

CS 4504

Parallel and Distributed Computation

Message Passing Interface (MPI)

Kun Suo

Computer Science, Kennesaw State University

<https://kevinsuo.github.io/>

Outline

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

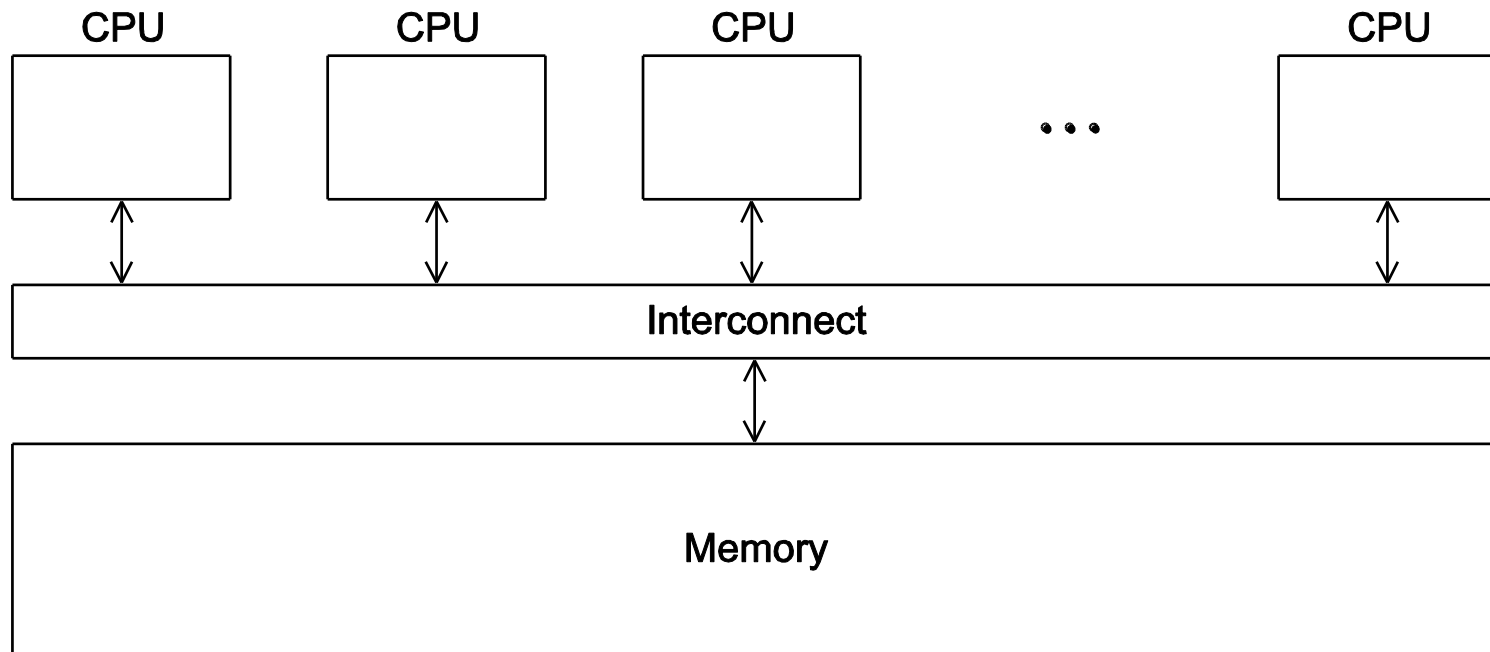


What is MPI?

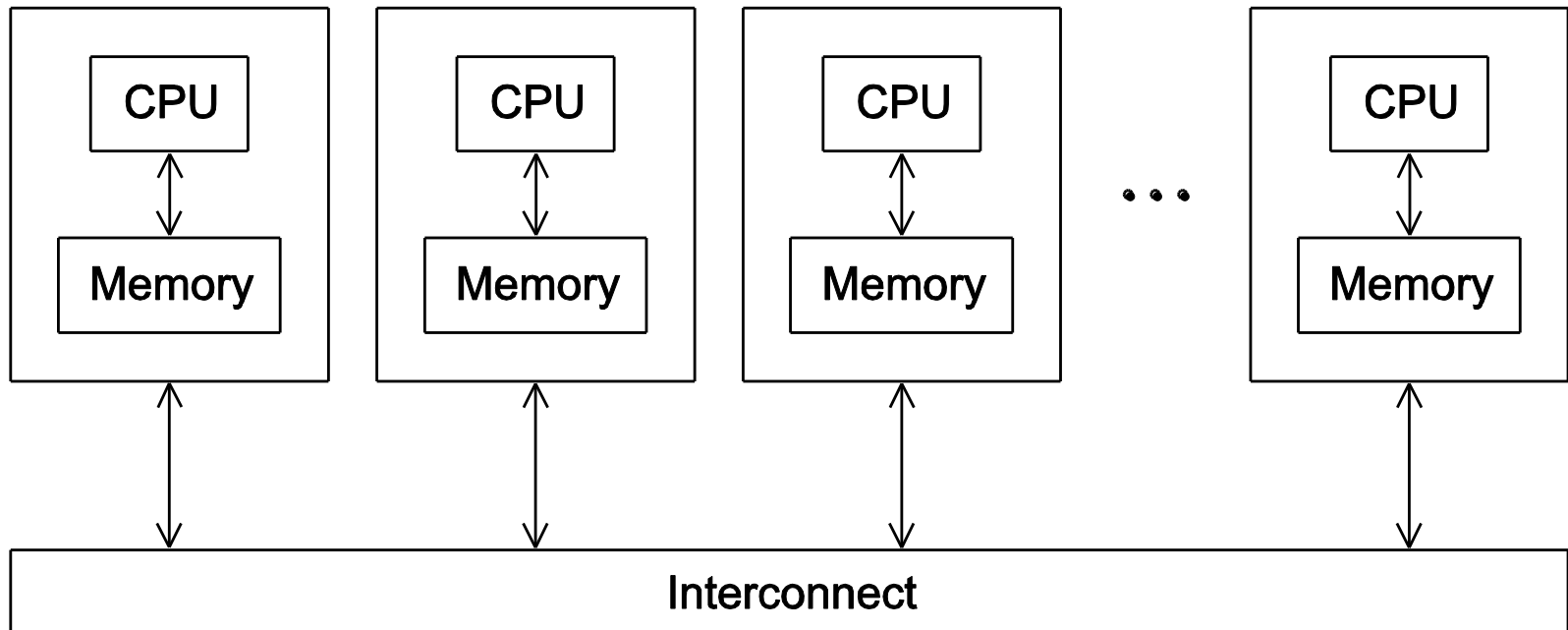


- Message Passing Interface
(MPI) is a standardized library
for parallel computing

A shared memory system



A distributed memory system



Hello World!

```
#include <stdio.h>

int main(void) {
    printf("hello, world\n");

    return 0;
}
```



helloworld-mpi.c

<https://github.com/kevinsuo/CS4504/blob/master/helloworld-mpi.c>

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

MPI Programs

- Written in C.
 - Has main.
 - Uses `stdio.h`, `string.h`, etc.
- Need to add `mpi.h` header file.
- Identifiers defined by MPI start with “MPI_”.
- First letter following underscore is uppercase (e.g., `MPI_Init`).
 - For function names and MPI-defined types.
 - Helps to avoid confusion.

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```



MPI Components

- MPI_Init

- Tells MPI to do all the necessary setup.

```
int MPI_Init(  
    int*      argc_p  /* in/out */,  
    char***   argv_p  /* in/out */);
```

- MPI_Finalize

- Tells MPI we're done, so clean up anything allocated for this program.

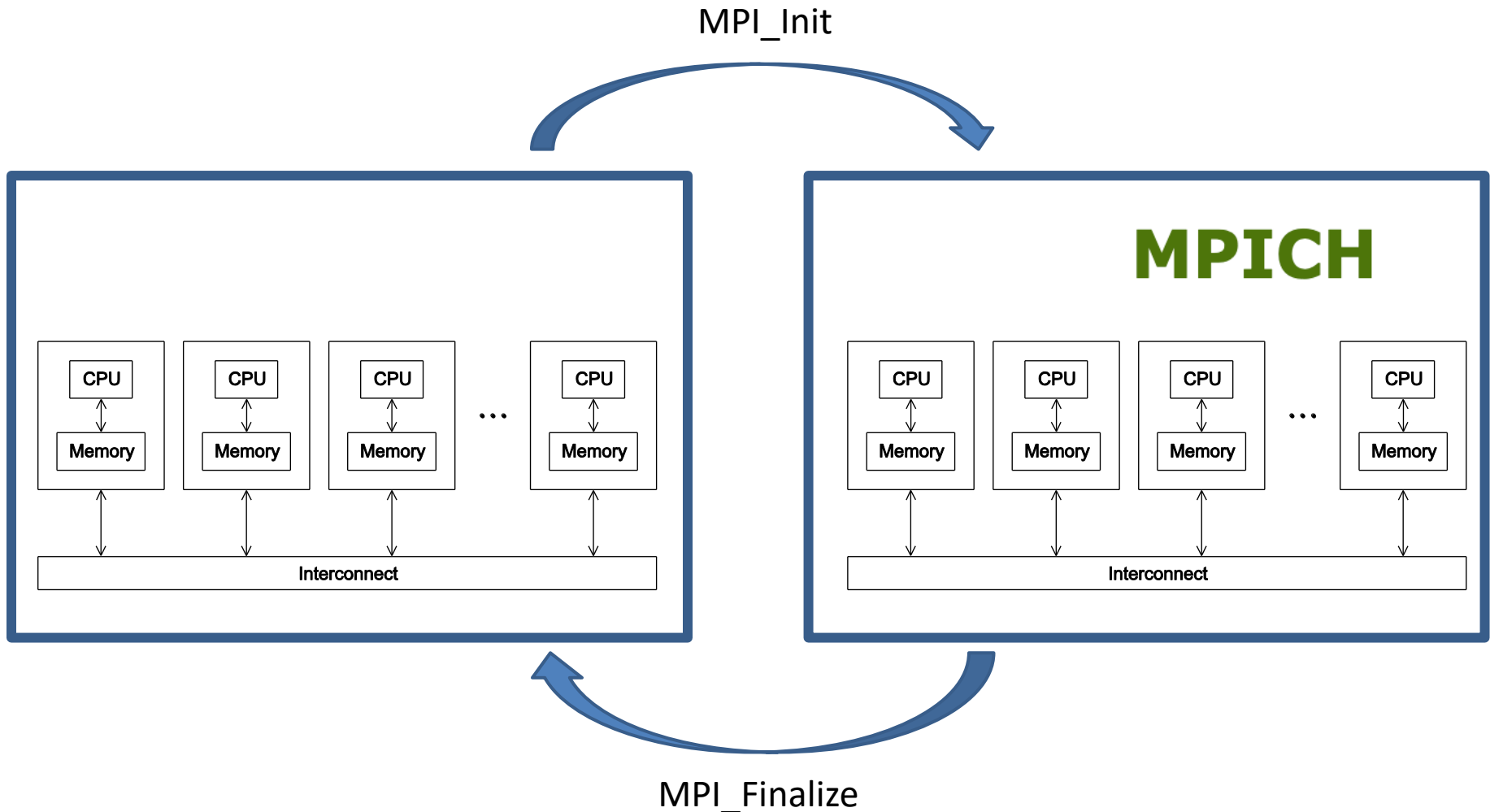
```
int MPI_Finalize(void);
```

Basic Outline

```
. . .  
#include <mpi.h>  
  
. . .  
int main(int argc, char* argv[]) {  
    . . .  
    /* No MPI calls before this */  
    MPI_Init(&argc, &argv);  
    . . .  
    MPI_Finalize();  
    /* No MPI calls after this */  
    . . .  
    return 0;  
}
```



MPI Components



helloworld-mpi.c

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);



    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();

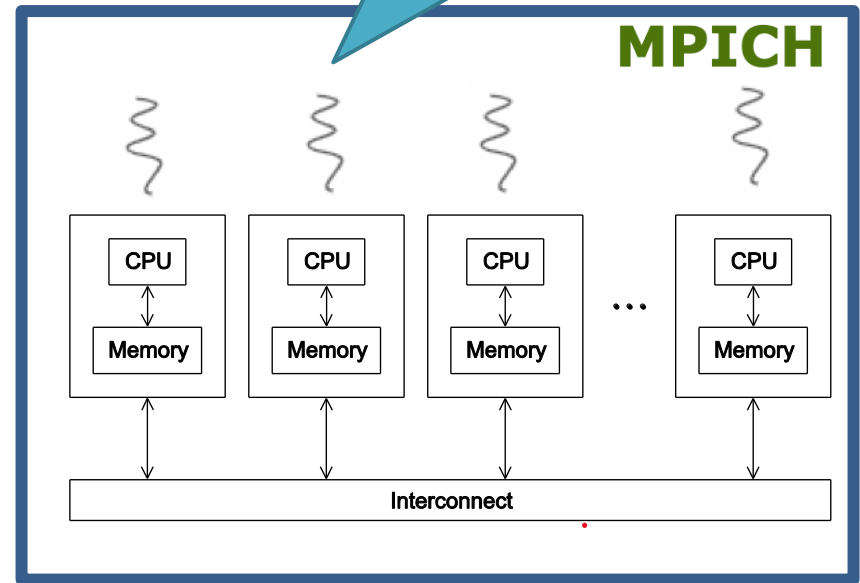
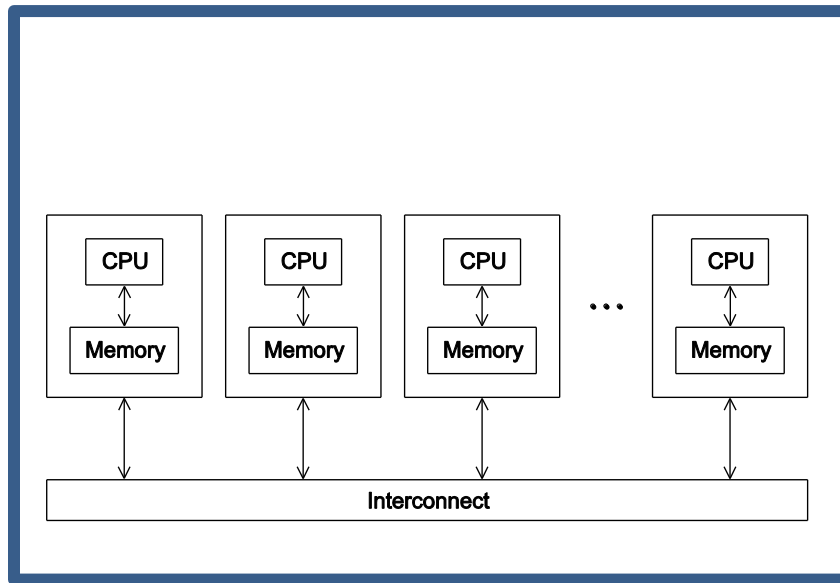
}
```



MPI Components

- 1) how many processes do I have?
- 2) how to differentiate them?
- 3) how they communicate?

MPI_Init



MPI_Finalize

Communicators

- A collection of processes that can send messages to each other.
- MPI_Init defines a communicator that consists of all the processes created when the program is started (defined by user).
- Called **MPI_COMM_WORLD**. → Number of process



helloworld-mpi.c

<https://github.com/kevinsuo/CS4504/blob/master/helloworld-mpi.c>

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);



    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();

}
```



Communicators

```
int MPI_Comm_size(  
    MPI_Comm comm      /* in */,  
    int* comm_sz_p     /* out */);
```



number of processes in the communicator

```
int MPI_Comm_rank(  
    MPI_Comm comm      /* in */,  
    int* my_rank_p     /* out */);
```



my rank
(the process making this call)

helloworld-mpi.c

<https://github.com/kevinsuo/CS4504/blob/main/helloworld-mpi.c>

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

int main(int argc, char** argv) {
    // Initialize the MPI environment ✓
    MPI_Init(NULL, NULL);

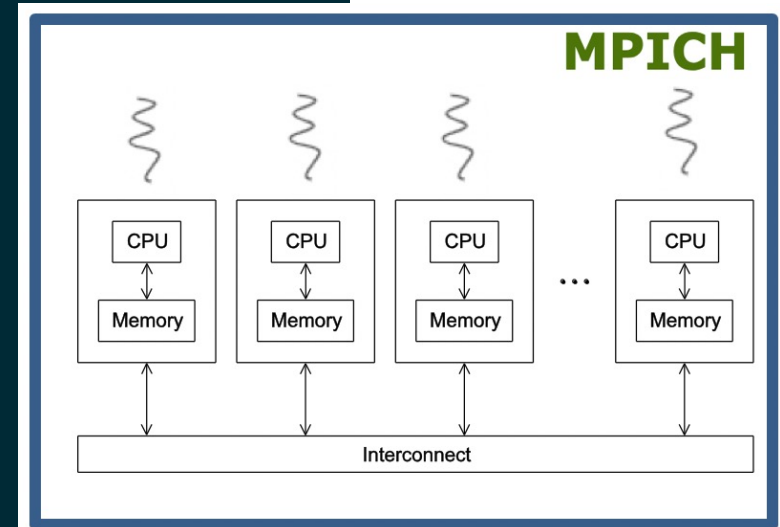
    // Get the number of processes ✓
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process ✓
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

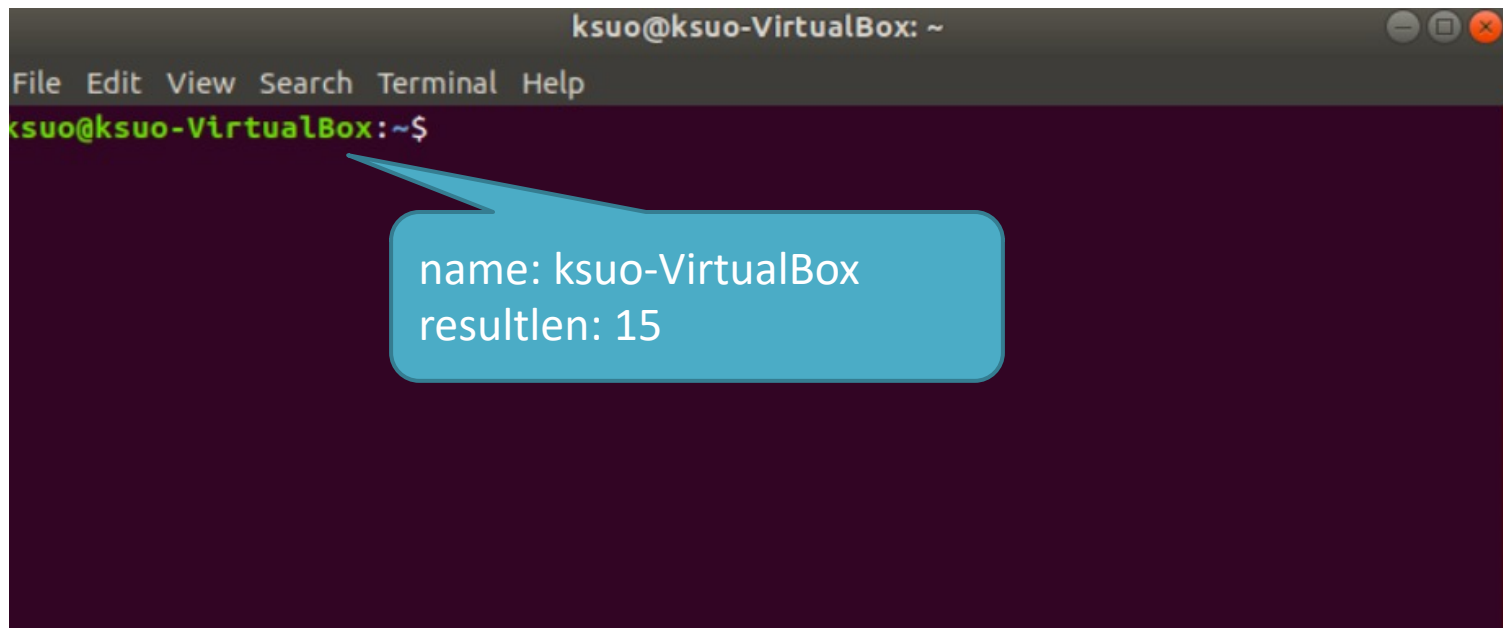
    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment. ✓
    MPI_Finalize();
}
```



MPI_Get_processor_name(char *name,int *resultlen)

- name: your machine name string
- resultlen: your machine name string length



A terminal window titled "ksuo@ksuo-VirtualBox: ~" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "ksuo@ksuo-VirtualBox:~\$". A blue callout box points to the prompt and contains the text "name: ksuo-VirtualBox" and "resultlen: 15".

helloworld-mpi.c

<https://github.com/kevinsuo/CS4504/blob/main/helloworld-mpi.c>

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

int main(int argc, char** argv) {
    // Initialize the MPI environment ✓
    MPI_Init(NULL, NULL);

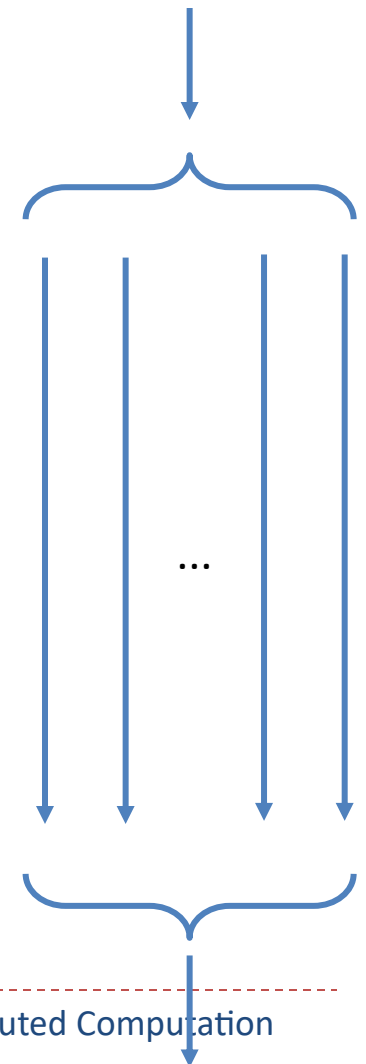
    // Get the number of processes ✓
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process ✓
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor ✓
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment. ✓
    MPI_Finalize();
}
```



Compile

wrapper script to compile

source file

`mpicc -g -Wall -o mpi_hello mpi_hello.c`

produce debugging information

create this executable file name (as opposed to default a.out)

turns on all warnings

```
fish /home/administrator
administrator@ubuntu1804vm ~>
mpicc -g -Wall helloworld-mpi.c -o helloworld-mpi.o

Command 'mpicc' not found, but can be installed with:

sudo apt install lam4-dev
sudo apt install libmpich-dev
sudo apt install libopenmpi-dev
```

Execution

`mpiexec -n <number of processes> <executable>`

`mpiexec -n 1 ./mpi_hello`

 *run with 1 process*

`mpiexec -n 4 ./mpi_hello`

 *run with 4 processes*



Execution

```
mpiexec -n 1 ./mpi_hello
```

Greetings from process 0 of 1 !

```
mpiexec -n 4 ./mpi_hello
```

Greetings from process 0 of 4 !

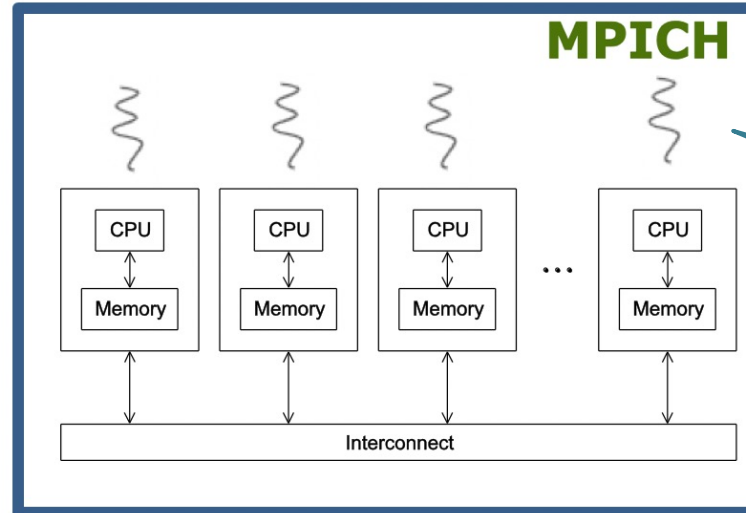
Greetings from process 1 of 4 !

Greetings from process 2 of 4 !

Greetings from process 3 of 4 !



```
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 4 ./helloworld-mpi.o
Hello world from processor ksuo-VirtualBox, rank 0 out of 4 processors
Hello world from processor ksuo-VirtualBox, rank 1 out of 4 processors
Hello world from processor ksuo-VirtualBox, rank 2 out of 4 processors
Hello world from processor ksuo-VirtualBox, rank 3 out of 4 processors
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 2 ./helloworld-mpi.o
Hello world from processor ksuo-VirtualBox, rank 1 out of 2 processors
Hello world from processor ksuo-VirtualBox, rank 0 out of 2 processors
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 1 ./helloworld-mpi.o
Hello world from processor ksuo-VirtualBox, rank 0 out of 1 processors
```



World_size: 4
World_rank: 2

helloworld-mpi.c

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

```
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 4 ./helloworld-mpi.o
Hello world from processor ksuo-VirtualBox, rank 0 out of 4 processors
Hello world from processor ksuo-VirtualBox, rank 1 out of 4 processors
Hello world from processor ksuo-VirtualBox, rank 2 out of 4 processors
Hello world from processor ksuo-VirtualBox, rank 3 out of 4 processors
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 2 ./helloworld-mpi.o
Hello world from processor ksuo-VirtualBox, rank 1 out of 2 processors
Hello world from processor ksuo-VirtualBox, rank 0 out of 2 processors
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 1 ./helloworld-mpi.o
Hello world from processor ksuo-VirtualBox, rank 0 out of 1 processors
```

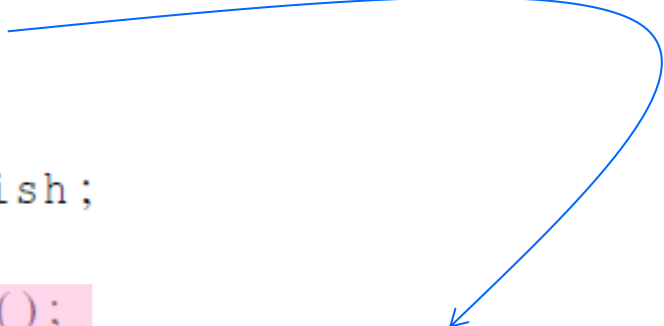

Performance evaluation



Elapsed parallel time

- Returns the number of seconds that have elapsed since some time in the past.

```
double MPI_Wtime(void);  
  
double start, finish;  
. . .  
start = MPI_Wtime();  
/* Code to be timed */  
. . .  
finish = MPI_Wtime();  
printf("Proc %d > Elapsed time = %e seconds\n"  
       my_rank, finish-start);
```



Example

https://raw.githubusercontent.com/kevinsuo/CS4504/master/time_sample.c

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    double t1, t2 = 0;

    MPI_Init( 0, 0 );
    sleep(10);
    printf("MPI_Wtime measured a 1 second sleep to be: %1.2f\n", t2-t1);fflush(stdout);
    MPI_Finalize( );
    return 0;
}
```



Example

https://raw.githubusercontent.com/kevinsuo/CS4504/master/time_sample.c

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    double t1, t2;

    MPI_Init( 0, 0 );
    t1 = MPI_Wtime(); ←
    sleep(10);
    t2 = MPI_Wtime(); ←
    printf("MPI_Wtime measured a 1 second sleep to be: %1.2f\n", t2-t1);fflush(stdout);
    MPI_Finalize( );
    return 0;
}
```



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);
```

Example

https://raw.githubusercontent.com/kevinsuo/C4504/master/time_sample2.c

```
#include "mpi.h"
#include <stdio.h>
#include <sys/time.h>

#define GET_TIME(now) { \
    struct timeval t; \
    gettimeofday(&t, NULL); \
    now = t.tv_sec + t.tv_usec/1000000.0; \
}

int main( int argc, char *argv[] )
{
    double t1, t2 = 0;

    MPI_Init( 0, 0 );
    sleep(10);
    printf("Elapsed time = %e\n", t2-t1);fflush(stdout);
    MPI_Finalize( );
    return 0;
}
```



Example

https://raw.githubusercontent.com/kevinsuo/C4504/master/time_sample2.c

```
#include "mpi.h"
#include <stdio.h>
#include <sys/time.h>

#define GET_TIME(now) { \
    struct timeval t; \
    gettimeofday(&t, NULL); \
    now = t.tv_sec + t.tv_usec/1000000.0; \
}

int main( int argc, char *argv[] )
{
    double t1, t2;

    MPI_Init( 0, 0 );
    GET_TIME(t1); ←
    sleep(10);
    GET_TIME(t2); ←
    printf("Elapsed time = %e\n", t2-t1);fflush(stdout);
    MPI_Finalize( );
    return 0;
}
```


Elapsed serial time in nanoseconds

```
#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);
}
```



Example

https://github.com/kevinsuo/CS4504/blob/master/time_sample3.c

```
#include "mpi.h"
#include <stdio.h>
#include <time.h>

int main( int argc, char *argv[] )
{
    struct timespec t1, t2;

    MPI_Init( 0, 0 );
    sleep(10);

    double diff = 0;

    printf("Time = %llu nanoseconds\n", (long long unsigned int)diff) ; fflush(stdout);
    MPI_Finalize( );
    return 0;
}
```



Example

https://github.com/kevinsuo/CS4504/blob/master/time_sample3.c

```
#include "mpi.h"
#include <stdio.h>
#include <time.h>

int main( int argc, char *argv[] )
{
    struct timespec t1, t2;

    MPI_Init( 0, 0 );
    clock_gettime(CLOCK_MONOTONIC, &t1); ←
    sleep(10);

    clock_gettime(CLOCK_MONOTONIC, &t2); ←

    double diff = 1000000000L * (t2.tv_sec - t1.tv_sec) + t2.tv_nsec - t1.tv_nsec;

    printf("Time = %llu nanoseconds\n", (long long unsigned int)diff) ; fflush(stdout);
    MPI_Finalize( );
    return 0;
}
```



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

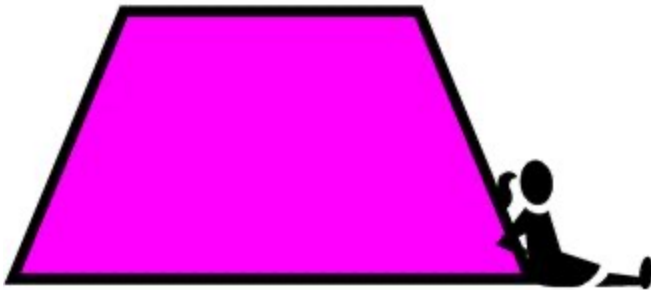
$$S(n, p) = \frac{T_{\text{serial}}(n)}{T_{\text{parallel}}(n, p)}$$

Speedups of Parallel Matrix-Vector Multiplication

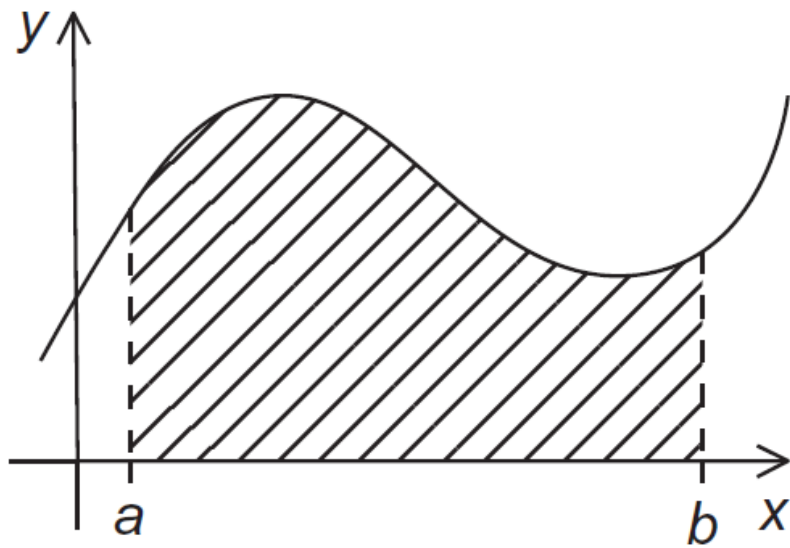
comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.0	1.0	1.0	1.0	1.0
2	1.8	1.9	1.9	1.9	2.0
4	2.1	3.1	3.6	3.9	3.9
8	2.4	4.8	6.5	7.5	7.9
16	2.4	6.2	10.8	14.2	15.5



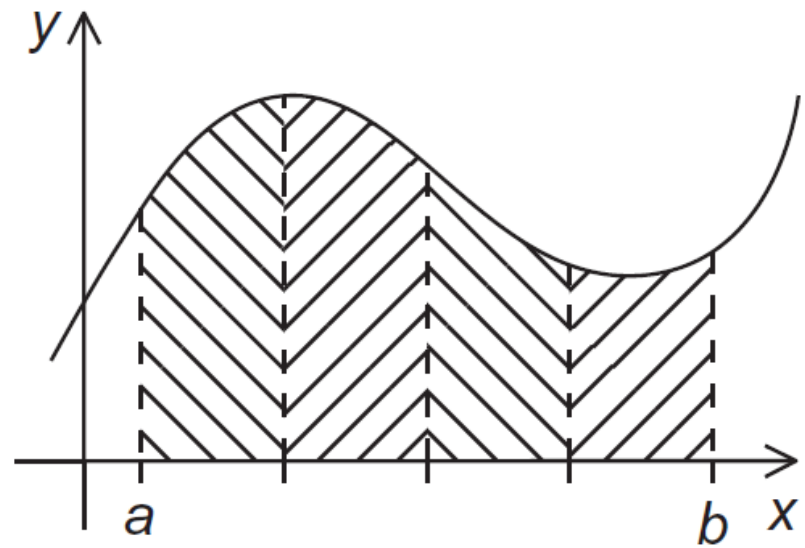
Trapezoidal rule in mpi



The Trapezoidal Rule

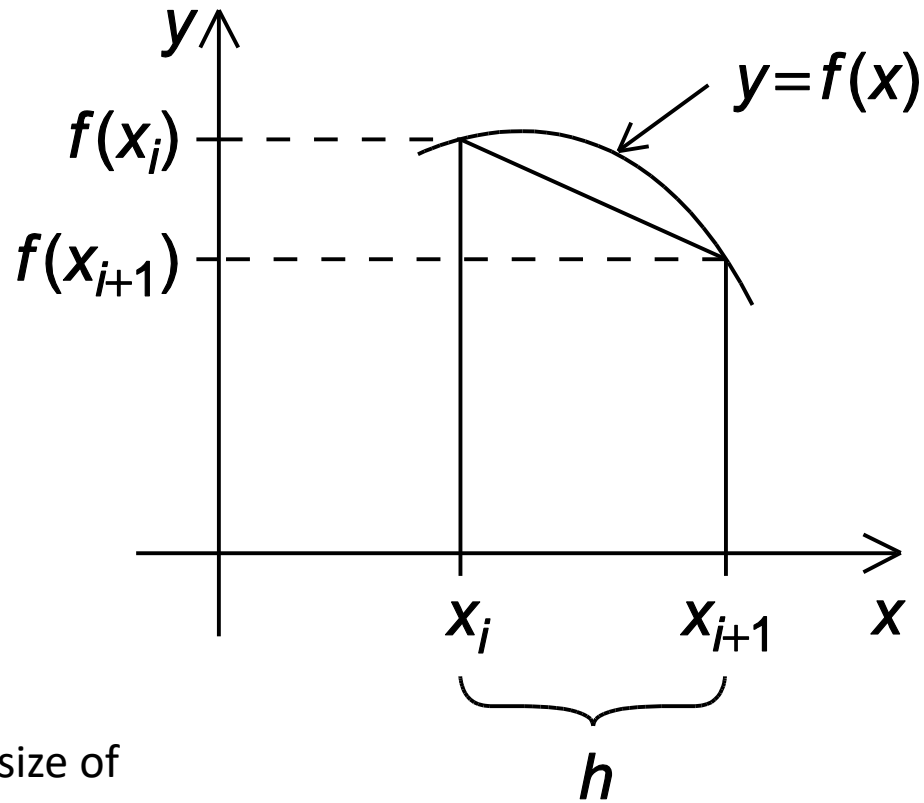


(a)



(b)

One trapezoid



How to get the size of
one trapezoid?

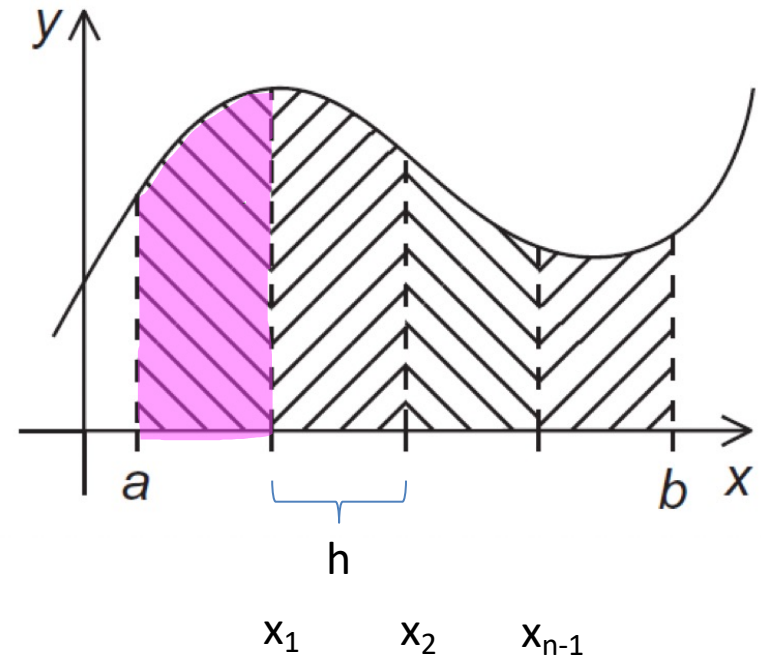
$$\text{Area of one trapezoid} = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$



The Trapezoidal Rule

$$\text{Area of one trapezoid} = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

$$h = \frac{b-a}{n}$$



$$x_0 = a, x_1 = a + h, x_2 = a + 2h, \dots, x_{n-1} = a + (n-1)h, x_n = b$$

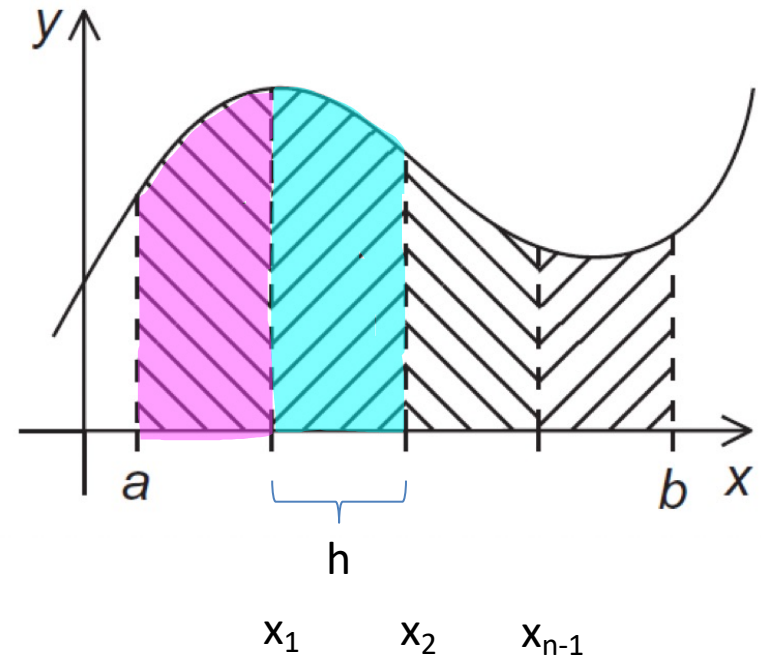
$$\frac{h}{2}[f(a) + f(x_1)] = h \left[\frac{f(a)}{2} + \frac{f(x_1)}{2} \right]$$



The Trapezoidal Rule

$$\text{Area of one trapezoid} = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

$$h = \frac{b-a}{n}$$



$$x_0 = a, x_1 = a + h, x_2 = a + 2h, \dots, x_{n-1} = a + (n-1)h, x_n = b$$

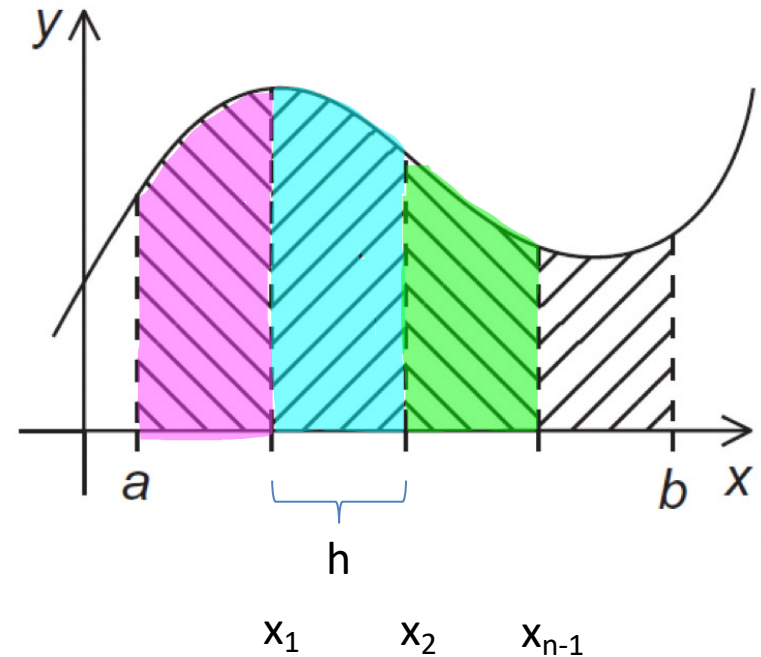
$$S = h \left[\frac{f(a)}{2} + \frac{f(x_1)}{2} + \frac{f(x_1)}{2} + \frac{f(x_2)}{2} \right]$$



The Trapezoidal Rule

$$\text{Area of one trapezoid} = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

$$h = \frac{b-a}{n}$$



$$x_0 = a, x_1 = a + h, x_2 = a + 2h, \dots, x_{n-1} = a + (n-1)h, x_n = b$$

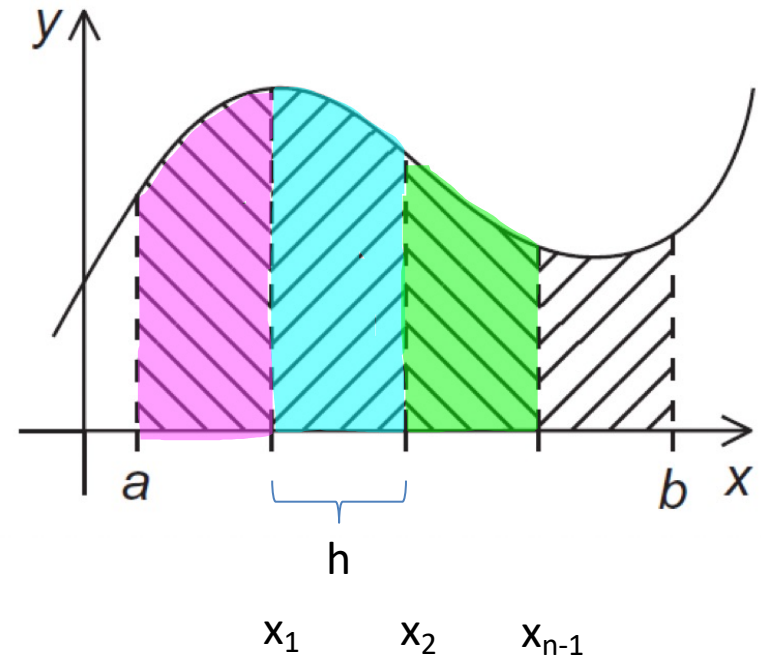
$$S = h \left[\frac{f(a)}{2} + \frac{f(x_1)}{2} + \frac{f(x_1)}{2} + \frac{f(x_2)}{2} + \frac{f(x_2)}{2} + \frac{f(x_3)}{2} \right]$$



The Trapezoidal Rule

$$\text{Area of one trapezoid} = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

$$h = \frac{b - a}{n}$$



$$x_0 = a, x_1 = a + h, x_2 = a + 2h, \dots, x_{n-1} = a + (n-1)h, x_n = b$$

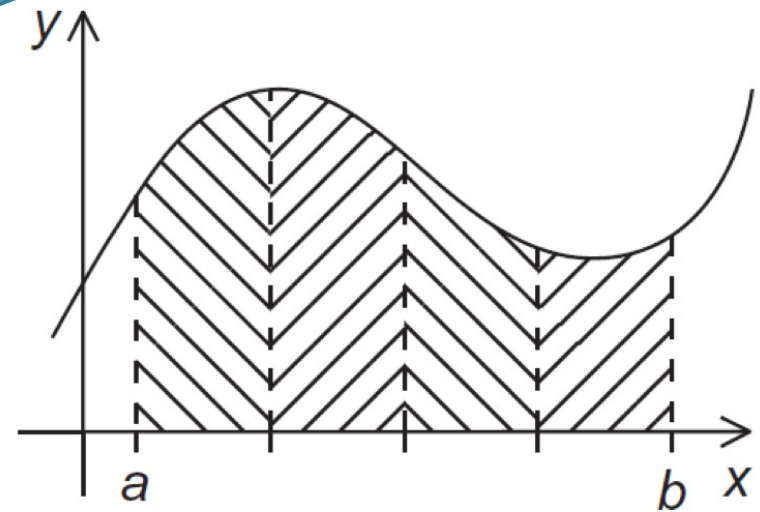
$$\text{Sum of trapezoid areas} = h[f(x_0)/2 + f(x_1) + f(x_2) + \dots + f(x_{n-1}) + f(x_n)/2]$$

Pseudo-code for a serial program

Sum of trapezoid areas $= h[f(x_0)/2 + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + f(x_n)/2]$

```
/* Input:  a, b, n */  
h = (b-a)/n;  
approx = (f(a) + f(b))/2.0;  
for (i = 1; i <= n-1; i++) {  
    x_i = a + i*h;  
    approx += f(x_i);  
}  
approx = h*approx;
```

$$\begin{aligned} & (f(a) + f(b))/2 \\ &= (f(x_0) + f(x_n))/2 \end{aligned}$$



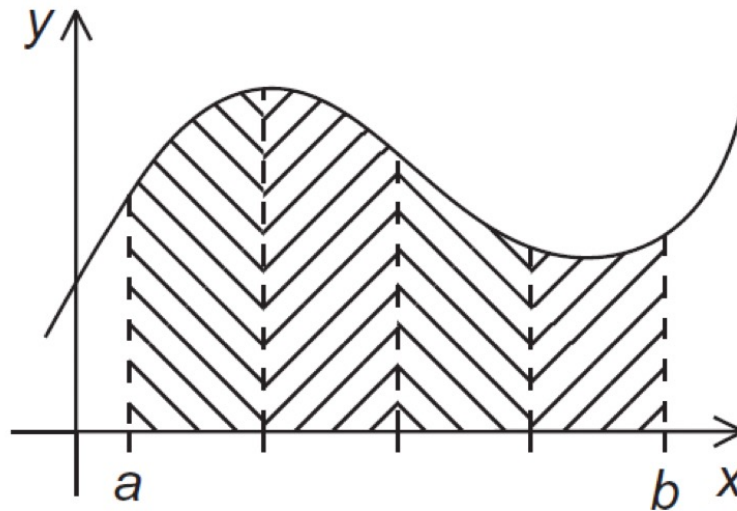
Parallelizing the Trapezoidal Rule

1. Partition problem solution into tasks.
2. Identify communication channels between threads.
3. Aggregate tasks into composite tasks.
4. Map composite tasks to threads/cores.



Parallelizing the Trapezoidal Rule

1. Partition problem solution into tasks.



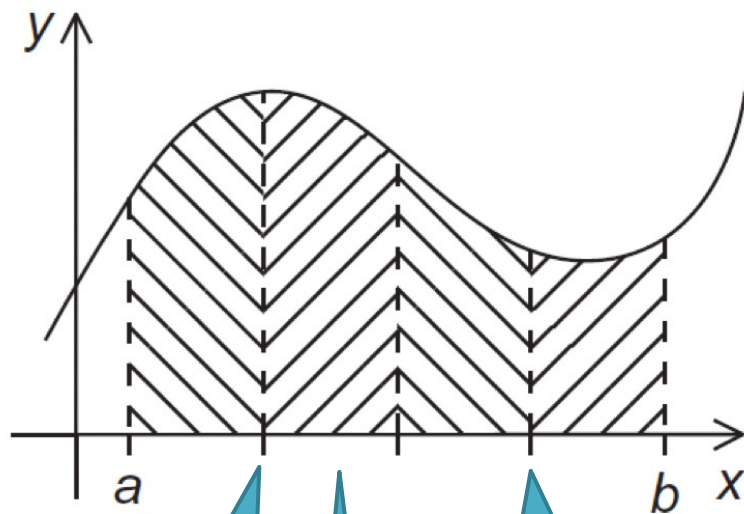
What is the problem?

What is the task?



Parallelizing the Trapezoidal Rule

1. Partition problem solution into tasks.



Left_endpt

Right_endpt

Base_len

Trap_count: how many
trapezoid for each thread

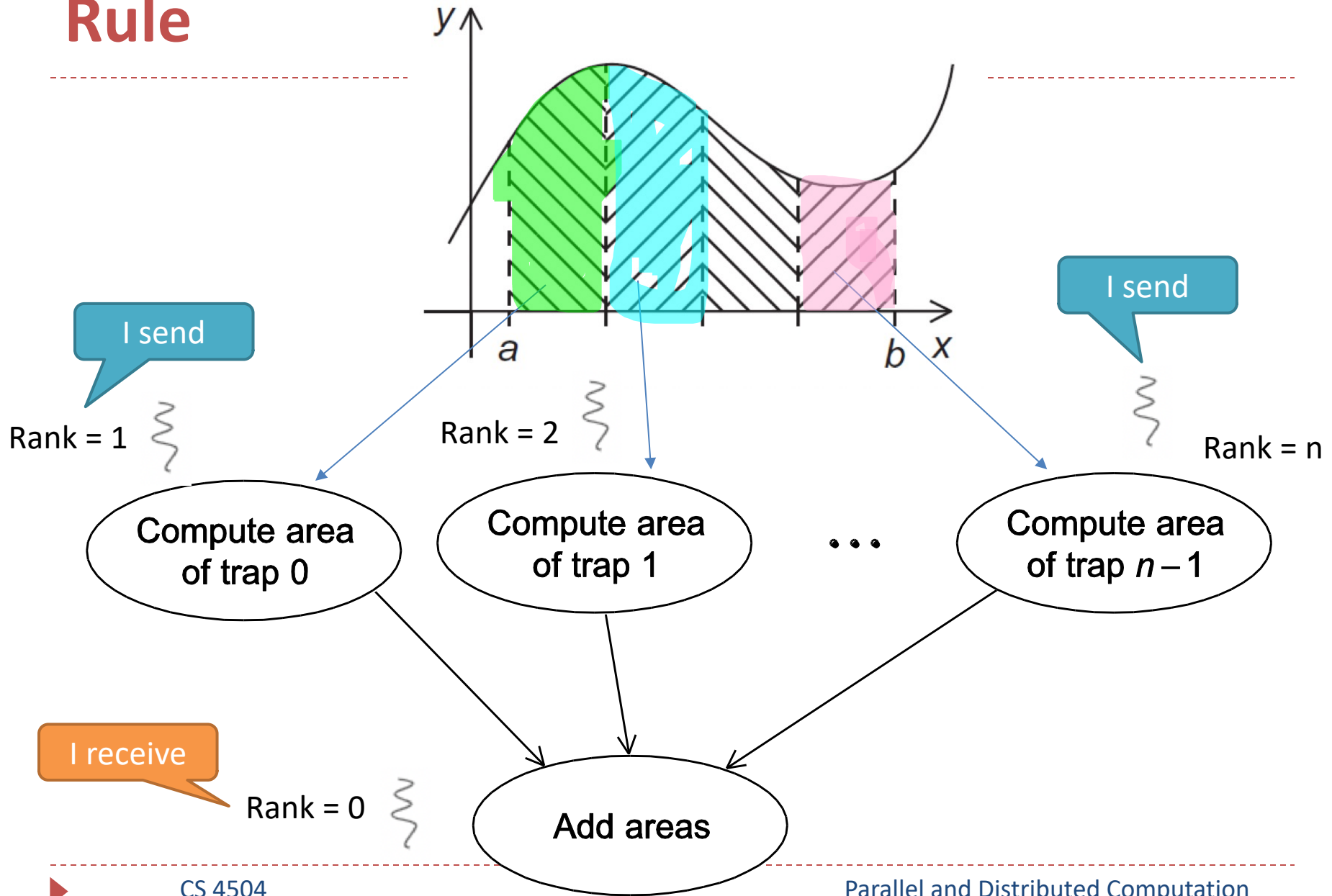
```
double Trap(  
    double left_endpt /* in */,  
    double right_endpt /* in */,  
    int trap_count /* in */,  
    double base_len /* in */) {  
  
    // function logic  
  
    return estimate;  
} /* Trap */
```

Parallelizing the Trapezoidal Rule

- ~~1. Partition problem solution into tasks.~~
2. Identify communication channels between threads.



Tasks and communications for Trapezoidal Rule



Communication

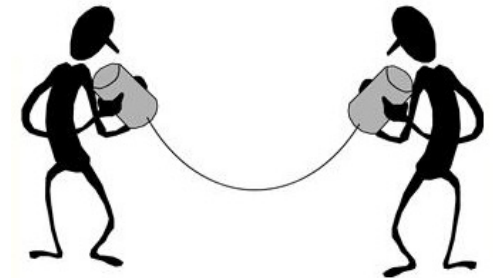
```
int MPI_Send(  
    void*      msg_buf_p    /* in */,  
    int        msg_size     /* in */,  
    MPI_Datatype msg_type    /* in */,  
    int        dest         /* in */,  
    int        tag          /* in */,  
    MPI_Comm   communicator /* in */);
```

Send what?

How many?

Type?

To where?



Data types

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG	signed long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	



Communication

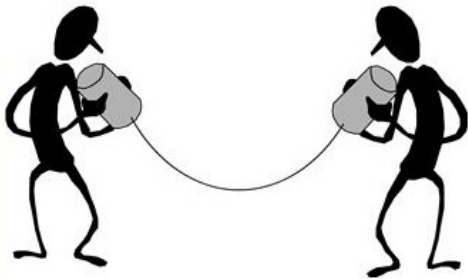
```
int MPI_Recv(  
    void*      msg_buf_p      /* out */,  
    int        buf_size       /* in */,  
    MPI_Datatype buf_type      /* in */,  
    int        source          /* in */,  
    int        tag             /* in */,  
    MPI_Comm   communicator    /* in */,  
    MPI_Status* status_p       /* out */);
```

Recv what?

How many?

Type?

From where?



Message Sent From q to r

```
MPI_Send(send_buf_p, send_buf_sz, send_type, dest, send_tag,  
         send_comm);
```

r

MPI_Send

src = q



MPI_Recv

dest = r

```
MPI_Recv(recv_buf_p, recv_buf_sz, recv_type, src, recv_tag,  
         recv_comm, &status);
```

q

Parallelizing the Trapezoidal Rule

- ~~1. Partition problem solution into tasks.~~
- ~~2. Identify communication channels between threads.~~
3. Aggregate tasks into composite tasks.
4. Map composite tasks to threads/cores.



Example: 1/2/3 send to 0 and 0 prints out messages

```
#include <mpi.h>
#include <stdio.h>
#include <string.h>

const int MAX_STRING = 100;

int main(int argc, char** argv) {
    char greeting[MAX_STRING];
    int comm_sz; //number of processes
    int my_rank; //my process rank

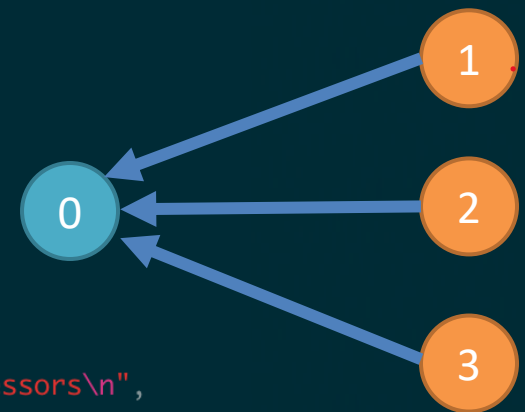
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    if (my_rank != 0) {
        sprintf(greeting, "Greetings from processor %s, rank %d out of %d processors\n",
            processor_name, my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    } else {
        printf("This is process: %d, Greetings from processor %s\n",
            my_rank, processor_name);
        for (int q = 1; q < comm_sz; q++) {
            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s", greeting);
        }
    }

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

```
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 4 ./helloworld-mpi.o
This is process: 0, Greetings from processor ksuo-VirtualBox
Greetings from processor ksuo-VirtualBox, rank 1 out of 4 processors
Greetings from processor ksuo-VirtualBox, rank 2 out of 4 processors
Greetings from processor ksuo-VirtualBox, rank 3 out of 4 processors
```



Send to 0 from 1,...,comm_z-1

Sample code:

```
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 4 ./helloworld-mpi.o
This is process: 0, Greetings from processor ksuo-VirtualBox
Greetings from processor ksuo-VirtualBox, rank 1 out of 4 processors
Greetings from processor ksuo-VirtualBox, rank 2 out of 4 processors
Greetings from processor ksuo-VirtualBox, rank 3 out of 4 processors
```

<https://github.com/kevinsuo/CS4504/blob/master/helloworld-mpi-sample.c>

```
#include <mpi.h>
#include <stdio.h>
#include <string.h>
```

```
const int MAX_STRING = 100;
```

```
int main(int argc, char** argv) {
    char greeting[MAX_STRING];
    int comm_sz; //number of processes
    int my_rank; //my process rank
```

```
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

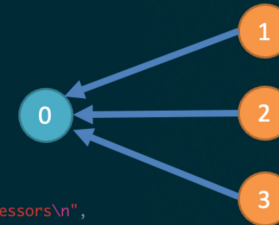
```
    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);
```

```
    if (my_rank != 0) {
        sprintf(greeting, "Greetings from processor %s, rank %d out of %d processors\n",
            processor_name, my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    } else {
        printf("This is process: %d, Greetings from processor %s\n",
            my_rank, processor_name);
        for (int q = 1; q < comm_sz; q++) {
            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s", greeting);
        }
    }
```

```
    // Finalize the MPI environment.
    MPI_Finalize();
}
```

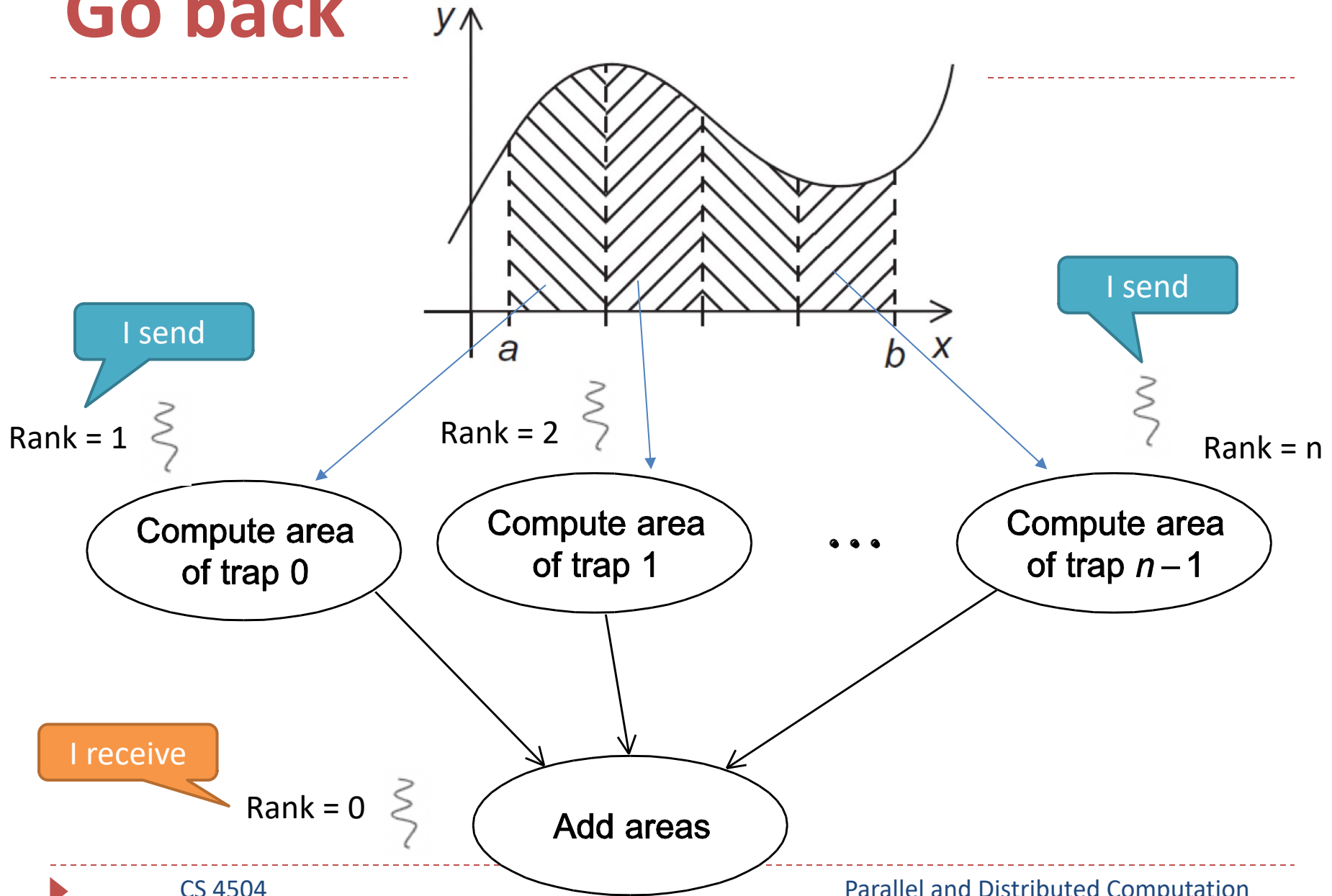
Example: 1/2/3 send to 0 and 0 prints out messages

```
ksuo@ksuo-VirtualBox ~/cs7172> mpiexec -n 4 ./helloworld-mpi.o
This is process: 0, Greetings from processor ksuo-VirtualBox
Greetings from processor ksuo-VirtualBox, rank 1 out of 4 processors
Greetings from processor ksuo-VirtualBox, rank 2 out of 4 processors
Greetings from processor ksuo-VirtualBox, rank 3 out of 4 processors
```



Send to 0 from 1,...,comm_z-1

Go back



Parallel pseudo-code

```
1   Get a, b, n;
2   h = (b-a)/n;
3   local_n = n/comm_sz;
4   local_a = a + my_rank*local_n*h;
5   local_b = local_a + local_n*h;
6   local_integral = Trap(local_a, local_b, local_n, h);
7   if (my_rank != 0)
8       Send local_integral to process 0;
9   else /* my_rank == 0 */
10       total_integral = local_integral;
11       for (proc = 1; proc < comm_sz; proc++) {
12           Receive local_integral from proc;
13           total_integral += local_integral;
14       }
15   }
16   if (my_rank == 0)
17       print result;
```

Task

First version (1)

```
1 double Trap(  
2     double left_endpt /* in */,  
3     double right_endpt /* in */,  
4     int trap_count /* in */,  
5     double base_len /* in */) {  
6     double estimate, x;  
7     int i;  
8  
9     estimate = (f(left_endpt) + f(right_endpt))/2.0;  
10    for (i = 1; i <= trap_count-1; i++) {  
11        x = left_endpt + i*base_len;  
12        estimate += f(x);  
13    }  
14    estimate = estimate*base_len;  
15  
16    return estimate;  
17 } /* Trap */
```

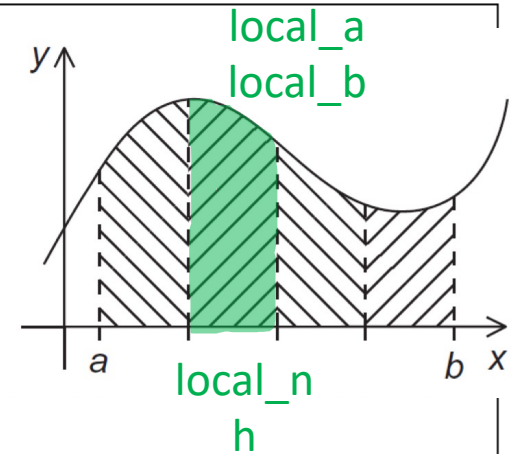


First version (2)

Please read the code by yourself
based on the pseudo-code

```
1  int main(void) {
2      int my_rank, comm_sz, n = 1024, local_n;
3      double a = 0.0, b = 3.0, h, local_a, local_b;
4      double local_int, total_int;
5      int source;
6
7      MPI_Init(NULL, NULL);
8      MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
9      MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
10
11     h = (b-a)/n;          /* h is the same for all processes */
12     local_n = n/comm_sz; /* So is the number of trapezoids */
13
14     local_a = a + my_rank*local_n*h;
15     local_b = local_a + local_n*h;
16     local_int = Trap(local_a, local_b, local_n, h);
17
18     if (my_rank != 0) {
19         MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
20                 MPI_COMM_WORLD);

```



First version (3)

```
21     } else {
22         total_int = local_int;
23         for (source = 1; source < comm_sz; source++) {
24             MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
25                     MPI_COMM_WORLD, MPI_STATUS_IGNORE);
26             total_int += local_int;
27         }
28     }
29
30     if (my_rank == 0) {
31         printf("With n = %d trapezoids, our estimate\n", n);
32         printf("of the integral from %f to %f = %.15e\n",
33               a, b, total_int);
34     }
35     MPI_Finalize();
36     return 0;
37 } /*  main  */
```



Conclusion

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



Practice 1

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processors\n",
           processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Thread 0:
print out "000000"

Thread 1 & 2:
print out "121212"

Thread 3:
print out "333333"



Practice 2

<https://github.com/kevinsuo/CS4504/blob/master/helloworld-mpi-sample.c>

```
#include <mpi.h>
#include <stdio.h>
#include <string.h>

const int MAX_STRING = 100;

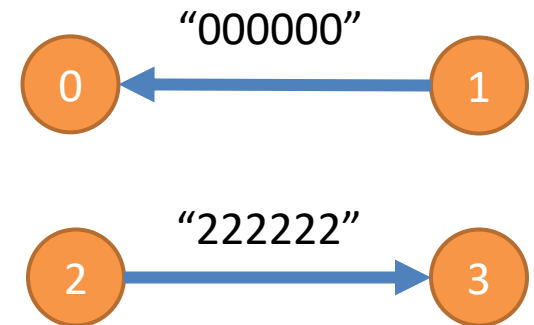
int main(int argc, char** argv) {
    char greeting[MAX_STRING];
    int comm_sz; //number of processes
    int my_rank; //my process rank

    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    if (my_rank != 0) {
        sprintf(greeting, "Greetings from processor %s, rank %d out of %d processors\n",
            processor_name, my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    } else {
        printf("This is process: %d, Greetings from processor %s\n",
            my_rank, processor_name);
        for (int q = 1; q < comm_sz; q++) {
            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s", greeting);
        }
    }

    // Finalize the MPI environment.
    MPI_Finalize();
}
```



Practice 3

<https://github.com/kevinsuo/CS4504/blob/master/helloworld-mpi-sample.c>

```
#include <mpi.h>
#include <stdio.h>
#include <string.h>

const int MAX_STRING = 100;

int main(int argc, char** argv) {
    char greeting[MAX_STRING];
    int comm_sz; //number of processes
    int my_rank; //my process rank

    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    if (my_rank != 0) {
        sprintf(greeting, "Greetings from processor %s, rank %d out of %d processors\n",
            processor_name, my_rank, comm_sz);
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
    } else {
        printf("This is process: %d, Greetings from processor %s\n",
            my_rank, processor_name);
        for (int q = 1; q < comm_sz; q++) {
            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s", greeting);
        }
    }
    // Finalize the MPI environment.
    MPI_Finalize();
}
```

“465”



“1+2+...+10”



“11+12+...+20”



“21+22+...+30”