32 ; Variable i: 8
Rang Thread: 10/32 ; Variable i: 7
Rang Thread: 0/32 ; Variable i: 1
Rang Thread: 29/32 ; Variable i: 0
Rang Thread: 21/32 ; Variable i: 3
Rang Thread: 15/32 ; Variable i: 2
Rang Thread: 31/32 ; Variable i: 5
Rang Thread: 12/32 ; Variable i: 6
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.009s
user    0m0.125s
sys     0m0.042s
```

It seems to be the same as dynamic. So, to see the difference i modified the number of threads by using the function `omp_set_num_threads(int num_threads)`.

Here's the result with default chunk-size and 4 threads :

```
[thibault.lejoux@localhost openmp]$ time ./saxpy-parallel.x
Rang Thread: 1/4 ; Variable i: 5
Rang Thread: 1/4 ; Variable i: 6
Rang Thread: 0/4 ; Variable i: 0
Rang Thread: 0/4 ; Variable i: 1
Rang Thread: 0/4 ; Variable i: 2
Rang Thread: 0/4 ; Variable i: 9
Rang Thread: 3/4 ; Variable i: 7
Rang Thread: 1/4 ; Variable i: 8
Rang Thread: 2/4 ; Variable i: 3
Rang Thread: 2/4 ; Variable i: 4
results 1 [expected: 1]
results 3 [expected: 3]
results 5 [expected: 5]
results 7 [expected: 7]
results 9 [expected: 9]
results 11 [expected: 11]
results 13 [expected: 13]
results 15 [expected: 15]
results 17 [expected: 17]
results 19 [expected: 19]

real    0m0.003s
user    0m0.000s
sys     0m0.004s
```

The threads are choosing randomly and the number of operation are decreasing one by one (4operations for threads 0, 3 for the second, 2 for the third thread and one for the last). The guided operation is quicker than the static and the dynamic in that case.

5)

Real is Wall clock time - time from start to end of call. This is all the elapsed time including the time slices used by other processes and the time that the process spends blocked

User is the amount of CPU time spent in code mode (outside the kernel) inside the process. This is only the actual CPU time used in the execution of the process. Other processes and the time the process spends Blocked do not count towards this number.

Sys is the CPU time spent in the kernel during the process.

6)

a) for f1 : -forward data dependency and input dependency

for f2 : input dependency

for f3 : there are no dependencies

b) To be vectorizable, there should be no backward dependency. So for the dependecny here, there are all vectorizable.

c) If data are dependent it is impossible to parallelized. So we can only do parallelization on f3 and f2 because input dependency are allowed.

7)

```
GNU nano 2.3.1 Fichier : race.c
#include <stdio.h>
#include "omp.h"

void f1(int ndim, float* a, float* b) {
    int i;
    /* Boucle à analyser */
    #pragma omp parallel for schedule(static, 5)
    for (i=0; i<ndim-1; i++){
        a[i] = (a[i] + a[i+1]) / 2;
    }

    /* Affichage: ne pas modifier / paralleliser */
    for (i=0; i<ndim; i++){
        printf("a[%d] = %.1f\n", i, a[i]);
    }
}

void f2(int ndim, float* a, float* b) {
    int i;
    /* Boucle à analyser */
    #pragma omp parallel for schedule(static, 5)
    for (i=3; i<ndim; i++){
        b[i] = (a[i] + a[i-3]) / 2;
    }

    /* Affichage: ne pas modifier / paralleliser */
    for (i=0; i<ndim; i++){
        printf("b[%d] = %.1f\n", i, b[i]);
    }
}

void f3(int ndim, float* a, float* b) {
    int i, j;
    float c;
    // boucles à analyser
    #pragma omp parallel for schedule(static, 5)
    for (i=0; i<ndim-2; i++){
        c = 0.0;
        for (j=0; j<2; j++) {
            c += a[i+j];
        }
        b[i] = a[i] * c;
    }

    /* Affichage: ne pas modifier / paralleliser */
    for (i=0; i<ndim; i++){
        printf("b[%d] = %.1f\n", i, b[i]);
    }
}

int main (int argc, char * argv[])
{
    int i, ndim=10;
    float a[ndim], b[ndim];

    /* initialisation */
    for (i=0; i<ndim; i++) {
```

For f1 before and after parallelisation :

```
[thibault.lejoux@localhost openmp]$ time ./race.x
a[0] = 4.5
a[1] = 3.5
a[2] = 2.5
a[3] = 1.5
a[4] = 0.5
a[5] = -0.5
a[6] = -1.5
a[7] = -2.5
a[8] = -3.5
a[9] = -4.0

real    0m0.002s
user    0m0.002s
sys     0m0.001s
[thibault.lejoux@localhost openmp]$ nano race.c
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o race.x race.c
[thibault.lejoux@localhost openmp]$ time ./race.x
a[0] = 4.5
a[1] = 3.5
a[2] = 2.5
a[3] = 1.5
a[4] = 0.5
a[5] = -0.5
a[6] = -1.5
a[7] = -2.5
a[8] = -3.5
a[9] = -4.0

real    0m0.009s
user    0m0.128s
sys     0m0.004s
```

Before and after for f2 :

```
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o race.x race.c
[thibault.lejoux@localhost openmp]$ time ./race.x
b[0] = 0.0
b[1] = 0.0
b[2] = 0.0
b[3] = 3.5
b[4] = 2.5
b[5] = 1.5
b[6] = 0.5
b[7] = -0.5
b[8] = -1.5
b[9] = -2.5

real    0m0.002s
user    0m0.000s
sys     0m0.002s
[thibault.lejoux@localhost openmp]$ nano race.c
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o race.x race.c
[thibault.lejoux@localhost openmp]$ time ./race.x
b[0] = 0.0
b[1] = 0.0
b[2] = 0.0
b[3] = 3.5
b[4] = 2.5
b[5] = 1.5
b[6] = 0.5
b[7] = -0.5
b[8] = -1.5
b[9] = -2.5

real    0m0.013s
user    0m0.151s
sys     0m0.039s
```

Before and after for f3 :

```
[thibault.lejoux@localhost openmp]$ time ./race.x
b[0] = 45.0
b[1] = 28.0
b[2] = 15.0
b[3] = 6.0
b[4] = 1.0
b[5] = -0.0
b[6] = 3.0
b[7] = 10.0
b[8] = 0.0
b[9] = 0.0

real    0m0.002s
user    0m0.000s
sys     0m0.002s
[thibault.lejoux@localhost openmp]$ nano race.c
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o race.x race.c
[thibault.lejoux@localhost openmp]$ time ./race.x
b[0] = 45.0
b[1] = 28.0
b[2] = 15.0
b[3] = 6.0
b[4] = 1.0
b[5] = -0.0
b[6] = 3.0
b[7] = 10.0
b[8] = 0.0
b[9] = 0.0

real    0m0.009s
user    0m0.127s
sys     0m0.003s
```

8)

```
GNU nano 2.3.1 Fichier : job_openmp.sh

#!/bin/bash
#SBATCH --job-name job
#SBATCH --output job.out
#SBATCH --nodes 1
#SBATCH --ntasks 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4

# Alternative
##SBATCH --ntasks 4
##SBATCH --cpus-per-task=1

# OpenMP runtime settings
export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Alternative
#export OMP_NUM_THREADS=4

echo "Running on: $SLURM_NODENAME"
echo "SLURM_NTASKS=$SLURM_NTASKS"
echo "SLURM_NTASKS_PER_NODE=$SLURM_NTASKS_PER_NODE"
echo "SLURM_CPUS_PER_TASK=$SLURM_CPUS_PER_TASK"
echo "SLURM_CPUS_ON_NODE=$SLURM_CPUS_ON_NODE"

gcc -fopenmp -openmp -o race.x race.c
```

```
GNU nano 2.3.1 Fichier : openmp.job

#!/bin/bash
#SBATCH --job-name openmp
#SBATCH --output openmp-%j.out
#SBATCH -n 1
##SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=2

# OpenMP runtime settings
export OMP_NUM_THREADS=$SLURM_CPUS_ON_NODE
mpirun ./race.x
```

9)

```
GNU nano 2.3.1 Fichier : openmp-363.out

a[0] = 4.5
a[1] = 3.5
a[2] = 2.5
a[3] = 1.5
a[4] = 0.5
a[5] = -0.5
a[6] = -1.5
a[7] = -2.5
a[8] = -3.5
a[9] = -4.0
```

10)

For f2 with 4 threads :

```
GNU nano 2.3.1 Fichier : openmp-372.out
OMP_NUM_THREADS=4
b[0] = 0.0
b[1] = 0.0
b[2] = 0.0
b[3] = 3.5
b[4] = 2.5
b[5] = 1.5
b[6] = 0.5
b[7] = -0.5
b[8] = -1.5
b[9] = -2.5
```

For f2 with 1 thread :

```
GNU nano 2.3.1 Fichier : openmp-373.out
OMP_NUM_THREADS=1
b[0] = 0.0
b[1] = 0.0
b[2] = 0.0
b[3] = 3.5
b[4] = 2.5
b[5] = 1.5
b[6] = 0.5
b[7] = -0.5
b[8] = -1.5
b[9] = -2.5
```

The result are always the same as before, we get the same thins for f1 and f3

11) Yes, we got always the same result for all the function

12)

```
/* fonction à intégrer */
double func(double x) {
    return 1/(1+(x*x));
}
```

13)

```
double integrate(int n, double x[]) {
    double a,b,dx;
    int i;
    float h, s,k,result;
    a=0.0;
    b=1.0;
    h=(b-a)/n;
    s=func(b)+func(a);
    for(i=1;i<n;i++){
        s = s +2*(func(a+i*h));
    }
    return (h/2)*s;
}
```

14)

```

/* fonction pour l'integration numerique de func() sur la grille de points x[]
n est la taille du tableau x en entree. Elle doit retourner l'integrale de func(x) */
double integrate(int n, double x[]) {
    double a,b,dx;
    int i;
    double h,k,result;
    a=0.0;
    b=1.0;
    result=0;
    dx=(b-a)/n;
    for(i=0;i<n;i++){
        result = result +(func(a+x[i])+func(a+x[i+1]))*(dx/2);
    };
    result = fabs( result*4 - M_PI);
    return result;
}

```

Result for N=10, 100, 1000

```

[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o integrate.x integrate.c
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 0.00166666

```

```

[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o integrate.x integrate.c
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-05

```

```

[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o integrate.x integrate.c
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-07

```

15) there are iteration, so we have to do a private and a reduction parallelization.

```

/* fonction pour l'integration numerique de func() sur la grille de points x[]
n est la taille du tableau x en entree. Elle doit retourner l'integrale de func(x) */
double integrate(int n, double x[]) {
    double a,b,dx;
    int i;
    double h,k,result,tmp;
    tmp=0;
    a=0.0;
    b=1.0;
    result=0;
    dx=(b-a)/n;
    #pragma omp parallel for reduction(+:result) private(tmp)
    for(i=0;i<n;i++){
        tmp=(func(a+x[i])+func(a+x[i+1]))*(dx/2);
        result+=tmp;
    };

    result = fabs( result*4 - M_PI);
    return result;
}

```

```

[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-07

real    0m0.663s
user    0m20.838s
sys     0m0.094s

```

16) result with 4 threads :

```
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o integrate.x integrate.c
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-07

real    0m0.285s
user    0m1.113s
sys     0m0.019s
```

17)

```
/* fonction pour l'integration numerique de func() sur la grille de points x[]
n est la taille du tableau x en entree. Elle doit retourner l'integrale de func(x) */
double integrate(int n, double x[]) {
    double a,b,dx;
    int i;
    double h,k,result,tmp;
    tmp=0;
    a=0.0;
    b=1.0;
    result=0;
    dx=(b-a)/n;
    omp_set_num_threads(1);
    #pragma omp parallel for reduction(+:result) private(tmp)
    for(i=0;i<n;i++){
        tmp=(func(a+x[i])+func(a+x[i+1]))*(dx/2);
        result+=tmp;
    };

    result = fabs( result*4 - M_PI);
    return result;
}
```

With p = 1:

```
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-07

real    0m0.379s
user    0m0.375s
sys     0m0.004s
```

P=2 :

```
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o integrate.x integrate.c
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-07

real    0m0.274s
user    0m0.511s
sys     0m0.036s
```

P=4 :

```
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o integrate.x integrate.c
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-07

real    0m0.226s
user    0m0.876s
sys     0m0.022s
```

P==8 :


```
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o integrate.x integrate.c
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-07

real    0m0.240s
user    0m1.886s
sys     0m0.018s
```

P=16 :

```
[thibault.lejoux@localhost openmp]$ gcc -fopenmp -lm -o integrate.x integrate.c
[thibault.lejoux@localhost openmp]$ time ./integrate.x
Result : 1.66667e-07

real    0m0.350s
user    0m5.527s
sys     0m0.044s
```

We can see that with more threads, at the beginning the real time decrease and then increase when it is more than 4.

18)

	Static chunk-size=5	Dynamic chunk-size=5	Guided chunk-size=5
P=1	Result : 1.66667e-07 real 0m0.385s user 0m0.376s sys 0m0.009s	Result : 1.66667e-07 real 0m0.486s user 0m0.479s sys 0m0.007s	Result : 1.66667e-07 real 0m0.390s user 0m0.385s sys 0m0.004s
P=2	Result : 1.66667e-07 real 0m0.264s user 0m0.495s sys 0m0.030s	Result : 1.66667e-07 real 0m0.603s user 0m1.173s sys 0m0.031s	Result : 1.66667e-07 real 0m0.440s user 0m0.854s sys 0m0.025s
P=4	Result : 1.66667e-07 real 0m0.240s user 0m0.930s sys 0m0.024s	Result : 1.66667e-07 real 0m0.653s user 0m2.566s sys 0m0.039s	Result : 1.66667e-07 real 0m0.246s user 0m0.965s sys 0m0.013s
P=8	Result : 1.66667e-07 real 0m0.258s user 0m2.017s sys 0m0.035s	Result : 1.66667e-07 real 0m0.607s user 0m4.782s sys 0m0.056s	Result : 1.66667e-07 real 0m0.418s user 0m3.296s sys 0m0.029s
P=16	Result : 1.66667e-07 real 0m0.383s user 0m6.045s sys 0m0.051s	Result : 1.66667e-07 real 0m0.742s user 0m11.808s sys 0m0.034s	Result : 1.66667e-07 real 0m0.701s user 0m11.128s sys 0m0.054s

For the static schedule, it does not change a lot the runtime. It just add a few time more.

For dynamic and guide dit add more time.

MPI

In parallelisation/mpi :

- 1) MPI allow us to create group of process which are identified with their rank. Rank start to 0 to N-1 processes where N is the size of the group.
- 2) For MPI_Send : in the input their are the
 - **buf (initial address of send buffer)**
 - count (number of éléments in send buffer)
 - datatype (datatype of each send buffer element)
 - dest (rank of destination)
 - tag (message tag)
 - comm (communicator)

For MPI_Recv :

- count (maximum number of éléments in receive buffer)
- datatype (datatype of each receive buffer element)
- source (rank of the source of the message)**
- tag (message tag, the tag can be used to add some information about what the data actually represents.**
- comm (communicator)

3) The blocking communication do not return until the communication is finished. In contrast, non blocking communication return immediately even if the communication is not finished.

4) MPI offers the choice of several communication modes that allow one to control the choice of the communication protocol. The send call described uses the standard communication mode. In this mode, it is up to MPI to decide whether outgoing messages will be buffered. MPI may buffer outgoing messages. In such a case, the send call may complete before a matching receive is involved. On the other hand, buffer space may be unavailable, or MPI may choose not to buffer outgoing messages, for performance reasons. In this case, the send call will not complete until a matching receive has been posted, and the data has been moved to the receiver.

Source : <https://www.codingame.com/playgrounds/47058/have-fun-with-mpi-in-c/communication-modes>

5) Deadlock happen when the message passing cannot be completed. Deadlock occurs when one process is blocked and waiting for a second process to complete its work and release locks, while the second process at the same time is blocked and waiting for the first process to release the lock.

A blocking send can complete as soon as the message was buffered, even if no matching receive has been executed by the receiver. On the other hand, message buffering can be expensive, as it entails additional memory-to-memory copying, and it requires the allocation of memory for buffering.

6)

Blocking communication :

Block (or blocking lock) occurs when two processes need access to same piece of data concurrently so one process locks the data and the other one needs to wait for the other one to complete and release the lock. As soon as the first process is complete, the blocked process resumes operation. The blocking chain is like a queue: once the blocking process is complete, the next processes can continue.

Deadlock :

Deadlock occurs when one process is blocked and waiting for a second process to complete its work and release locks, while the second process at the same time is blocked and waiting for the first process to release the lock.

Source :

<https://blog.pythian.com/locks-blocks-deadlocks/>

So the difference is : a deadlock is when the processes are already blocking each other and there needs an external intervention to resolve it. The blocking communication is more like a queue, a process wait an other process to finish.

7) During a broadcast, one process sends the same data to all processes in a communicator and the scatter sends chunks of an array to different processes.

MPI_Gather is the inverse of MPI_Scatter. Instead of spreading elements from one process to many processes, MPI_Gather takes elements from many processes and gathers them to one single process. And the difference between gather and allgather is that MPI_Allgather is an MPI_Gather followed by an MPI_Bcast.

Reduce is a data reduction that involves reducing a set of numbers into a smaller set of numbers via a function. MPI_Allreduce is identical to MPI_Reduce with the exception that it does not need a root process id. MPI_Allreduce is the equivalent of doing MPI_Reduce followed by an MPI_Bcast

8)

```
[thibault.lejoux@localhost mpi]$ ls
allgather_inplace.c  gather_one.c    recv_deadlock.c  send_buffering.c
allgather_multi.c   hello.c         reduce.c         send_deadlock.c
bcast.c             hello_node.c    scatter.c        send_nonblocking.c
gather_multi.c       job_mpi.sh      send_anytag.c    send_simple.c
[thibault.lejoux@localhost mpi]$ nano reduce.c
[thibault.lejoux@localhost mpi]$ nano send_deadlock.c
```

Each file do the method corresponding to the name of the file. For exemple the reduce do a reduction of data to a smaller data with the function MPI_reduce which reduce values on all processes to a single value :

```
MPI_Reduce(a,b,2,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
if (rank == 0) {
    printf("I am %i b[0:1] = %i, %i\n",rank,b[0],b[1]);
}
```

Same for `allgather_inplace.c`. It use the function `MPI_algather` to gather data from all tasks and distribute the combined data to all tasks.

```
for (i=0; i<2*size; i++) printf("I am %i: before a[%i] = %i\n",rank,i,a[i]);
MPI_Allgather(MPI_IN_PLACE,0,MPI_INT,a,2,MPI_INT,MPI_COMM_WORLD);
if (rank == 0) {
    for (i=0; i<2*size; i++) printf("I am %i: after a[%i] = %i\n",rank, i,a[i]);
}
MPI_Finalize();
```

For `gather_one` the function use `MPI_gatfer` to gather together values from a group of processes :

```
MPI_Gather(&a,1,MPI_INT,b,1,MPI_INT,0,MPI_COMM_WORLD);
if (rank == 0) {
    for (i=0; i<size; i++) printf("b[%i] = %i\n",i,b[i]);
}
MPI_Finalize();
```

For `recv_deadlock`, it send a Synchronous blocking `Ssend`.

For `sned_buffering` it uses `MPI_send` to perfoms a blocking send.

Each of this file is using a MPI function to perform a method corresponding to their name, we can fin the explication of these method on the internet like on

<https://www.mpich.org/>