Introduction to High Performance Computing

\\

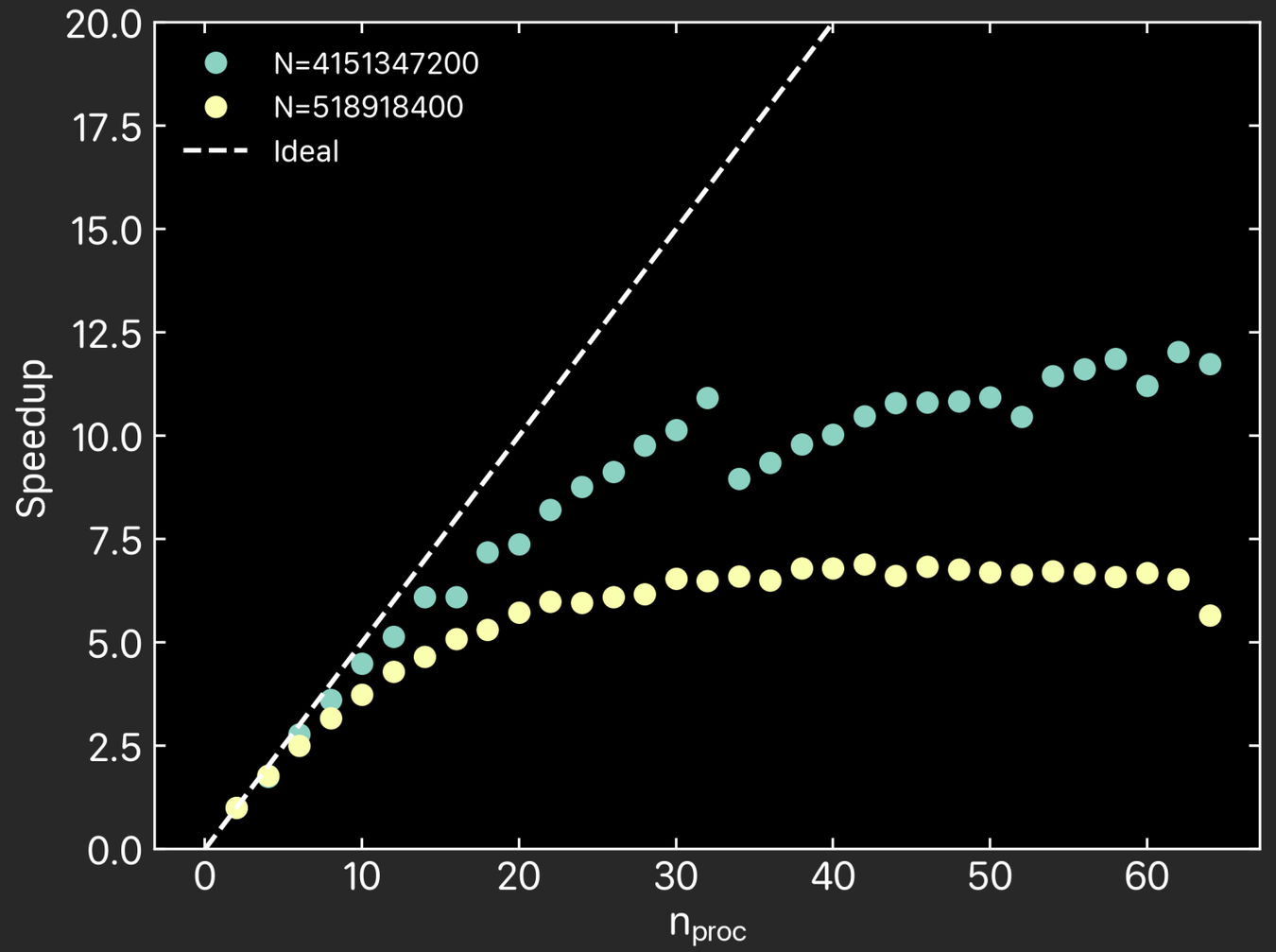
SDS406 – Fall semester, 2024 - 2025

\\

L03: Introduction to OpenMP, 14^{th} October 2024

Parallel calculation of π

n_{proc}	t [sec]	t [sec]
2	4.35	33.93
4	2.45	19.32
6	1.73	12.17
8	1.37	9.39
10	1.16	7.55
12	1.01	6.60
14	0.93	5.56
16	0.85	5.56
18	0.82	4.73
20	0.76	4.60
22	0.73	4.13
24	0.73	3.87
26	0.71	3.72
28	0.70	3.47
30	0.66	3.35
32	0.67	3.11
34	0.66	3.78
36	0.67	3.63
38	0.64	3.46
40	0.64	3.38
42	0.63	3.24
44	0.66	3.14
46	0.64	3.14
48	0.64	3.13
50	0.65	3.10
52	0.65	3.24
54	0.65	2.96
56	0.65	2.92
58	0.66	2.86
60	0.65	3.03
62	0.67	2.82
64	0.77	2.89



Outline

Shared-memory parallel programming

- Introduction to OpenMP
  - Parallelizing programs across the cores of the same node

Performance and optimization

- Assessing performance of programs
  - Floating point performance and memory I/O
  - Assessing bottlenecks

Parallel programming using OpenMP

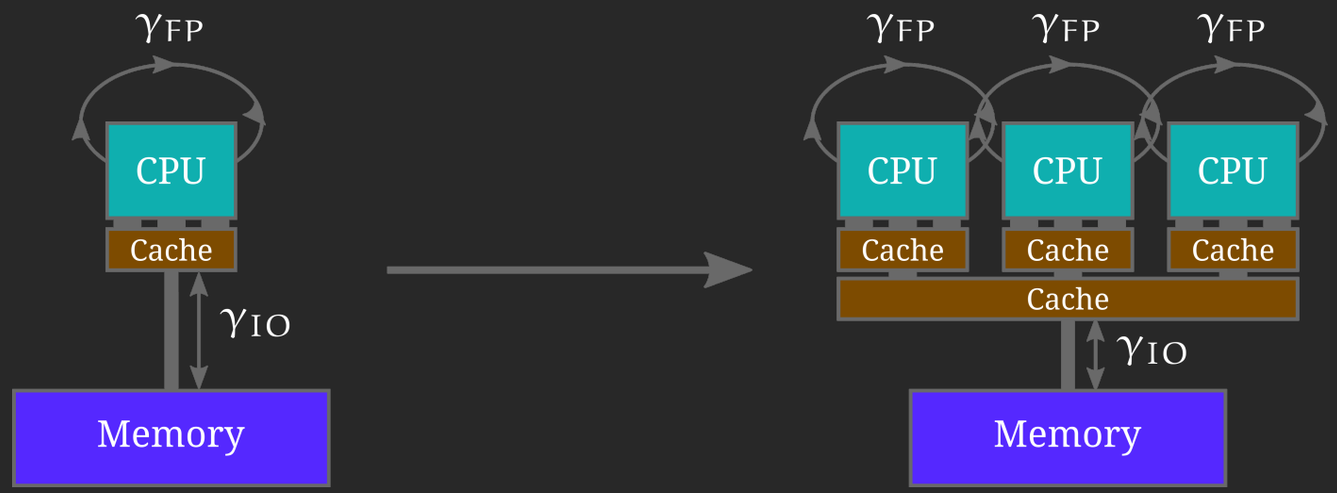
Introduction with examples

Parallel programming

For the remainder of the semester we will introduce three models

- Shared memory programming via the use of OpenMP (starting today)
- Distributed memory programming via the use of MPI
- GPU programming with CUDA

OpenMP introduction



OpenMP

- *Shared memory* parallelization
- Use multiple threads that share a common memory address space
- Pragma-based, i.e. uses directives rather than functions (mostly)
- Also an API, i.e. some simple functionality through function calls

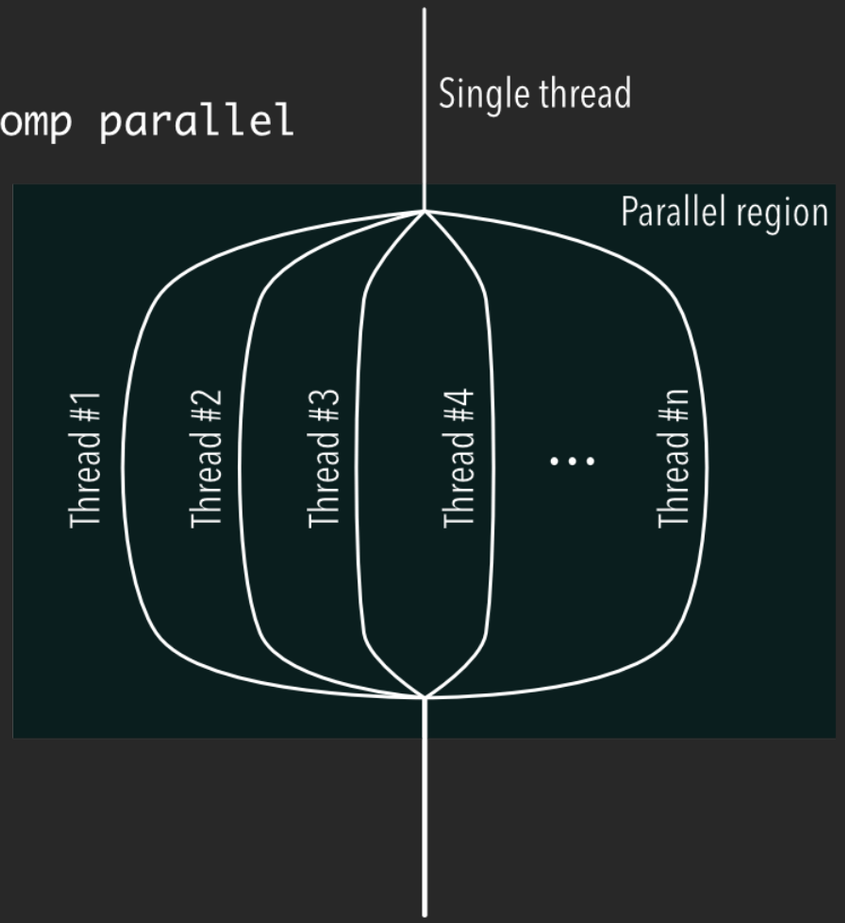
OpenMP introduction

- Starts with a single thread
- Define *parallel regions*
- More than one parallel regions can be defined
- So-called *fork-join* concept

```
main();

#pragma omp parallel
{

}
```



OpenMP introduction

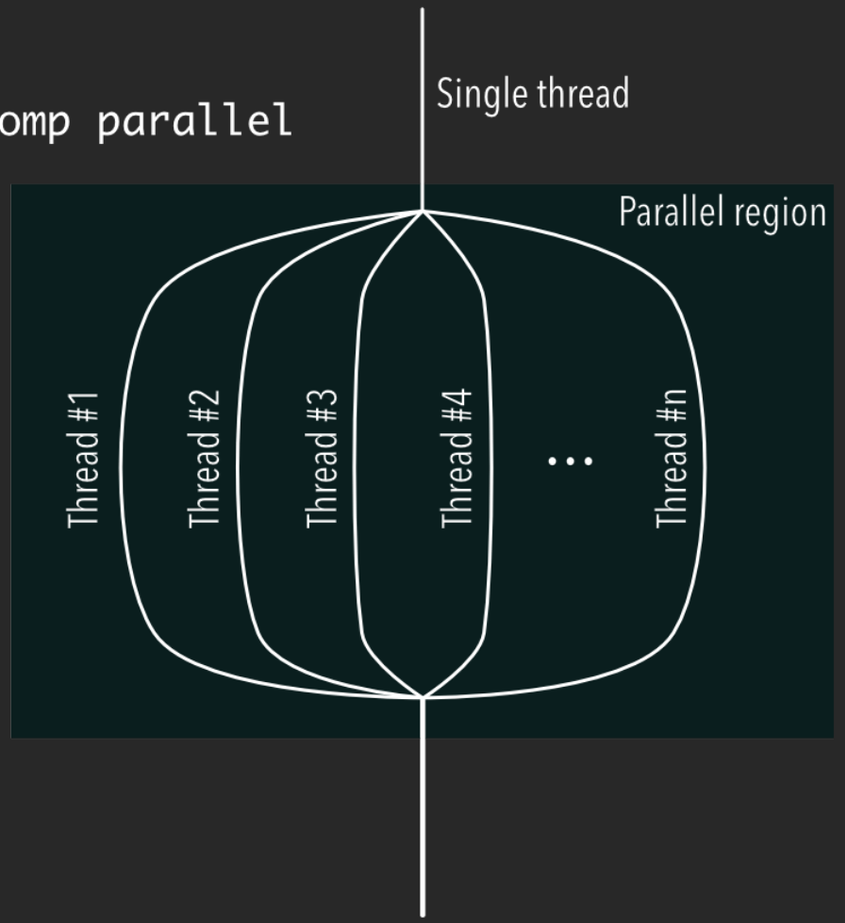
- Starts with a single thread
- Define *parallel regions*
- More than one parallel regions can be defined
- So-called *fork-join* concept

```
int
main()
{
  /* ...
     work to do outside parallel region
     ... */
#pragma omp parallel
  {
    /* ...
       work to do in parallel
       ... */
  }
  /* ...
     more work outside parallel region
     ... */
  return 0;
}
```

```
main();

#pragma omp parallel
{

}
```



OpenMP introduction

Parallel regions:

- No jumping in or out (e.g. `goto`)
- No branching in or out (e.g. inside `if-else` block)
- A thread can terminate the program from within a block

OpenMP

- OpenMP *runtime* takes care of thread management, forking, joining, etc.
- Specify number of threads via environment variable `OMP_NUM_THREADS`

```
int
main()
{
  /* ...
     work to do outside parallel region
     ... */
#pragma omp parallel
  {
    /* ...
       work to do in parallel
       ... */
  }
  /* ...
     more work outside parallel region
     ... */
  return 0;
}
```

OpenMP introduction

Parallel regions:

- No jumping in or out (e.g. `goto`)
- No branching in or out (e.g. inside `if-else` block)
- A thread can terminate the program from within a block

OpenMP

- OpenMP *runtime* takes care of thread management, forking, joining, etc.
- Specify number of threads via environment variable `OMP_NUM_THREADS`
- In parallel region, use: `omp_get_thread_num()` and `omp_get_num_threads()`

```
int
main()
{
  /* ...
     work to do outside parallel region
     ... */
#pragma omp parallel
  {
    /* ...
       work to do in parallel
       ... */
  }
  /* ...
     more work outside parallel region
     ... */
  return 0;
}
```

```
#include <omp.h>
...
/* Return a unique thread ID for each thread */
int tid = omp_get_thread_num();
...
/* Return the total number of threads */
int nth = omp_get_num_threads();
```

OpenMP introduction

Compiling and running

- Using GNU or Intel compilers, compile via:

```
[ikoutsou@front02 ~]$ cc -fopenmp program.c -o program
```

OpenMP introduction

Compiling and running

- Using GNU or Intel compilers, compile via:

```
[ikoutsou@front02 ~]$ cc -fopenmp program.c -o program
```

- Note that, depending on the compiler, the `#pragma` may not cause an error **even** if you accidentally omit `-fopenmp`. You will just produce a scalar code.

OpenMP introduction

Compiling and running

- Using GNU or Intel compilers, compile via:

```
[ikoutsou@front02 ~]$ cc -fopenmp program.c -o program
```

- Note that, depending on the compiler, the `#pragma` may not cause an error **even** if you accidentally omit `-fopenmp`. You will just produce a scalar code.
- Run via:

```
[ikoutsou@front02 ~]$ export OMP_NUM_THREADS=8
[ikoutsou@front02 ~]$ ./program
```

or

```
[ikoutsou@front02 ~]$ OMP_NUM_THREADS=8 ./program
```

OpenMP introduction

Compiling and running

- Using GNU or Intel compilers, compile via:

```
[ikoutsou@front02 ~]$ cc -fopenmp program.c -o program
```

- Note that, depending on the compiler, the `#pragma` may not cause an error **even** if you accidentally omit `-fopenmp`. You will just produce a scalar code.
- Run via:

```
[ikoutsou@front02 ~]$ export OMP_NUM_THREADS=8
[ikoutsou@front02 ~]$ ./program
```

or

```
[ikoutsou@front02 ~]$ OMP_NUM_THREADS=8 ./program
```

This will run on the frontend node.

OpenMP introduction

Example: every thread says hi

- Make a directory for this part of the lesson, e.g. `l03`:

```
[ikoutsou@front02 SDS406]$ mkdir l03
[ikoutsou@front02 SDS406]$ cd l03
```

- Copy first exercise (`ex01`):

```
[ikoutsou@front02 l03]$ cp -r /onyx/data/sds406f24/l03/ex01 .
[ikoutsou@front02 l03]$ cd ex01
```

- Inspect file `a.c`, compile it, and run on one node with `srun`:

```
[ikoutsou@front02 l03]$ more a.c
...
[ikoutsou@front02 l03]$ srun -N 1 -p p100 ./a
```

OpenMP introduction

Example: every thread says hi

Now, let's add a parallel region around the print statement:

- Add the parallel region:

```
#include <stdio.h>

int
main()
{
#pragma omp parallel
    {
        printf("Hi\n");
    }
    return 0;
}
```

- Compile, adding the `-fopenmp` option, then run:

```
[ikoutsou@front02 l03]$ cc -fopenmp -o a a.c
[ikoutsou@front02 l03]$ srun -N 1 -p p100 ./a
Hi
Hi
```

you should see 2 `Hi`s

OpenMP introduction

Example: every thread says hi

The *default number of threads* is 2, but we can control this with `OMP_NUM_THREADS`:

- Set `OMP_NUM_THREADS` before running. No need to compile again:

```
[ikoutsou@front02 l03]$ OMP_NUM_THREADS=1 srun -N 1 -p p100 ./a
Hi
[ikoutsou@front02 l03]$ OMP_NUM_THREADS=2 srun -N 1 -p p100 ./a
Hi
Hi
[ikoutsou@front02 l03]$ OMP_NUM_THREADS=3 srun -N 1 -p p100 ./a
Hi
Hi
Hi
```

- You can also set `OMP_NUM_THREADS` to something larger than the number of physical cores. You will simply be *over-subscribing* the cores, i.e. more than one thread will run per core.

```
[ikoutsou@front02 l03]$ OMP_NUM_THREADS=256 srun -N 1 -p p100 ./a
Hi
... (254 lines suppressed)
Hi
```

OpenMP introduction

Example: every thread says hi

Now let's see how to use the OpenMP API. We want each thread that says `Hi`, to also write its thread id and the total number of threads.

- Add the following:
    1. Include `<omp.h>` in the beginning of the source code
    2. Get the thread id with `omp_get_thread_num()`
    3. Get the number of threads with `omp_get_num_threads()`

OpenMP introduction

Example: every thread says hi

```c
#include <stdio.h>
#include <omp.h>

int
main()
{
#pragma omp parallel
    {
        int tid = omp_get_thread_num();
        int nth = omp_get_num_threads();
        printf("Hi, I am thread: %2d of %2d\n", tid, nth);
    }
    return 0;
}
```

- Compile and run as usual. You should see something like:

```
[ikoutsou@front02 l03]$ OMP_NUM_THREADS=5 srun -N 1 -p p100 ./a
Hi, I am thread:  0 of  5
Hi, I am thread:  3 of  5
Hi, I am thread:  4 of  5
Hi, I am thread:  1 of  5
Hi, I am thread:  2 of  5
```

- Note that the order by which each thread reaches the `printf()` statement is **non-deterministic**
- Indeed, you should make no assumptions on the order by which each thread runs

OpenMP overview

We will overview some OpenMP pragmas that will be used in the exercises and examples that follow

- The list is non-exhaustive. We will look at a subset of pragmas that are more commonly used
- The latest specification is OpenMP version 5. The full reference can be found here: OpenMP reference guide
- Note that OpenMP is a *specification*. It defines a set of pragmas and their functionality. Most compilers implement some version of the specification, but not all implement the latest version

OpenMP overview

- For loops

```
#pragma omp parallel for
for(int i=0; i<n; i++){
        ...
}
```

  will split the `n` iterations over the available threads

- Static scheduling, e.g.:

```
#pragma omp parallel for schedule(static, 10)
```

  a chunk is 10 iterations. Threads receive a chunk to work on in order.

- Dynamic scheduling, e.g.:

```
#pragma omp parallel for schedule(dynamic, 10)
```

  a chunk is 10 iterations. The next available thread receives a chunk to work on until all are exhausted.

- Guided scheduling, e.g.:

```
#pragma omp parallel for schedule(guided)
```

  chunk size is modified as iterations are consumed.

OpenMP overview

- For loops

```
#pragma omp parallel
{
#pragma omp for
    for(int i=0; i<n; i++){
        ...
    }
}
```

use within a parallel region, i.e. when a parallel region is already open.

- Reduction within for loop:

```
int sum_variable = 0;
#pragma omp parallel for reduction(+: sum_variable)
for(int i=0; i<n; i++){
    sum_variable +=  ... ;
    ...
}
```

outside the parallel region, `sum_variable` contains the sum over all threads

OpenMP overview

- Critical regions: each thread should run the region one-at-a-time

```
#pragma omp parallel
{
    #pragma omp critical
    {
        ... code to be run by each thread, one-at-a-time ...
    }
}
```

  of course, critical regions are **serialized**

- Single regions: within a parallel region, run by one thread

```
#pragma omp parallel
{
    #pragma omp single
    {
        printf("Hi\n");
    }
}
```

OpenMP overview

- Tasks: define a block of code, a *task* to be run by a thread:

```
#pragma omp task
{
    ...
}
```

- Usually we want one thread to create tasks. This is done within a `single` region

```
int a = 1;
int b = 2;
#pragma omp parallel
{
#pragma omp single
{
    #pragma omp task
    {
        /* This task is created by the thread that enters the single region
           and will be consumed by one of the threads of the pool */
        a = a+1;
    }
    #pragma omp task
    {
        /* This task is created by the thread that enters the single region
           and will be consumed by one of the threads of the pool */
        b = b+1;
    }
}
}
```

OpenMP overview

- Data sharing attributes

```
int a = 1;
int b = 2;
#pragma omp parallel private(a) shared(b)
{
    ...
}
```

- Each thread will have a local copy of `a`. Modifying `a` within the parallel region?
- The variable `b` is shared between threads. Each thread can modify it and all threads will see the same data
- You can also set a default attribute for data sharing

```
int a = 1, b = 2, c = 3, d = 4, e = 5;
# pragma omp parallel default(shared) private(b)
{
    ...
}
```

- All variables are shared, except `b` which is private

OpenMP Examples

Data sharing example

- Copy `ex02` as before:

```
[ikoutsou@front02 ex01]$ cd ../
[ikoutsou@front02 l03]$ cp -r /onyx/data/sds406f24/l03/ex02 .
[ikoutsou@front02 l03]$ cd ex02
```

- Inspect, compile, and run `a.c`:

```
[ikoutsou@front02 ex02]$ cc -fopenmp -o a a.c
```

OpenMP Examples

Data sharing example

- Copy `ex02` as before:

```
[ikoutsou@front02 ex01]$ cd ../
[ikoutsou@front02 l03]$ cp -r /onyx/data/sds406f24/l03/ex02 .
[ikoutsou@front02 l03]$ cd ex02
```

- Inspect, compile, and run `a.c`:

```
[ikoutsou@front02 ex02]$ cc -fopenmp -o a a.c
```

```
[ikoutsou@front02 l03]$ OMP_NUM_THREADS=5 srun -N 1 -p p100 ./a
Thread:  2 of  5, some_var = 42
Thread:  4 of  5, some_var = 42
Thread:  0 of  5, some_var = 42
Thread:  1 of  5, some_var = 42
Thread:  3 of  5, some_var = 42
```

  all threads have `some_var` set to the value `42`

OpenMP Examples

Data sharing example

- Now change the code so that the variable is modified within the parallel block, for example:

```
#include <stdio.h>
#include <omp.h>

int
main()
{
  int some_var = 42;
#pragma omp parallel
  {
    int tid = omp_get_thread_num();
    some_var = tid;
    int nth = omp_get_num_threads();
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
  }
  return 0;
}
```

- Compile and run. Run it a few times.

OpenMP Examples

Data sharing example

- Now change the code so that the variable is modified within the parallel block, for example:

```
#include <stdio.h>
#include <omp.h>

int
main()
{
  int some_var = 42;
#pragma omp parallel
  {
    int tid = omp_get_thread_num();
    some_var = tid;
    int nth = omp_get_num_threads();
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
  }
  return 0;
}
```

- Compile and run. Run it a few times.
- Try adding a small delay, e.g.:

```
    ...
    int tid = omp_get_thread_num();
    some_var = tid;
    int nth = omp_get_num_threads();
    sleep(1);
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
    ...
```

OpenMP Examples

Data sharing example

- Set the variable to private, to avoid this *race condition*

```c
#include <stdio.h>
#include <omp.h>

int
main()
{
  int some_var = 42;
#pragma omp parallel private(some_var)
  {
    int tid = omp_get_thread_num();
    some_var = tid;
    int nth = omp_get_num_threads();
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
  }
  return 0;
}
```

OpenMP Examples

Data sharing example

- Set the variable to private, to avoid this *race condition*

```c
#include <stdio.h>
#include <omp.h>

int
main()
{
  int some_var = 42;
#pragma omp parallel private(some_var)
  {
    int tid = omp_get_thread_num();
    some_var = tid;
    int nth = omp_get_num_threads();
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
  }
  return 0;
}
```

```
[ikoutsou@front02 ex02]$ OMP_NUM_THREADS=5 srun -N 1 -p p100 ./a
Thread:  0 of  5, some_var = 0
Thread:  4 of  5, some_var = 4
Thread:  2 of  5, some_var = 2
Thread:  3 of  5, some_var = 3
Thread:  1 of  5, some_var = 1
```

OpenMP Examples

Data sharing example

- Set the variable to private, to avoid this *race condition*

```
#include <stdio.h>
#include <omp.h>

int
main()
{
  int some_var = 42;
#pragma omp parallel private(some_var)
  {
    int tid = omp_get_thread_num();
    some_var = tid;
    int nth = omp_get_num_threads();
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
  }
  return 0;
}
```

```
[ikoutsou@front02 ex02]$ OMP_NUM_THREADS=5 srun -N 1 -p p100 ./a
Thread:  0 of  5, some_var = 0
Thread:  4 of  5, some_var = 4
Thread:  2 of  5, some_var = 2
Thread:  3 of  5, some_var = 3
Thread:  1 of  5, some_var = 1
```

- What is the value of `some_var` after the parallel region ends?

OpenMP Examples

Data sharing example

- Initial value of a private variable

```c
#include <stdio.h>
#include <omp.h>
#include <unistd.h>

int
main()
{
  int some_var = 42;
#pragma omp parallel private(some_var)
  {
    int tid = omp_get_thread_num();
    some_var = some_var+tid;
    int nth = omp_get_num_threads();
    sleep(1);
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
  }
  return 0;
}
```

- What does this code to produce?

OpenMP Examples

Data sharing example

- Initial value of a private variable

```c
#include <stdio.h>
#include <omp.h>
#include <unistd.h>

int
main()
{
  int some_var = 42;
#pragma omp parallel private(some_var)
  {
    int tid = omp_get_thread_num();
    some_var = some_var+tid;
    int nth = omp_get_num_threads();
    sleep(1);
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
  }
  return 0;
}
```

- What does this code to produce?
- `some_var` is uninitialized in the parallel region. Its initial value is undefined and not guaranteed to be zero.

OpenMP Examples

Data sharing example

- Initial value of a private variable

```c
#include <stdio.h>
#include <omp.h>
#include <unistd.h>

int
main()
{
  int some_var = 42;
#pragma omp parallel private(some_var)
  {
    int tid = omp_get_thread_num();
    some_var = some_var+tid;
    int nth = omp_get_num_threads();
    sleep(1);
    printf("Thread: %2d of %2d, some_var = %d\n", tid, nth, some_var);
  }
  return 0;
}
```

- What does this code to produce?
- `some_var` is uninitialized in the parallel region. Its initial value is undefined and not guaranteed to be zero.
- Replace `private(some_var)` with `firstprivate(some_var)`.

OpenMP Examples

Data sharing example

- Shared vs private array

```
#include <stdio.h>
#include <omp.h>

int
main()
{
  int arr[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
#pragma omp parallel shared(arr)
  {
    int tid = omp_get_thread_num();
    int nth = omp_get_num_threads();
    arr[tid] = tid;
  }

  for(int i=0; i<32; i++) {
    printf("arr[%d] = %d\n", i, arr[i]);
  }
  return 0;
}
```

- What do you expect the output of this program to be?

OpenMP Examples

Data sharing example

- Shared vs private array

```
#include <stdio.h>
#include <omp.h>

int
main()
{
  int arr[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
#pragma omp parallel shared(arr)
  {
    int tid = omp_get_thread_num();
    int nth = omp_get_num_threads();
    arr[tid] = tid;
  }

  for(int i=0; i<32; i++) {
    printf("arr[%d] = %d\n", i, arr[i]);
  }
  return 0;
}
```

- What do you expect the output of this program to be?

```
[ikoutsou@front02 ex02]$ OMP_NUM_THREADS=5 srun -N 1 -p p100  ./a
arr[0] = 0
arr[1] = 1
arr[2] = 2
arr[3] = 3
arr[4] = 4
arr[5] = 0
 ...
arr[31] = 0
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

- Copy `ex03` as before:

```
[ikoutsou@front02 ex02]$ cd ../
[ikoutsou@front02 l03]$ cp -r /onyx/data/sds406f24/l03/ex03 .
[ikoutsou@front02 l03]$ cd ex03
```

- Inspect, compile, and run `axpy.c`:

```
[ikoutsou@front02 ex03]$ cc -std=c99 -fopenmp -o axpy axpy.c
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=5 srun -N 1 -p p100 ./axpy $((1024*1024*32))
t0 = 0.184823 sec, t1 = 0.183408 sec, diff z norm = 0.000000e+00
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

- Copy `ex03` as before:

```
[ikoutsou@front02 ex02]$ cd ../
[ikoutsou@front02 l03]$ cp -r /onyx/data/sds406f24/l03/ex03 .
[ikoutsou@front02 l03]$ cd ex03
```

- Inspect, compile, and run `axpy.c`:

```
[ikoutsou@front02 ex03]$ cc -std=c99 -fopenmp -o axpy axpy.c
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=5 srun -N 1 -p p100 ./axpy $((1024*1024*32))
t0 = 0.184823 sec, t1 = 0.183408 sec, diff z norm = 0.000000e+00
```

- Use an OpenMP pragma to parallelize the second occurrence of the main for loop

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Change:

```
for(int i=0; i<n; i++) {
  z_1[i] = a*x[i] + y[i];
}
```

```
#pragma omp parallel for
for(int i=0; i<n; i++) {
  z_1[i] = a*x[i] + y[i];
}
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Change:

```
for(int i=0; i<n; i++) {
  z_1[i] = a*x[i] + y[i];
}
```

```
#pragma omp parallel for
for(int i=0; i<n; i++) {
  z_1[i] = a*x[i] + y[i];
}
```

- It's also useful to report the total number of threads:

```
#pragma omp parallel
{
    int nth = omp_get_num_threads();
#pragma omp single
    printf(" nth = %2d, t0 = %lf sec, t1 = %lf sec, diff z norm = %e\n", nth, t0, t1, norm);
}
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1, 2, and 4

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1, 2, and 4

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=1 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  1, t0 = 0.130369 sec, t1 = 0.134538 sec, diff z norm = 0.000000e+00
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1, 2, and 4

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=1 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  1, t0 = 0.130369 sec, t1 = 0.134538 sec, diff z norm = 0.000000e+00
```

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=2 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  2, t0 = 0.130446 sec, t1 = 0.109080 sec, diff z norm = 0.000000e+00
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1, 2, and 4

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=1 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  1, t0 = 0.130369 sec, t1 = 0.134538 sec, diff z norm = 0.000000e+00
```

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=2 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  2, t0 = 0.130446 sec, t1 = 0.109080 sec, diff z norm = 0.000000e+00
```

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=4 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  4, t0 = 0.129832 sec, t1 = 0.110167 sec, diff z norm = 0.000000e+00
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1, 2, and 4

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=1 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  1, t0 = 0.130369 sec, t1 = 0.134538 sec, diff z norm = 0.000000e+00
```

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=2 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  2, t0 = 0.130446 sec, t1 = 0.109080 sec, diff z norm = 0.000000e+00
```

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=4 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  4, t0 = 0.129832 sec, t1 = 0.110167 sec, diff z norm = 0.000000e+00
```

- If you see no improvement (like above), this is because of default configuration of Slurm which is reserving only one CPU core for all threads

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1, 2, and 4

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=1 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  1, t0 = 0.130369 sec, t1 = 0.134538 sec, diff z norm = 0.000000e+00
```

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=2 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  2, t0 = 0.130446 sec, t1 = 0.109080 sec, diff z norm = 0.000000e+00
```

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=4 srun -N 1 -p p100 ./axpy $((1024*1024*32))
nth =  4, t0 = 0.129832 sec, t1 = 0.110167 sec, diff z norm = 0.000000e+00
```

- If you see no improvement (like above), this is because of default configuration of Slurm which is reserving only one CPU core for all threads
- Use `--cpus-per-task`

```
[ikoutsou@front02 ex03]$ OMP_NUM_THREADS=4 srun -N 1 --cpus-per-task=8  -p p100 ./axpy $((1024*1024*32))
nth =  4, t0 = 0.130742 sec, t1 = 0.036430 sec, diff z norm = 0.000000e+00
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1,...,8. How does the runtime scale?

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1,...,8. How does the runtime scale?

```
[ikoutsou@front02 ex03]$ for ((n=1;n⩽8;n++))
> do
> OMP_NUM_THREADS=$n srun -N 1 --cpus-per-task=16 -p p100 ./axpy $((1024*1024*32))
> done
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

- Run for `OMP_NUM_THREADS` from 1,...,8. How does the runtime scale?

```
[ikoutsou@front02 ex03]$ for ((n=1;n<=8;n++))
> do
> OMP_NUM_THREADS=$n srun -N 1 --cpus-per-task=16 -p p100 ./axpy $((1024*1024*32))
> done
```

```
 nth =  1, t0 = 0.131535 sec, t1 = 0.133377 sec, diff z norm = 0.000000e+00
 nth =  2, t0 = 0.129817 sec, t1 = 0.069848 sec, diff z norm = 0.000000e+00
 nth =  3, t0 = 0.129444 sec, t1 = 0.047347 sec, diff z norm = 0.000000e+00
 nth =  4, t0 = 0.130564 sec, t1 = 0.036938 sec, diff z norm = 0.000000e+00
 nth =  5, t0 = 0.130000 sec, t1 = 0.029731 sec, diff z norm = 0.000000e+00
 nth =  6, t0 = 0.129202 sec, t1 = 0.025233 sec, diff z norm = 0.000000e+00
 nth =  7, t0 = 0.135529 sec, t1 = 0.023321 sec, diff z norm = 0.000000e+00
 nth =  8, t0 = 0.130832 sec, t1 = 0.022796 sec, diff z norm = 0.000000e+00
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

How many floating point operations are carried out?

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

How many floating point operations are carried out?

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

How many floating point operations are carried out?

- For each iteration, i.e. for each element `i`

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

How many floating point operations are carried out?

- For each iteration, i.e. for each element `i`
  - One multiplication ($a x_i$)

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

How many floating point operations are carried out?

- For each iteration, i.e. for each element `i`
  - One multiplication ($a x_i$)
  - One addition ($+ y_i$)

OpenMP Examples

Linear algebra

Operation:

$$z_i = ax_i + y_i$$

How many floating point operations are carried out?

- For each iteration, i.e. for each element `i`
  - One multiplication (ax_i)
  - One addition ($+y_i$)
  - Therefore, in total, `2*n` floating point operations, irrespective of the number of processes

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

How many floating point operations are carried out?

- For each iteration, i.e. for each element `i`
  - One multiplication ($a x_i$)
  - One addition ($+ y_i$)
  - Therefore, in total, `2*n` floating point operations, irrespective of the number of processes
- Modify the code to also report the *sustained Gflop/s* of the parallelized `axpy` part

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

How many floating point operations are carried out?

- For each iteration, i.e. for each element `i`
  - One multiplication ($a x_i$)
  - One addition ($+ y_i$)
  - Therefore, in total, `2*n` floating point operations, irrespective of the number of processes
- Modify the code to also report the *sustained Gflop/s* of the parallelized `axpy` part

```
double p = 2.0*n/t1/1e9;
printf(" nth = %2d, t0 = %lf sec, t1 = %lf sec, diff z norm = %e, perf = %lf Gflop/s\n", nth, t0, t1, norm, p);
```

OpenMP Examples

Linear algebra

Operation:

$$z_i = a x_i + y_i$$

How many floating point operations are carried out?

- For each iteration, i.e. for each element `i`
  - One multiplication ($a x_i$)
  - One addition ($+ y_i$)
  - Therefore, in total, `2*n` floating point operations, irrespective of the number of processes
- Modify the code to also report the *sustained Gflop/s* of the parallelized `axpy` part

```
double p = 2.0*n/t1/1e9;
printf(" nth = %2d, t0 = %lf sec, t1 = %lf sec, diff z norm = %e, perf = %lf Gflop/s\n", nth, t0, t1, norm, p);
```

```
 nth =  1, t0 = 0.129865 sec, t1 = 0.132042 sec, diff z norm = 0.000000e+00, perf = 0.508239 Gflop/s
 nth =  2, t0 = 0.130650 sec, t1 = 0.069006 sec, diff z norm = 0.000000e+00, perf = 0.972505 Gflop/s
 nth =  3, t0 = 0.132397 sec, t1 = 0.046861 sec, diff z norm = 0.000000e+00, perf = 1.432086 Gflop/s
 nth =  4, t0 = 0.130751 sec, t1 = 0.036120 sec, diff z norm = 0.000000e+00, perf = 1.857945 Gflop/s
 nth =  5, t0 = 0.129818 sec, t1 = 0.029459 sec, diff z norm = 0.000000e+00, perf = 2.278043 Gflop/s
 nth =  6, t0 = 0.132037 sec, t1 = 0.025257 sec, diff z norm = 0.000000e+00, perf = 2.657054 Gflop/s
 nth =  7, t0 = 0.129651 sec, t1 = 0.022796 sec, diff z norm = 0.000000e+00, perf = 2.943868 Gflop/s
 nth =  8, t0 = 0.130022 sec, t1 = 0.022132 sec, diff z norm = 0.000000e+00, perf = 3.032221 Gflop/s
```

OpenMP Examples

Linear algebra

Dot product operation:

$$r = x^T y = \sum_{i=0}^{n-1} x_i y_i$$

- Copy `ex04` as before:

```
[ikoutsou@front02 ex03]$ cd ../
[ikoutsou@front02 ex03]$ cp -r /onyx/data/sds406f24/l03/ex04 .
[ikoutsou@front02 ex03]$ cd ex04
```

OpenMP Examples

Linear algebra

Dot product operation:

$$r = x^T y = \sum_{i=0}^{n-1} x_i y_i$$

- Copy `ex04` as before:

```
[ikoutsou@front02 ex03]$ cd ../
[ikoutsou@front02 ex03]$ cp -r /onyx/data/sds406f24/l03/ex04 .
[ikoutsou@front02 ex03]$ cd ex04
```

- Inspect, compile, and run `xdoty.c`:

```
[ikoutsou@front02 ex04]$ cc -std=c99 -o xdoty xdoty.c
[ikoutsou@front02 ex04]$ srun -N 1 -p p100 ./xdoty $((32*1024*1024))
t0 = 0.086199 sec, t1 = 0.087750 sec, norms = 8.387960e+06, 8.387960e+06
```

OpenMP Examples

Linear algebra

Dot product operation:

$$r = x^T y = \sum_{i=0}^{n-1} x_i y_i$$

- Copy `ex04` as before:

```
[ikoutsou@front02 ex03]$ cd ../
[ikoutsou@front02 ex03]$ cp -r /onyx/data/sds406f24/l03/ex04 .
[ikoutsou@front02 ex03]$ cd ex04
```

- Inspect, compile, and run `xdoty.c`:

```
[ikoutsou@front02 ex04]$ cc -std=c99 -o xdoty xdoty.c
[ikoutsou@front02 ex04]$ srun -N 1 -p p100 ./xdoty $((32*1024*1024))
t0 = 0.086199 sec, t1 = 0.087750 sec, norms = 8.387960e+06, 8.387960e+06
```

- Use an OpenMP pragma to parallelize the second occurrence of the main for loop

OpenMP Examples

Linear algebra

Dot product operation:

$$r = x^T y = \sum_{i=0}^{n-1} x_i y_i$$

```
double norm_1 = 0;

for(int i=0; i<n; i++) {
   norm_1 += x[i]*y[i];
}
```

```
double norm_1 = 0;
#pragma omp parallel for reduction(+:norm_1)
for(int i=0; i<n; i++) {
   norm_1 += x[i]*y[i];
}
```

OpenMP Examples

Linear algebra

Dot product operation:

$$r = x^T y = \sum_{i=0}^{n-1} x_i y_i$$

```
double norm_1 = 0;

for(int i=0; i<n; i++) {
   norm_1 += x[i]*y[i];
}
```

```
double norm_1 = 0;
#pragma omp parallel for reduction(+:norm_1)
for(int i=0; i<n; i++) {
   norm_1 += x[i]*y[i];
}
```

- Try, e.g. 4 threads

```
[ikoutsou@front02 ex04]$ OMP_NUM_THREADS=4 srun -N 1 --cpus-per-task=16 -p p100 ./xdoty $((32*1024*1024))
nthr =  4, t0 = 0.086064 sec, t1 = 0.023266 sec, norms = 8.387960e+06, 8.387960e+06
```

OpenMP Examples

Linear algebra

Dot product operation:

$$r = x^T y = \sum_{i=0}^{n-1} x_i y_i$$

How many floating point operations are carried out?

OpenMP Examples

Linear algebra

Dot product operation:

$$r = x^T y = \sum_{i=0}^{n-1} x_i y_i$$

How many floating point operations are carried out?

- Modify the code to also report the *sustained Gflop/s* of the parallelized `xdoty` part

OpenMP Examples

Linear algebra

Operation:

$$r = \sum_{i=0}^{n-1} x_i \cdot y_i$$

OpenMP Examples

Linear algebra

Operation:

$$r = \sum_{i=0}^{n-1} x_i \cdot y_i$$

Number of *floating point operations*

OpenMP Examples

Linear algebra

Operation:

$$r = \sum_{i=0}^{n-1} x_i \cdot y_i$$

Number of *floating point operations*

- For each iteration, i.e. for each element `i`

OpenMP Examples

Linear algebra

Operation:

$$r = \sum_{i=0}^{n-1} x_i \cdot y_i$$

Number of *floating point operations*

- For each iteration, i.e. for each element `i`
  - One multiplication ($x_i \cdot y_i$)

OpenMP Examples

Linear algebra

Operation:

$$r = \sum_{i=0}^{n-1} x_i \cdot y_i$$

Number of *floating point operations*

- For each iteration, i.e. for each element `i`
  - One multiplication ($x_i \cdot y_i$)
  - One addition ($r = r + x_i \cdot y_i$)

OpenMP Examples

Linear algebra

Operation:

$$r = \sum_{i=0}^{n-1} x_i \cdot y_i$$

Number of *floating point operations*

- For each iteration, i.e. for each element `i`
  - One multiplication ($x_i \cdot y_i$)
  - One addition ($r = r + x_i \cdot y_i$)
- Therefore, a total of 2 floating point operations per iteration

OpenMP Examples

Linear algebra

Operation:

$$r = \sum_{i=0}^{n-1} x_i \cdot y_i$$

Number of *floating point operations*

- For each iteration, i.e. for each element `i`
  - One multiplication ($x_i \cdot y_i$)
  - One addition ($r = r + x_i \cdot y_i$)
- Therefore, a total of 2 floating point operations per iteration
- For `n` iterations, `2*n/t1/1e9` in Gflops/s

OpenMP for linear algebra

```
[ikoutsou@front02 ex04]$ for ((n=1;n⩽8;n++))
> do
> OMP_NUM_THREADS=$n srun -N 1 --cpus-per-task=16 -p p100 ./xdoty $((1024*1024*32))
> done
```

OpenMP for linear algebra

```
[ikoutsou@front02 ex04]$ for ((n=1;n⩽8;n++))
> do
> OMP_NUM_THREADS=$n srun -N 1 --cpus-per-task=16 -p p100 ./xdoty $((1024*1024*32))
> done
```

n_{OMP}	`axpy` [Gflop/s]	`xdoty` [Gflop/s]
1	0.51	0.78
2	0.98	1.53
3	1.43	2.21
4	1.83	2.88
5	2.28	3.59
6	2.61	4.28
7	2.93	4.93
8	2.98	5.28

OpenMP for linear algebra

```
[ikoutsou@front02 ex04]$ for ((n=1;n⩽8;n++))
> do
> OMP_NUM_THREADS=$n srun -N 1 --cpus-per-task=16 -p p100 ./xdoty $((1024*1024*32))
> done
```

n_{OMP}	`axpy` [Gflop/s]	`xdoty` [Gflop/s]
1	0.51	0.78
2	0.98	1.53
3	1.43	2.21
4	1.83	2.88
5	2.28	3.59
6	2.61	4.28
7	2.93	4.93
8	2.98	5.28