

## Parallel Numerics

### Exercise 2: Numerical Integration & P2P Communication I

## 1 Numerical Integration

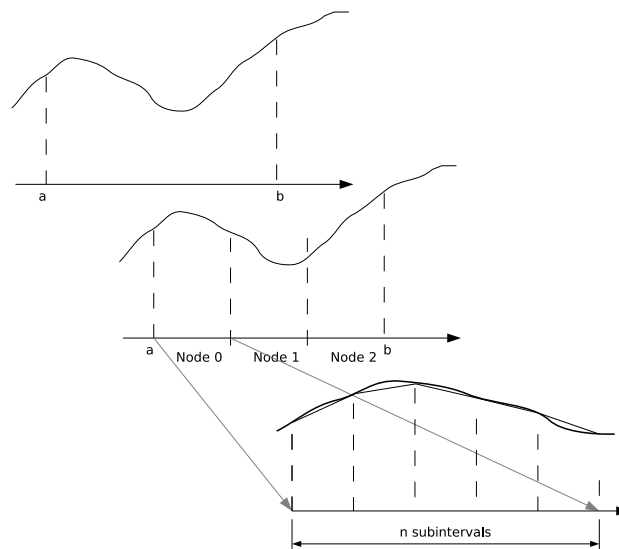


Figure 1: Subdivision of  $[a, b]$  into  $\#proc$  subintervals and subdivision of  $[a_k, b_k]$  into  $n$  subsubintervals.

We are to calculate the definite integral of a function  $f(x)$  in an interval  $[a, b]$  using the trapezoidal rule. Before, we split up the overall interval  $[a, b]$  into  $\#proc$  equidistant subintervals  $[a_k, b_k]$ ,  $k \in \{0, 1, \dots, \#proc - 1\}$ .  $\#proc$  is the number of nodes available on a parallel computer. Afterwards, we divide the interval  $[a_k, b_k]$  into  $n$  intervals with length  $h = \frac{b_k - a_k}{n}$  and approximate the integral by a sum of trapezoids:

$$\begin{aligned}
 \int_{a_k}^{b_k} f(x) dx &\approx \sum_{i=1}^n \frac{h}{2} (f(a_k + (i-1)h) + f(a_k + ih)) \\
 &= \frac{h}{2} (f(a_k) + f(b_k)) + h \cdot \sum_{i=1}^{n-1} f(a_k + ih)
 \end{aligned} \tag{1}$$

- a) Implement the algorithm using the MPI send and receive operations.  
 See sourcecode to corresponding tutorial on webpage.
- b) Test the application for  $f(x) = -3x^2 + 3$ ,  $[a, b] = [-1, 1]$ ,  $n = 2, 4, 8$ , and different numbers of processors.  
 Test on your own with sourcecode from a).
- c) Test the application for  $f(x) = x \sin(10x)$ ,  $[a, b] = [0, 2\pi]$ ,  $n = 2, 4, 8$  and different numbers of processors. Compute the analytical solution and compare it to the numerical

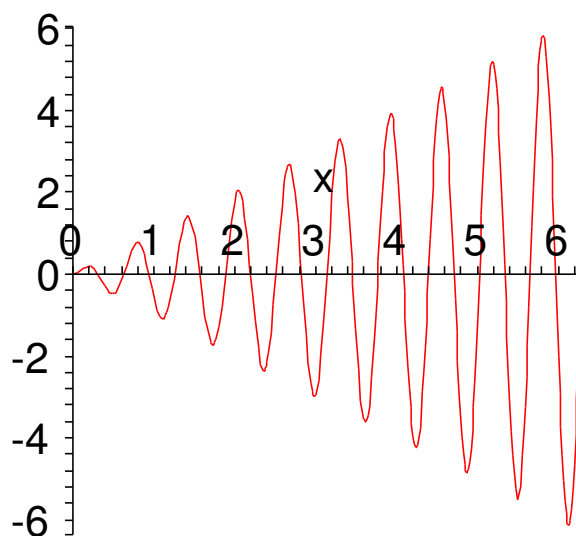


Figure 2:  $f(x) = x \sin(10x)$ .

one. What do you observe and why?

If function is not smooth a sufficient small mesh size  $h$  is required to reduce the computed numerical error. The numerical error is not a parallel phenomenon but relies more on where the boundaries of the subintervals (number of processes) were chosen. One possible remedy is adaptivity.

- d) Implement the following strategy for determining  $n$  on a node:

```

n = 1
Compute  $F_{numerical}^{(n)}$  (see formula (1))
do {
    n++
    Compute  $F_{numerical}^{(n)}$  (see formula (1))
} while (  $F_{numerical}^{(n)} - F_{numerical}^{(n-1)} \geq \varepsilon$  )

```

Test the algorithm for

$$f(x) = \begin{cases} 0 & \text{for } x \leq \frac{1}{2} \\ 1 & \text{for } x > \frac{1}{2} \end{cases}, \quad (2)$$

$[a, b] = [0, 1]$ ,  $n = 2, 3, 4$  and different values for  $\varepsilon$ . Try to find out, how many iterations per node are performed. Is there a bottleneck?

Test on your own. For non-smooth functions the mesh size has to be sufficient small. This trivial domain splitting is overkill for functions with only few non-smooth regions like for instance a tight gaussian peak. Disadvantage: adaptive schemes make load balancing challenging.

- e) Before the application terminates, each node sends its contribution to the numerical result to node 0. You have used MPI send and receive to implement this behaviour. Is there a technical term for such a communication scheme?

Reduce/Reduction.

## 2 P2P Communication I: MPI\_Send and MPI\_Recv

- a) Write an operation sequence that receives four integers from arbitrary senders (use the constant `MPI_ANY_SOURCE` as source number for the `MPI_Recv` operation) using the message tag 0 for each of them. Store the results in four integers `b0`, `b1`, `b2`, and `b3`.

```
int b0,b1,b2,b3;
MPI_Recv(&b0, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD);
MPI_Recv(&b1, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD);
MPI_Recv(&b2, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD);
MPI_Recv(&b3, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD);
```

- b) Consider the following code snippet

```
1   int a = 0;
2   MPI_Send(&a, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);
3   a = 1;
4   MPI_Send(&a, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);
```

running on node 0, and

```
1   int a = 2;
2   MPI_Send(&a, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);
3   a = 3;
4   MPI_Send(&a, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);
```

running on node 1.

What are possible values for `b0`, `b1`, `b2`, and `b3` when executing the four `MPI_Recv` operations from a) on node 2? Is there a unique solution? Are messages allowed to overtake?

The arrival pattern is not deterministic if `MPI_ANY_SOURCE` is used. One receive stores exactly one message in a variable. Messages from the same sender are not allowed to overtake. Thus, the following possibilities on node 2 are:

<code>b0</code>	0	0	0	2	2	2
<code>b1</code>	1	2	2	0	0	3
<code>b2</code>	2	1	3	1	3	0
<code>b3</code>	3	3	1	3	1	1

c) Define the term *blocking* for MPI commands.

Operation is left if variable content (buffer) is set/send completely and can be reused again.

d) Define the term *not synchronized* for MPI commands.

The standard MPI operations are not synchronized, i.e., there is no handshake semantics and it cannot be assured that the message is going to be received.

### 3 Predefined MPI datatypes

For every MPI send or receive command, one has to specify a datatype. Make yourself familiar with the datatypes offered by the MPI interface. To what C datatypes do they correspond?

MPI	C (32 bit)
CHAR	signed char
SHORT	signed short int
INT	signed int
LONG	signed long int
FLOAT	float
DOUBLE	double