CS 4504
Parallel and Distributed Computation

OpenMP

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Outline

- OpenMP introduction
  - Helloworld of OpenMP
- Performance evaluation
- Example: how to solve problems in OpenMP
  - Trapezoidal problem

OpenMP

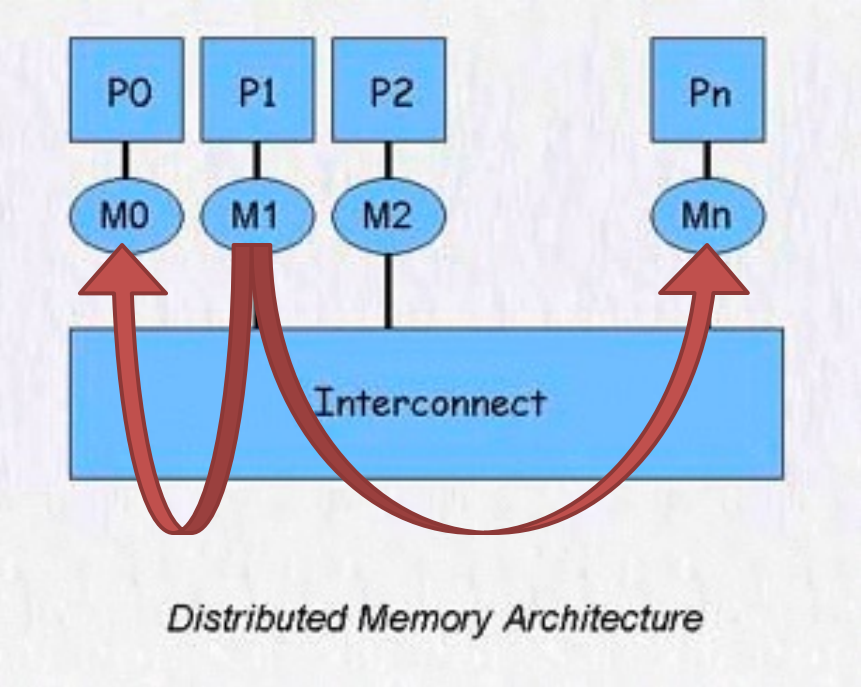
- MP = multiprocessing

https://www.openmp.org/

- An API for shared-memory parallel programming.

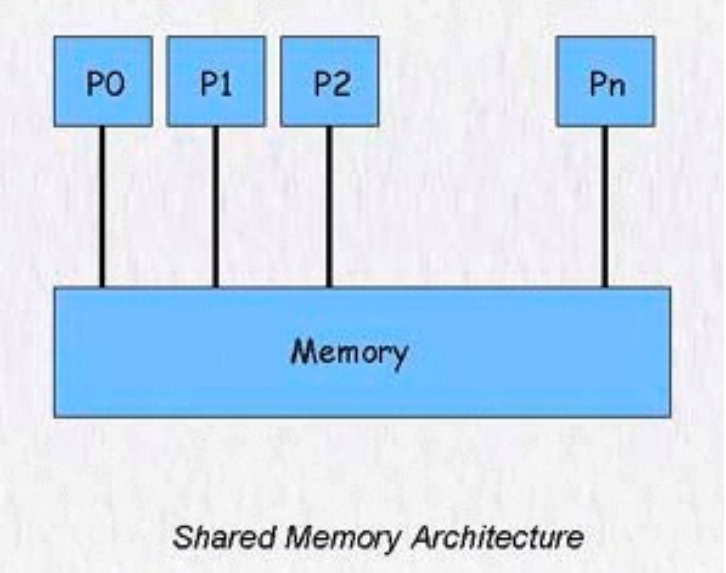
Distributed Memory

- Each processor has its own memory
- Parallel programming by message passing (MPI)



Shared Memory

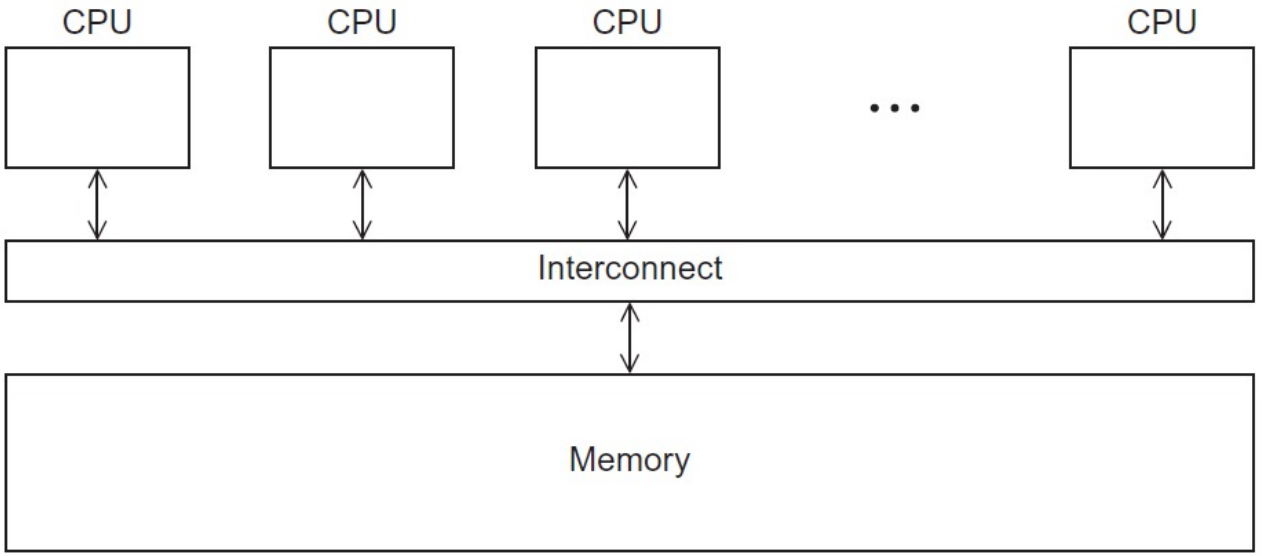
- Processors shared memory
- Parallel programming approaches
  - message passing (MPI)
  - pthread
  - directives-based interface - OpenMP



OpenMP

- Designed for systems in which each thread or process can potentially have access to ***all available memory***.

- System is viewed as a collection of ***cores or CPU's***, all of which have access to ***main memory***.



Pros and Cons Of OpenMP

- Pros
  - Prevalence of multi-core computers
  - Requires less code modification than using MPI
  - OpenMP directives can be treated as comments if OpenMP is not available
  - Directives can be added incrementally

Pros and Cons Of OpenMP

- Cons
  - OpenMP codes cannot be run on distributed memory computers (exception is Intel's OpenMP)
  - Requires a compiler that supports OpenMP (most do)
  - limited by the number of processors available on a single computer
  - rely more on parallelizable loops

Examples of Applications That Use OpenMP

- Applications
  - Matlab
  - Mathematica

https://www.wolfram.com/mathematica/

Example

https://github.com/kevinsuo/CS7172/blob/master/omp-helloworld.c

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

void Hello(void)
{
        int my_thread_ID = omp_get_thread_num();
        int thread_count = omp_get_num_threads();
        printf( "Hello from thread %d of %d\n", my_thread_ID , thread_count );
}

int main (int argc, char *argv[]) {

#pragma omp parallel
        Hello();

        return 0;
}
```

omp.h file

Return 0,1,2,3

Thread number is decided by core number

Example

- Compile
  - gcc omp-helloworld.c -o omp-helloworld.o -fopenmp

- Run
  - ./omp-helloworld.o

```
ksuo@ksuo-VirtualBox ~/cs7172> ./omp-helloworld.o
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 0 of 4
Hello from thread 3 of 4
```

Pragmas

https://github.com/kevinsuo/CS7172/blob/master/omp-helloworld.c

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

void Hello(void)
{
        int my_thread_ID = omp_get_thread_num();
        int thread_count = omp_get_num_threads();
        printf( "Hello from thread %d of %d\n", my_thread_ID , thread_count );
}

int main (int argc, char *argv[]) {

#pragma omp parallel
        Hello();

        return 0;
}
```

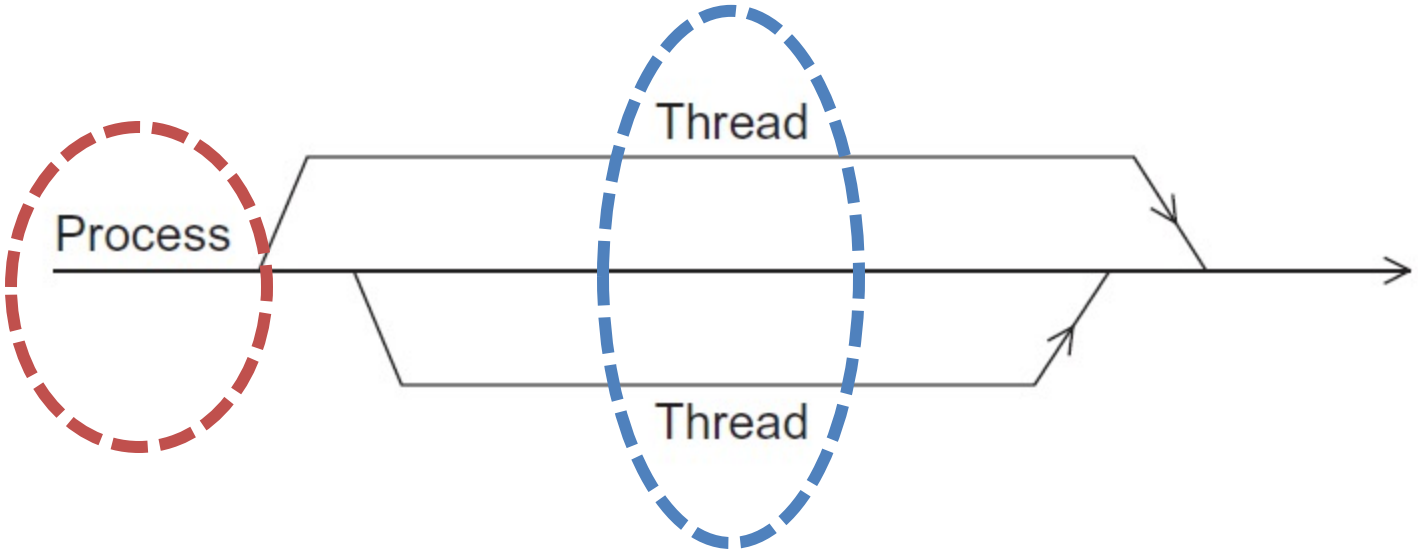
omp.h file

Return 0,1,2,3

Thread number is decided by core number

Example

https://github.com/kevinsuo/CS7172/blob/master/omp-helloworld.c



Master thread

worker thread

#pragma omp parallel

omp_get_thread_num
omp_get_num_threads
Get total number and rankID

Example

- Compile
  - gcc -fopenmp omp-helloworld.c -o omp-helloworld.o

```
ksuo@ksuo-VirtualBox ~/cs7172> ./omp-helloworld.o
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 0 of 4
Hello from thread 3 of 4
ksuo@ksuo-VirtualBox ~/cs7172> ./omp-helloworld.o
Hello from thread 0 of 4
Hello from thread 3 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
ksuo@ksuo-VirtualBox ~/cs7172> ./omp-helloworld.o
Hello from thread 0 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4
Hello from thread 1 of 4
```

Hello from thread 0
Hello from thread 1
Hello from thread 2
Hello from thread 3

possible outcomes

Hello from thread 1
Hello from thread 2
Hello from thread 0
Hello from thread 3

Hello from thread 3
Hello from thread 1
Hello from thread 2
Hello from thread 0

How to Compile and Run an OpenMP Program

Compiler	Compiler Options	Default behavior for # of threads (OMP_NUM_THREADS not set)
GNU (gcc, g++, gfortran)	-fopenmp	as many threads as available cores
Intel (icc ifort)	-openmp	as many threads as available cores
Portland Group (pgcc,pgCC,pgf77,pgf90)	-mp	one thread

From Sequential to Parallel in OpenMP

https://github.com/kevinsuo/CS7172/blob/master/serial-code.c

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

void SumForNumber()
{
        int a = 0;
        for (int i = 0; i<10000000; i++)
                a++;
}

int main()
{
        struct timespec start, end;
        clock_gettime(CLOCK_MONOTONIC, &start);

        for (int i = 0; i<100; i++)
                SumForNumber();

        clock_gettime(CLOCK_MONOTONIC, &end);

        double diff = 1000000000L * (end.tv_sec - start.tv_sec) + end.tv_nsec - start.tv_nsec;
        printf("time: %lf\n", diff);
        return 0;
}
```

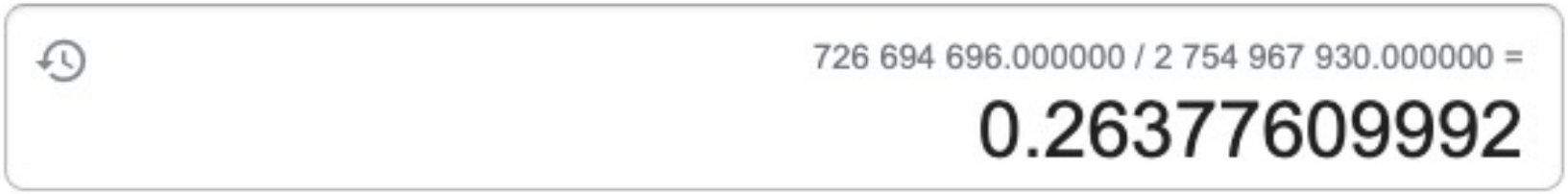
From Sequential to Parallel in OpenMP

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

void SumForNumber()
{
        int a = 0;
        for (int i = 0; i<10000000; i++)
                a++;
}

int main()
{
        struct timespec start, end;
        clock_gettime(CLOCK_MONOTONIC, &start);

#pragma omp parallel for
        for (int i = 0; i<100; i++)
                SumForNumber();

        clock_gettime(CLOCK_MONOTONIC, &end);

        double diff = 1000000000L * (end.tv_sec - start.tv_sec) + end.tv_nsec - start.tv_nsec;
        printf("time: %lf\n", diff);
        return 0;
}
```

From Sequential to Parallel in OpenMP

```
ksuo@LinuxKernel2 ~> gcc test.c -o test.o
ksuo@LinuxKernel2 ~> ./test.o
time: 2754967930.000000
```

```
ksuo@LinuxKernel2 ~> gcc test.c -o test.o -fopenmp
ksuo@LinuxKernel2 ~> ./test.o
time: 726694696.000000
```



Clause: #pragma omp parallel

- Each core creates one thread to execute the function
- The number of thread is determined by runtime system

```
int main (int argc, char *argv[]) {

#pragma omp parallel
        Hello();
```

Example: #pragma omp parallel num_threads(thread_count)

https://github.com/kevinsuo/CS7172/blob/master/omp-helloworld-2.c

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

void Hello(void)
{
        int my_thread_ID = omp_get_thread_num();
        int thread_count = omp_get_num_threads();
        printf( "Hello from thread %d of %d\n", my_thread_ID , thread_count );
}

int main (int argc, char *argv[])
{
        /* Get number of threads from command line */
        int thread_count = strtol(argv[1], NULL, 10);

#pragma omp parallel num_threads( thread_count )
        Hello();

        return 0;
}
```

Thread number is decided by user input

Clause: #pragma omp parallel num_threads(thread_count)

- The num_threads clause can be added to a parallel directive.

- It allows the programmer to specify the number of threads that should execute the following block.

```
#pragma omp parallel num_threads( thread_count )
        Hello();

        return 0;
}
```

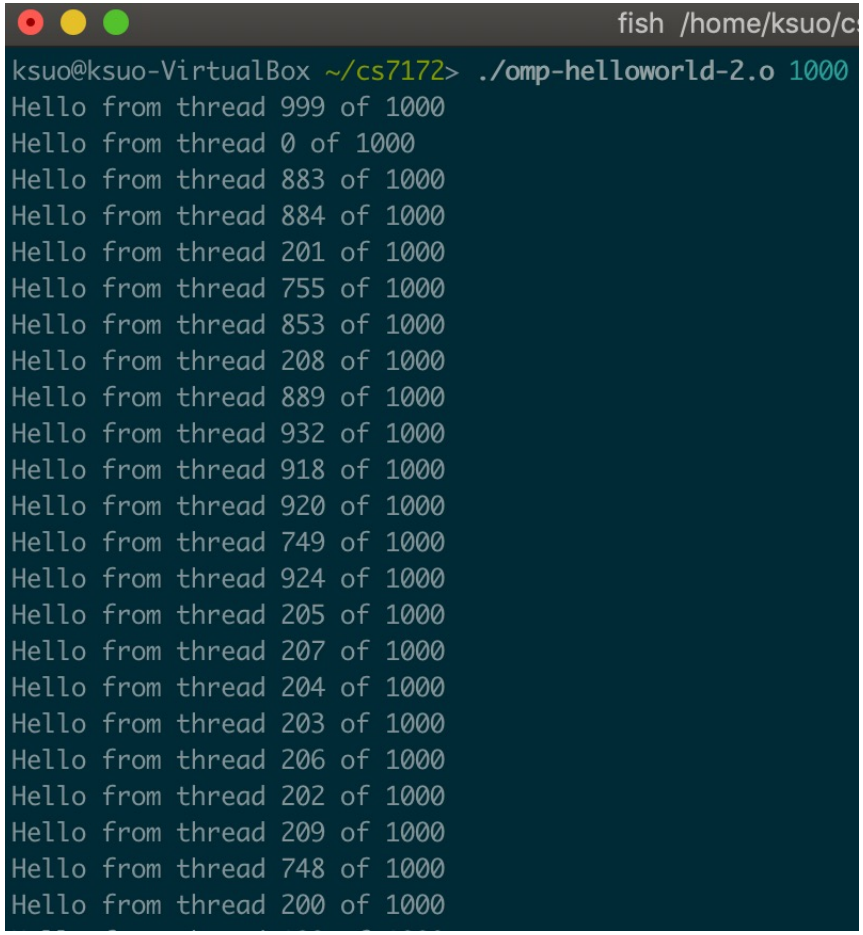
Thread number is decided by user input

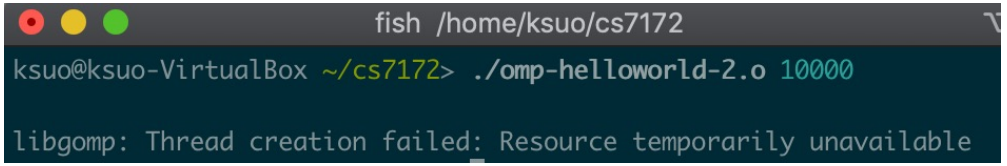
Example: #pragma omp parallel num_threads(thread_count)

- Compile
  - gcc -fopenmp -o omp-helloworld-2.o omp-helloworld-2.c

```
ksuo@ksuo-VirtualBox ~/cs7172> ./omp-helloworld-2.o 8
Hello from thread 3 of 8
Hello from thread 4 of 8
Hello from thread 1 of 8
Hello from thread 6 of 8
Hello from thread 5 of 8
Hello from thread 2 of 8
Hello from thread 7 of 8
Hello from thread 0 of 8
ksuo@ksuo-VirtualBox ~/cs7172> ./omp-helloworld-2.o 8
Hello from thread 0 of 8
Hello from thread 6 of 8
Hello from thread 1 of 8
Hello from thread 4 of 8
Hello from thread 3 of 8
Hello from thread 5 of 8
Hello from thread 7 of 8
Hello from thread 2 of 8
ksuo@ksuo-VirtualBox ~/cs7172> ./omp-helloworld-2.o 8
Hello from thread 1 of 8
Hello from thread 6 of 8
Hello from thread 2 of 8
Hello from thread 0 of 8
Hello from thread 5 of 8
Hello from thread 3 of 8
Hello from thread 7 of 8
Hello from thread 4 of 8
```

Of note…



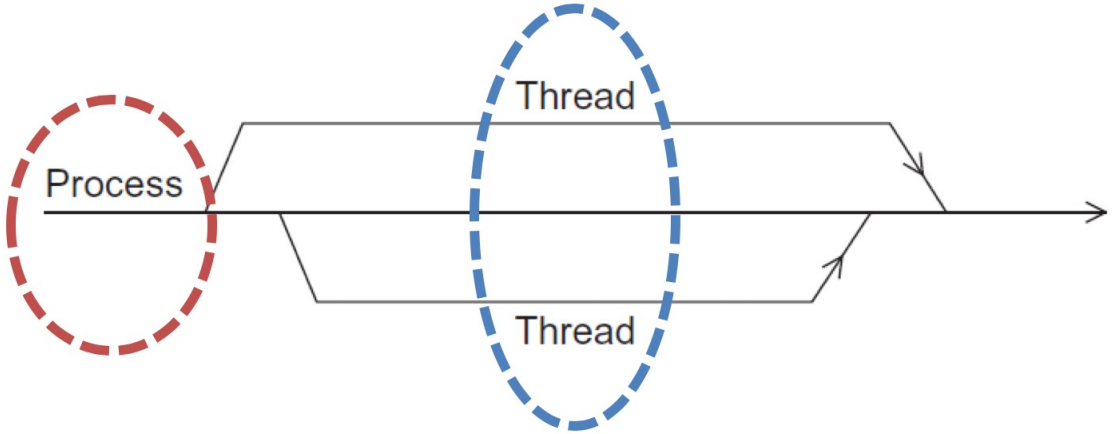


10000 threads fail!

Too many.

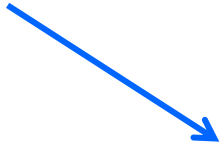
Some terminology

- In OpenMP parlance the collection of threads executing the parallel block — the original thread and the new threads — is called a team, the original thread is called the master, and the additional threads are called slaves.



In case the compiler doesn't support OpenMP

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



```
#ifdef _OPENMP
# include <omp.h>
#endif
```

In case the compiler doesn't support OpenMP

```
# ifdef _OPENMP
   int my_rank = omp_get_thread_num ( );
   int thread_count = omp_get_num_threads ( );
# else
   int my_rank = 0;
   int thread_count = 1;
# endif
```

Performance evaluation

Elapsed serial time

- In this case, you don't need to link in the MPI libraries.

- Returns time in **microseconds** elapsed from some point in the past.

```
#include "timer.h"
. . .
double now;
. . .
GET_TIME(now);
```

Elapsed serial time

```
#include "timer.h"
. . .
double start, finish;
. . .
GET_TIME(start);
/* Code to be timed */
. . .
GET_TIME(finish);
printf("Elapsed time = %e seconds\n", finish-start);
```

Elapsed serial time in nanoseconds

```c
#include <time.h>

{
        struct timespec start, end;

        clock_gettime(CLOCK_MONOTONIC, &start);

        //... do something

        clock_gettime(CLOCK_MONOTONIC, &end);

        u_int64_t diff = 1000000000L * (end.tv_sec - start.tv_sec) + end.tv_nsec - start.tv_nsec;

        printf("elapsed time = %llu nanoseconds\n", (long long unsigned int) diff);

}
```

Run-times of serial and parallel matrix-vector multiplication

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	4.1	16.0	64.0	270	1100
2	2.3	8.5	33.0	140	560
4	2.0	5.1	18.0	70	280
8	1.7	3.3	9.8	36	140
16	1.7	2.6	5.9	19	71

(Seconds)

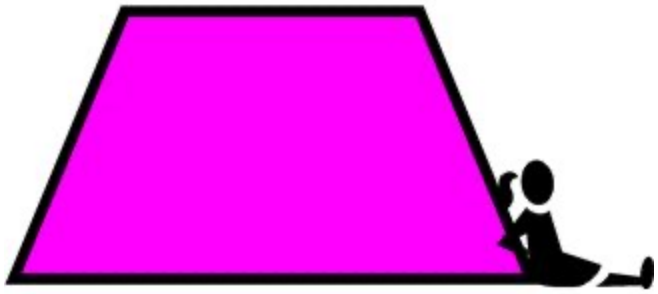
Speedup

$$S(n,p) = \frac{T_{\text{serial}}(n)}{T_{\text{parallel}}(n,p)}$$

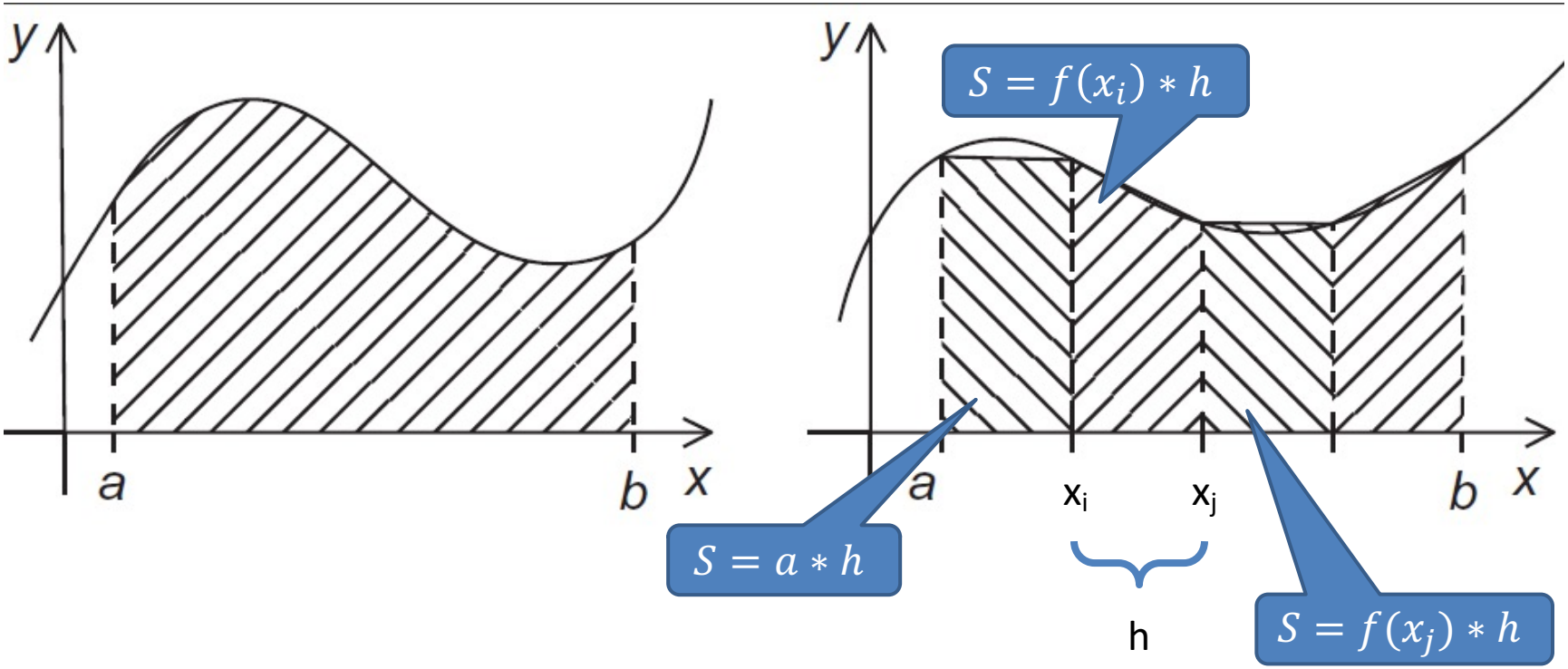
Speedups of Parallel Matrix-Vector Multiplication

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.0	1.0	1.0	1.0	1.0
2	1.8	1.9	1.9	1.9	2.0
4	2.1	3.1	3.6	3.9	3.9
8	2.4	4.8	6.5	7.5	7.9
16	2.4	6.2	10.8	14.2	15.5

The Trapezoidal Rule



The trapezoidal rule

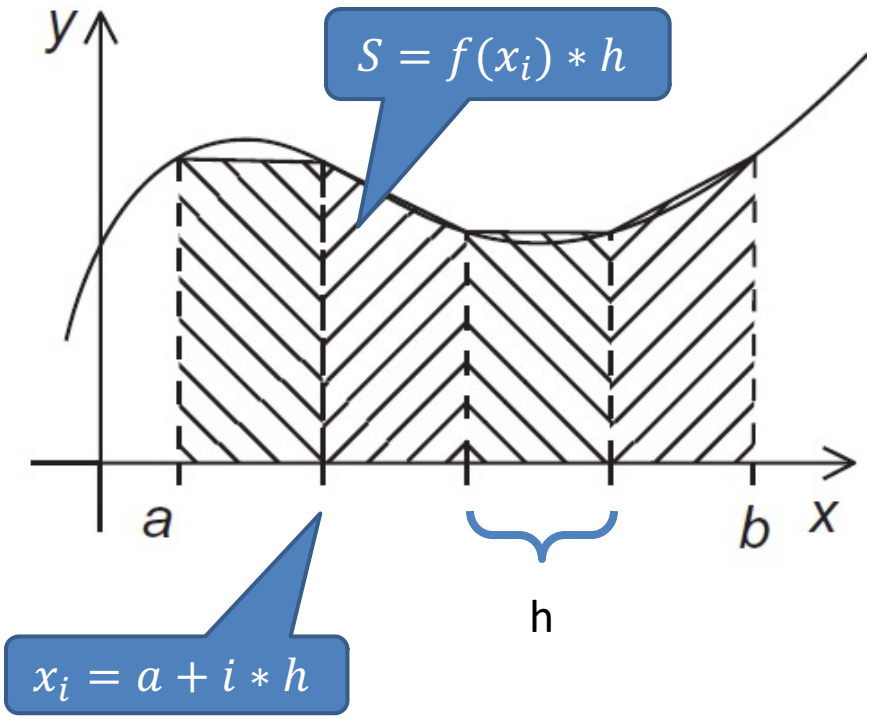


Here we used rectangle size to approximate calculate the size of trapezoid

Serial algorithm

https://github.com/kevinsuo/CS7172/blob/master/trapezoidal-serial.c

```
double f(double x)
{
        return sin(x) + 2;
}

double Trap(double a, double b, int n)
{
        double h = (b-a)/n;
        int i;

        for (i = 0; i < n; i++) {
                double x_i = a + i*h;
                Size = Size + f(x_i) * h;
        }

        return Size;
}
```



A First OpenMP Version

```
double f(double x)
{
        return sin(x) + 2;
}

double Trap(double a, double b, int n)
{
        double h = (b-a)/n;
        int i;

        for (i = 0; i < n; i++) {
                double x_i = a + i*h;
                Size = Size + f(x_i) * h;
        }

        return Size;
}
```

(a)

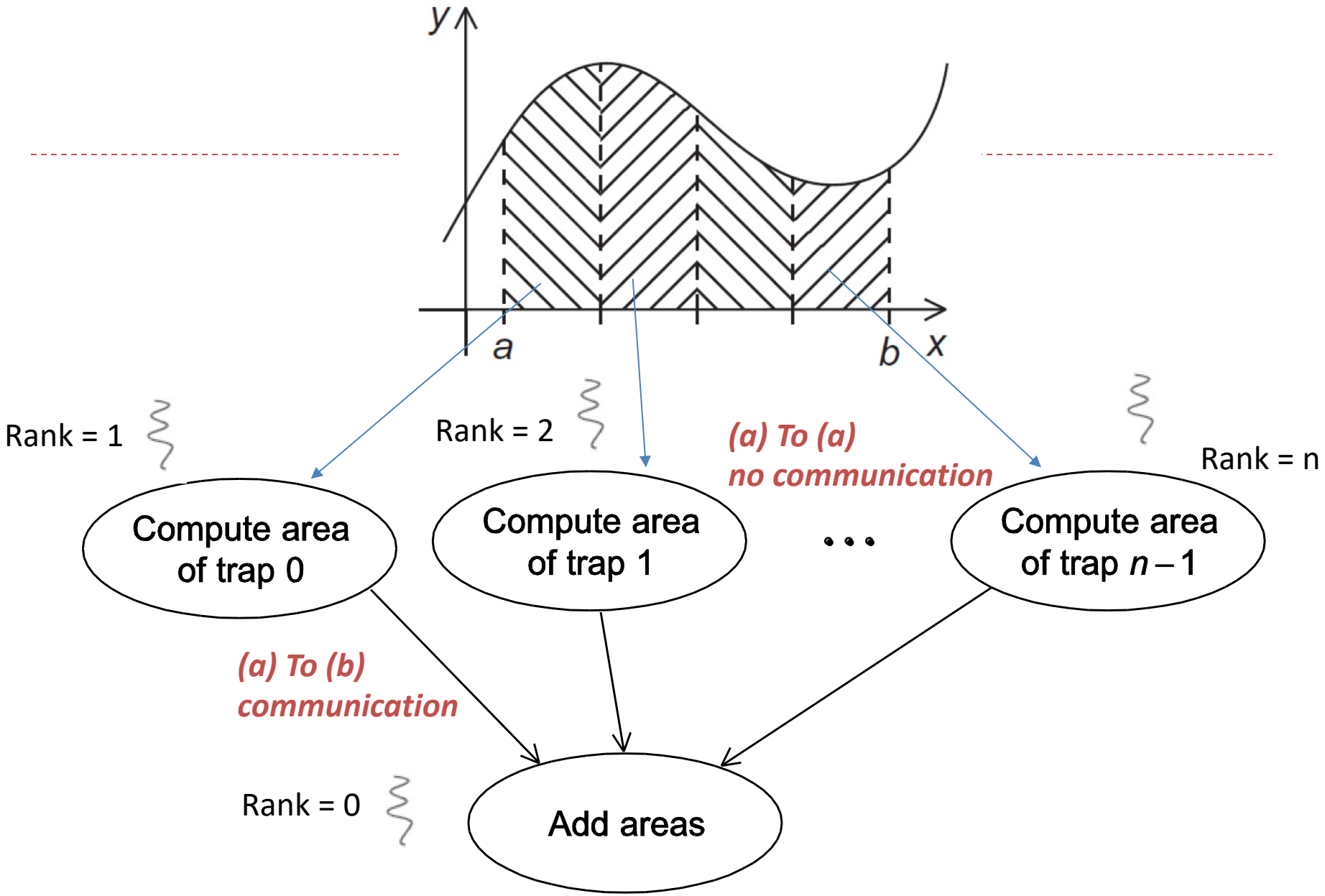
(b)

1) We identified two types of tasks:

   a) computation of the areas of individual trapezoids, and

   b) adding the areas of trapezoids.

2) There is **no communication** among the tasks in the first collection, but each task in the first collection **communicates** with task 1b.

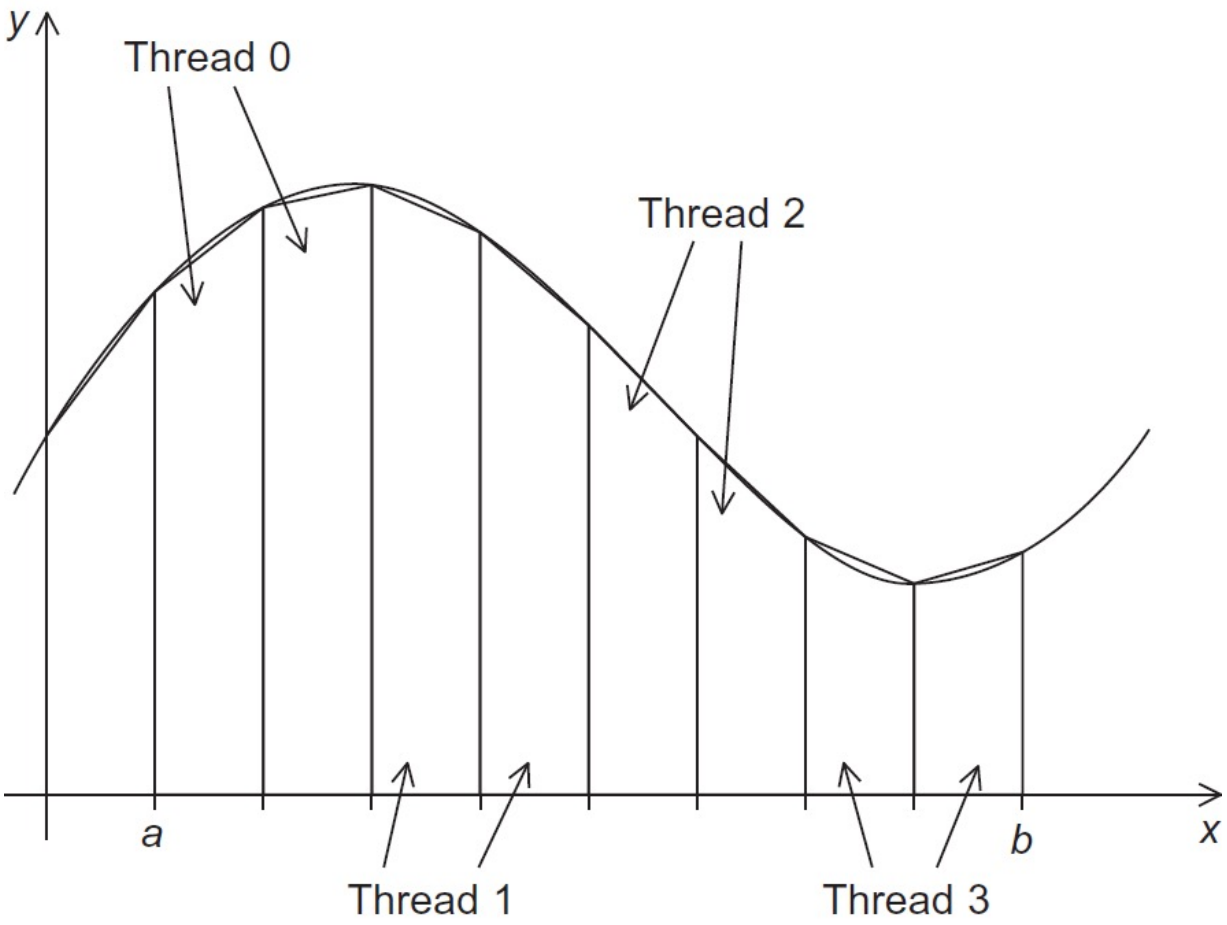

y
a
b
x
Rank = 1
Rank = 2
Rank = n
(a) To (a)
no communication
Compute area
of trap 0
Compute area
of trap 1
• • •
Compute area
of trap n – 1
(a) To (b)
communication
Rank = 0
Add areas

A First OpenMP Version

3) We assumed that there would be many more trapezoids than cores.

- So we aggregated tasks by assigning a contiguous block of trapezoids to each thread (and a single thread to each core).

Assignment of trapezoids to threads



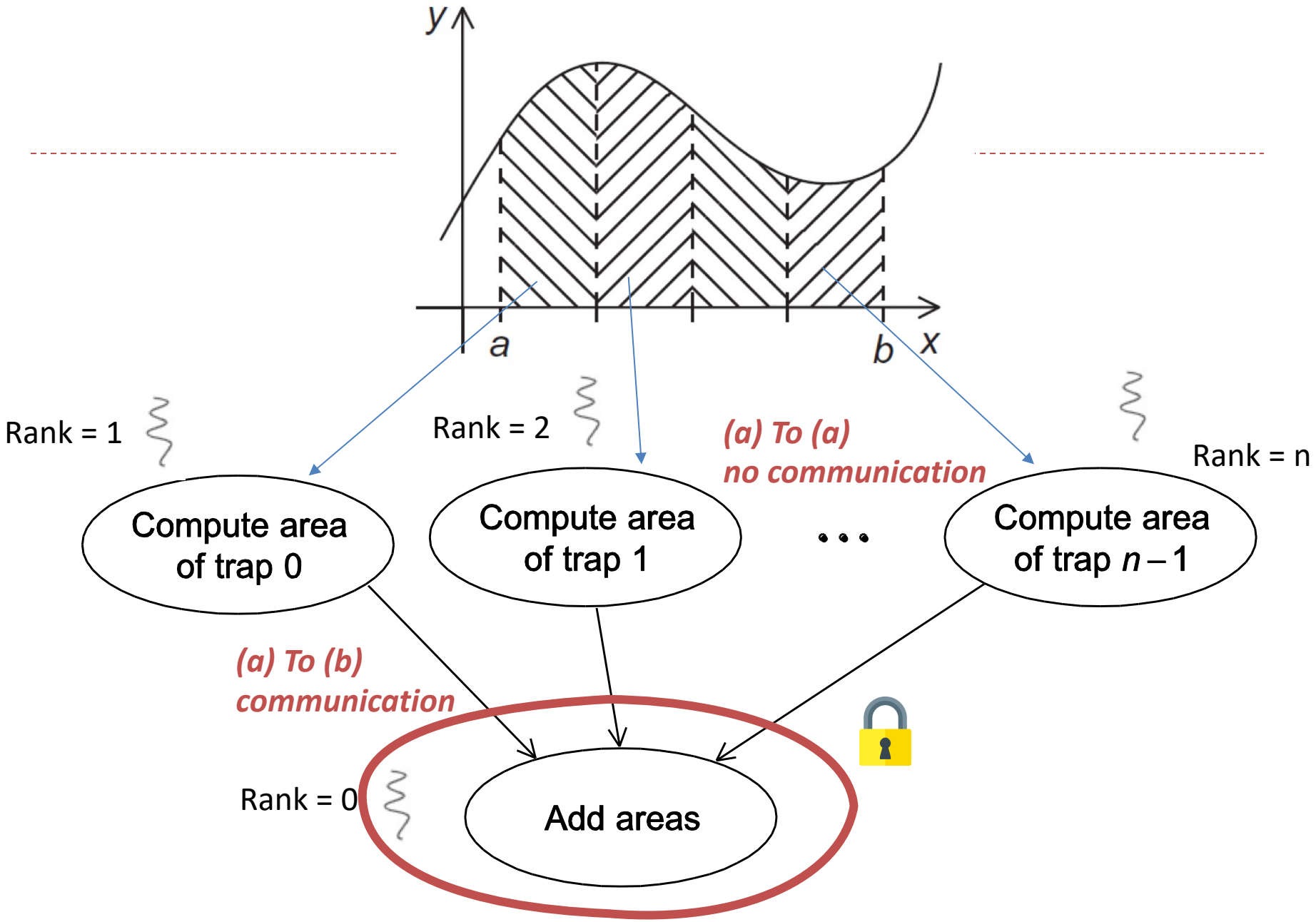
Unpredictable results

```
double f(double x)
{
        return sin(x) + 2;
}

double Trap(double a, double b, int n)
{
        double h = (b-a)/n;
        int i;

        for (i = 0; i < n; i++) {
                double x_i = a + i*h;
                Size = Size + f(x_i) * h;
        }

        return Size;
}
```

```
double f(double x)
{
        return sin(x) + 2;
}

double Trap(double a, double b, int n)
{
        double h = (b-a)/n;
        int i;

        for (i = 0; i < n; i++) {
                double x_i = a + i*h;
                Size = Size + f(x_i) * h;
        }

        return Size;
}
```

- Unpredictable results when two (or more) threads attempt to simultaneously execute:

Size = Size + f(x_i) * h ;


y
a
b
x
Rank = 1
Rank = 2
Rank = n
(a) To (a)
no communication
Compute area
of trap 0
Compute area
of trap 1
• • •
Compute area
of trap n – 1
(a) To (b)
communication
Rank = 0
Add areas

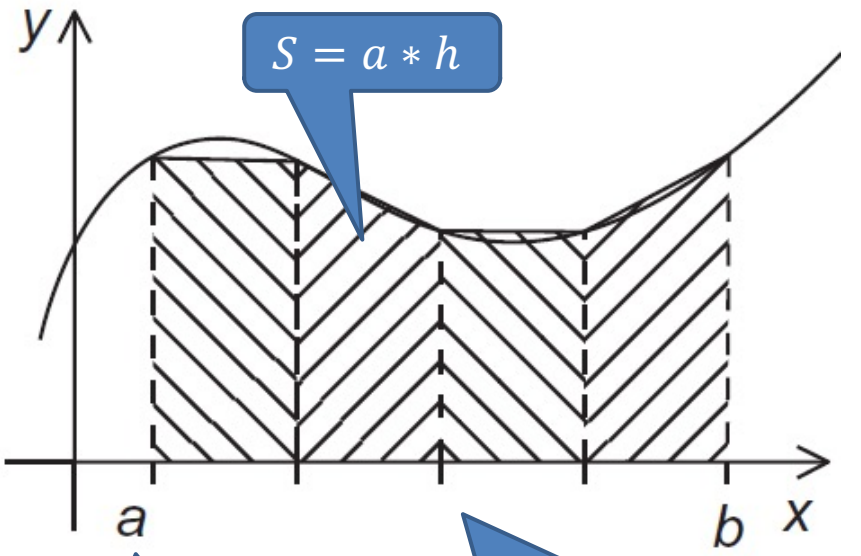
Mutual exclusion

```
# pragma omp critical
   Size = Size + f(x_i) * h ;
```

only one thread can execute
the following structured block at a time

OpenMP Version

https://github.com/kevinsuo/CS7172/blob/master/trapezoidal-omp.c



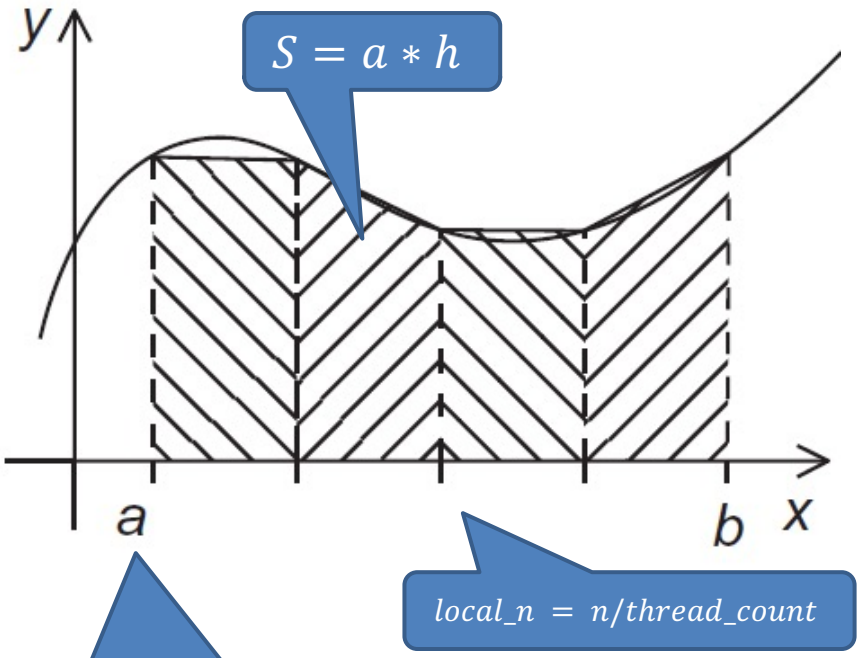

```c
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <unistd.h>
#include <omp.h>
#include <stdlib.h>

double Size;

double f(double x)
{
        return sin(x) + 2;
}

double Trap(double a, double b, int n)
{
        double h = (b-a)/n;
        int i;

        double local_a;
        int local_n;
        double local_Size;

        int my_thread_ID = omp_get_thread_num();
        int thread_count = omp_get_num_threads();

        local_n = n/thread_count;
        local_a = a + (n/thread_count)*h*my_thread_ID;

        for (i = 0; i < local_n; i++) {
                double x_i = local_a + i*h;
                local_Size = local_Size + f(x_i) * h;
        }

#pragma omp critical
        Size = Size + local_Size;

        return Size;
}
```

Task (a)

Task (b)

OpenMP Version

https://github.com/kevinsuo/CS7172/blob/master/trapezoidal-omp.c




```c
int main(int argc, char* argv[]) {
        double a, b, Size;
        int n;
        struct timeval tvs,tve;

        gettimeofday(&tvs,NULL);  //get start time

        a = 1, b = 10;
        n = 1000;

        /* Get number of threads from command line */
        int thread_count = strtol(argv[1], NULL, 10);
#pragma omp parallel num_threads( thread_count )
        Size = Trap(a, b, n);

        printf("Size = %.2lf\n", Size);

        gettimeofday(&tve,NULL);   //get end time
        double span = tve.tv_sec-tvs.tv_sec + (tve.tv_usec-tv
s.tv_usec)/1000000.0;
        printf("Time: %.12f\n", span);

        return 0;
}
```

OpenMP Version

```
ksuo@ksuo-VirtualBox ~/cs7172> ./trapezoidal-serial.o
Size = 19.39
Time: 0.000133000000
ksuo@ksuo-VirtualBox ~/cs7172> ./trapezoidal-omp.o 10
Size = 19.39
Time: 0.000494000000
ksuo@ksuo-VirtualBox ~/cs7172> ./trapezoidal-omp.o 100
Size = 19.39
Time: 0.002283000000
```

- Sometime the overhead of synchronization is larger than the benefit of parallelism

Conclusion

- OpenMP introduction
  - Helloworld of OpenMP
- Performance evaluation
- Example: how to solve problems in OpenMP
  - Trapezoidal problem