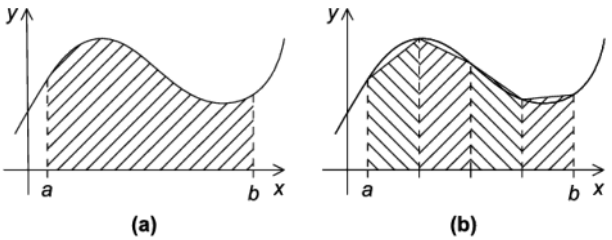
The trapezoidal rule in MPI

Just writing programs that print messages isnt really useful, instead lets let's write a program that implements the trapezoidal rule for numerical integration.



Note:
- Finding the area under the curve can be done by creating trapizoids from a to b where each trapizoid is a subsection of the curve.
  - The idea is the more subsections (traps) the more accurate area under the curve

- Each subinterval is a trapizoid, whose base is the subinterval, whose vertical sides are the vertical lines through the end points of the subinterval, and whose fourth side is the secant line joining the points where the vertical lines cross the graph.
  - If the endpoints of the subinterval are xi and xi+1, then the length of the subinterval is $h = x_{i+1} - x_i$.
  - Also, if the lengths of the two vertical segments are f(xi) and f(xi+1), then the area of the trapezoid is

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

- Since we chose the length of n subintervals they will all have the same length thus we derive that
  - x is a endpoint where x =a and x= b, where a and b are the leftmost endpoint and rightmost endpoint
    - Thus we can say that the length of each subinterval can be

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

  - We can also see that from x_0 to x_n where x is an endpoint, we have that

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

    Where x_0 will be the first endpoint then the next one will be found by taking the last endpoint and adding h to it and so on
- Then we can also say that the sum of each area of trapezoids will be the approx total area under the curve by

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

  - $f(x_0)/2$ and $f(x_n)/2$ have the division by 2 to make it even

- Pseudocode for a serial program of trapezoidal rule would look like:

```
/* 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;
```

Note:
- Find length of each interval first
- Then approx the first and last subinterval
- Then sum up in approx all the other subintervals
- I don't know what the last line does

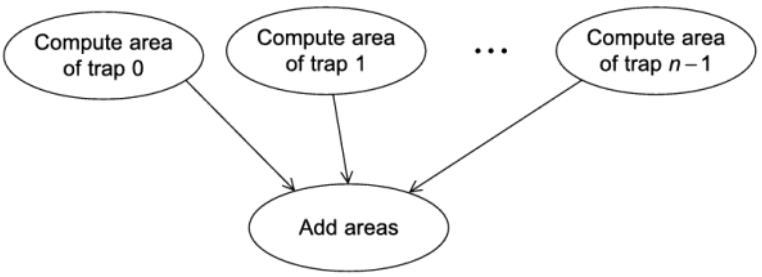
Parallelizing the trapezoidal rule

Recall that we can design a parallel program by using four basic steps:

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

We will apply these rules to parallelize the trapizoidal rule

1) Partition the problem solution into tasks:
   a. We identify two tasks: finding the area of a single trapezoid and computing the sum of those trapezoids
2) Identify the communication channels between tasks:
   a. We see that the communication channels will join each of the tasks of finding the area of a single trap. To the second task which is computing the sum of the traps.
3)
4) Tasks 3 and 4: aggregate the tasks and map them to cores.
   a. The more traps we use the more accurate thus we can see that we will have more traps to compute than we have cores, so we will have to compute the area of traps into groups.
      i. One way to do this is to split the intervals [a,b] of the curve we want to compute into comm_sz subintervals.
         1) For example if we have 20 subintervals and 8 cores we could do 20/8 so that each core has that many traps to compute evenly divided
         2) We can then have say process 0 add the estimates



- The pseudocode for this in MPI would look like:

```
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;
```

- The first take implementation of the trapizoidal rule will look like:

```
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);
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  */
```

Program 3.2: First version of MPI trapezoidal rule.

Note:

- We take a moment to talk a bit about local and global variables as it pertains to MPI
- Local variable: variables are variables whose contents are significant only on the process that's using them.
- global variables: Variables whose contents are significant to all the processes

- Trap function

```
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  */
```

Program 3.3: `Trap` function in MPI trapezoidal rule.