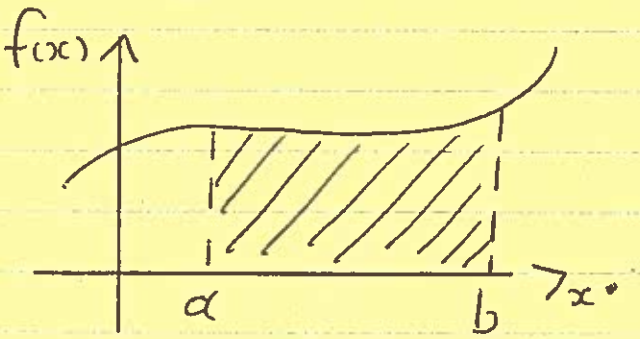
L6 (PPMP1 CH4)

NUMERICAL INTEGRATION

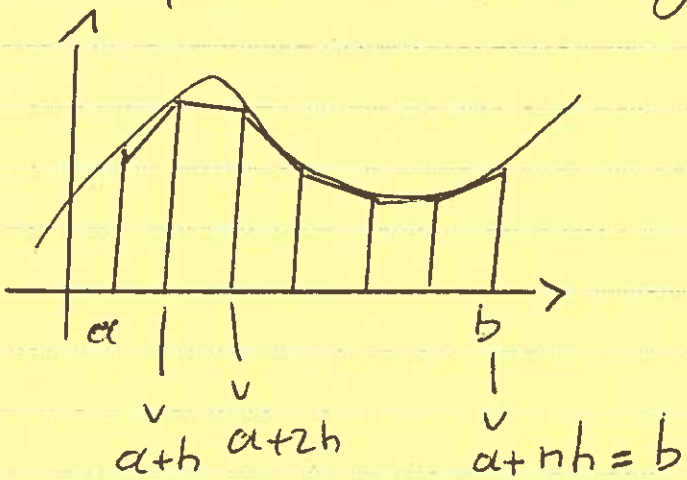
TRAPEZOIDAL RULE

$$\int_a^b f(x)\,dx$$ = area bounded by the vertical lines $x=a$ & $x=b$ and the graph of $f(x)$

(TAKE $f(x)$ to be non-negative)



approach to estimate the integral:
partition this region into trapezoids, each one based on the x-axis and on an edge joining two points on the graph of $f(x)$



we choose the bases to have the same length h,

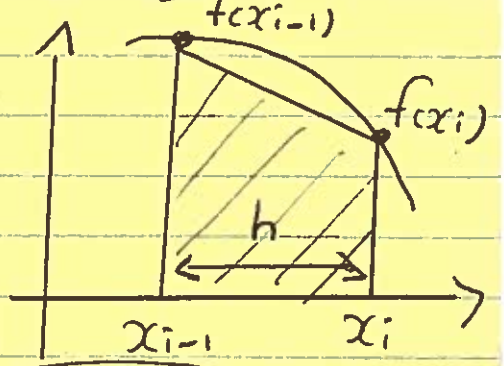
$$h = \frac{b-a}{n}$$ (case of n traps)

i-th trapezoid has basis $[a+(i-1)h,\ a+ih]$ $i=1,\dots,n$

NOTATION:
$x_i = a+ih$, $i=0,\dots,n$

area of i-th trapezoid is

$$\frac{1}{2}h\left(f(x_{i-1})+f(x_i)\right)$$



area of the entire integral approx. ~>

$$h\left[\frac{f(a)+f(b)}{2} + f(x_1)+\cdots+f(x_{n-1})\right]$$

~> serial program (with hard coded definition of f(x))

```
/* serial.c -- serial trapezoidal rule
 *
 * Calculate definite integral using trapezoidal rule.
 * The function f(x) is hardwired.
 * Input: a, b, n.
 * Output: estimate of integral from a to b of f(x)
 *    using n trapezoids.
 *
 * See Chapter 4, pp. 53 & ff. in PPMPI.
 */

#include <stdio.h>

main() {
    float  integral;   /* Store result in integral   */
    float  a, b;       /* Left and right endpoints   */
    int    n;          /* Number of trapezoids       */
    float  h;          /* Trapezoid base width       */
    float  x;
    int    i;

    float f(float x);  /* Function we're integrating */

    printf("Enter a, b, and n\n");
    scanf("%f %f %d", &a, &b, &n);

    h = (b-a)/n;
    integral = (f(a) + f(b))/2.0;
    x = a;
    for (i = 1; i <= n-1; i++) {
        x = x + h;
        integral = integral + f(x);
    }
    integral = integral*h;

    printf("With n = %d trapezoids, our estimate\n",
        n);
    printf("of the integral from %f to %f = %f\n",
        a, b, integral);
} /* main */


float f(float x) {
    float return_val;
    /* Calculate f(x).  Store calculation in return_val. */

    return_val = x*x;
    return return_val;
} /* f */
```

Parallelization of Trap Rule

we can assign a subinterval of $[a, b]$ to each processor, and each process computes the integral of f over the subinterval, + summing all interm. results at the end.

suppose that p ($\#proc^s$) divides n ($\#trap^s$) exactly $\left(\frac{n}{p} = q\right)$

process	subinterval
0	$[a, a+qh]$
1	$[a+qh, a+2qh]$
⋮	
i	$[a+iqh, a+(i+1)qh]$
⋮	
$p-1$	$[a+(p-1)qh, b]$

{ each process needs to know // $\#proc^s p$, rank, $[a, b]$, n

MPI_Comm_size, MPI_Comm_rank
we also hardcode a, b, n to avoid I/O complications

how are interm. results added?
send each $proc^s$ result to proc 0.

~> MPI program // (w/out I/O)

```
/* trap.c -- Parallel Trapezoidal Rule, first version
 *
 * Input: None.
 * Output:  Estimate of the integral from a to b of f(x)
 *    using the trapezoidal rule and n trapezoids.
 *
 * Algorithm:
 *    1.  Each process calculates "its" interval of
 *        integration.
 *    2.  Each process estimates the integral of f(x)
 *        over its interval using the trapezoidal rule.
 *    3a. Each process != 0 sends its integral to 0.
 *    3b. Process 0 sums the calculations received from
 *        the individual processes and prints the result.
 *
 * Notes:
 *    1.  f(x), a, b, and n are all hardwired.
 *    2.  The number of processes (p) should evenly divide
 *        the number of trapezoids (n = 1024)
 *
 * See Chap. 4, pp. 56 & ff. in PPMPI.
 */
#include <stdio.h>

/* We'll be using MPI routines, definitions, etc. */
#include "mpi.h"


main(int argc, char** argv) {
    int         my_rank;   /* My process rank           */
    int         p;         /* The number of processes   */
    float       a = 0.0;   /* Left endpoint             */
    float       b = 1.0;   /* Right endpoint            */
    int         n = 1024;  /* Number of trapezoids      */
    float       h;         /* Trapezoid base length     */
    float       local_a;   /* Left endpoint my process  */
    float       local_b;   /* Right endpoint my process */
    int         local_n;   /* Number of trapezoids for  */
                           /* my calculation            */
    float       integral;  /* Integral over my interval */
    float       total;     /* Total integral            */
    int         source;    /* Process sending integral  */
    int         dest = 0;  /* All messages go to 0      */
    int         tag = 0;
    MPI_Status  status;

    float Trap(float local_a, float local_b, int local_n,
              float h);    /* Calculate local integral  */

    /* Let the system do what it needs to start up MPI */
    MPI_Init(&argc, &argv);

    /* Get my process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* Find out how many processes are being used */
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    h = (b-a)/n;     /* h is the same for all processes */
    local_n = n/p;  /* So is the number of trapezoids */
```

```
    /* Length of each process' interval of
     * integration = local_n*h.  So my interval
     * starts at: */
    local_a = a + my_rank*local_n*h;
    local_b = local_a + local_n*h;
    integral = Trap(local_a, local_b, local_n, h);

    /* Add up the integrals calculated by each process */
    if (my_rank == 0) {
        total = integral;
        for (source = 1; source < p; source++) {
            MPI_Recv(&integral, 1, MPI_FLOAT, source, tag,
                MPI_COMM_WORLD, &status);
            total = total + integral;
        }
    } else {
        MPI_Send(&integral, 1, MPI_FLOAT, dest,
            tag, MPI_COMM_WORLD);
    }

    /* Print the result */
    if (my_rank == 0) {
        printf("With n = %d trapezoids, our estimate\n",
            n);
        printf("of the integral from %f to %f = %f\n",
            a, b, total);
    }

    /* Shut down MPI */
    MPI_Finalize();
} /*  main  */


float Trap(
          float  local_a   /* in */,
          float  local_b   /* in */,
          int    local_n   /* in */,
          float  h         /* in */) {

    float integral;   /* Store result in integral  */
    float x;
    int i;

    float f(float x); /* function we're integrating */

    integral = (f(local_a) + f(local_b))/2.0;
    x = local_a;
    for (i = 1; i <= local_n-1; i++) {
        x = x + h;
        integral = integral + f(x);
    }
    integral = integral*h;
    return integral;
} /*  Trap  */


float f(float x) {
    float return_val;
    /* Calculate f(x). */
    /* Store calculation in return_val. */
    return_val = x*x;
```

comments on trap.c

(1) use SPMD paradigm.
proc 0 executes a different code than proc^s 1, ..., p-1 by branching based on process rank.

(2) careful distinction btw var^s
& global to all proc^s ~> a, b, n
& local to individual proc^s ~>
local_a
local_b
local_n

(3) subtle point:
proc_0 computes an integral
AND the sum of all other integrals from proc^s 1, ..., p-1

I/O on parallel systems

suppose that we added

scanf ("%f%f%d", &a, &b, &n);

in the parallel program.

the user types | 0 1 1024 |

which proc(s) get which data?

what happens when several proc's write output to a file?

assumption: process 0 can do I/O

implication: process 0 needs to send the user input to other processes.

~> I/O fct Get_data
(uses MPI_Send, MPI_Recv)

Comments on Get_data fct

(1) we use different tags for the messages containing a_ptr, b_ptr, n_ptr

(2) some systems allow each process to read from standard input and write to standard output

```
/* get_data.c -- Parallel Trapezoidal Rule, uses basic Get_data function
for
 *     input.
 *
 * Input:
 *    a, b: limits of integration.
 *    n: number of trapezoids.
 * Output:  Estimate of the integral from a to b of f(x)
 *    using the trapezoidal rule and n trapezoids.
 *
 * Notes:
 *    1.  f(x) is hardwired.
 *    2.  Assumes number of processes (p) evenly divides
 *        number of trapezoids (n).
 *
 * See Chap. 4, pp. 60 & ff in PPMPI.
 */
#include <stdio.h>

/* We'll be using MPI routines, definitions, etc. */
#include "mpi.h"

main(int argc, char** argv) {
    int         my_rank;   /* My process rank           */
    int         p;         /* The number of processes   */
    float       a;         /* Left endpoint             */
    float       b;         /* Right endpoint            */
    int         n;         /* Number of trapezoids      */
    float       h;         /* Trapezoid base length     */
    float       local_a;   /* Left endpoint my process  */
    float       local_b;   /* Right endpoint my process */
    int         local_n;   /* Number of trapezoids for  */
                           /* my calculation            */
    float       integral;  /* Integral over my interval */
    float       total;     /* Total integral            */
    int         source;    /* Process sending integral  */
    int         dest = 0;  /* All messages go to 0      */
    int         tag = 0;
    MPI_Status  status;

    void Get_data(float* a_ptr, float* b_ptr,
         int* n_ptr, int my_rank, int p);
    float Trap(float local_a, float local_b, int local_n,
              float h);    /* Calculate local integral  */

    /* Let the system do what it needs to start up MPI */
    MPI_Init(&argc, &argv);

    /* Get my process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* Find out how many processes are being used */
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    Get_data(&a, &b, &n, my_rank, p);

    h = (b-a)/n;     /* h is the same for all processes */
    local_n = n/p;  /* So is the number of trapezoids */

    /* Length of each process' interval of
     * integration = local_n*h.  So my interval
```

```
     * starts at: */
    local_a = a + my_rank*local_n*h;
    local_b = local_a + local_n*h;
    integral = Trap(local_a, local_b, local_n, h);

    /* Add up the integrals calculated by each process */
    if (my_rank == 0) {
        total = integral;
        for (source = 1; source < p; source++) {
            MPI_Recv(&integral, 1, MPI_FLOAT, source, tag,
                MPI_COMM_WORLD, &status);
            total = total + integral;
        }
    } else {
        MPI_Send(&integral, 1, MPI_FLOAT, dest,
            tag, MPI_COMM_WORLD);
    }

    /* Print the result */
    if (my_rank == 0) {
        printf("With n = %d trapezoids, our estimate\n",
            n);
        printf("of the integral from %f to %f = %f\n",
            a, b, total);
    }

    /* Shut down MPI */
    MPI_Finalize();
} /*  main  */


/*********************************************************************/
/* Function Get_data
 * Reads in the user input a, b, and n.
 * Input parameters:
 *     1.  int my_rank:  rank of current process.
 *     2.  int p:  number of processes.
 * Output parameters:
 *     1.  float* a_ptr:  pointer to left endpoint a.
 *     2.  float* b_ptr:  pointer to right endpoint b.
 *     3.  int* n_ptr:  pointer to number of trapezoids.
 * Algorithm:
 *     1.  Process 0 prompts user for input and
 *         reads in the values.
 *     2.  Process 0 sends input values to other
 *         processes.
 */
void Get_data(
          float*  a_ptr    /* out */,
          float*  b_ptr    /* out */,
          int*    n_ptr    /* out */,
          int     my_rank  /* in  */,
          int     p        /* in  */) {

    int source = 0;    /* All local variables used by */
    int dest;          /* MPI_Send and MPI_Recv       */
    int tag;
    MPI_Status status;

    if (my_rank == 0){
        printf("Enter a, b, and n\n");
```

```
        scanf("%f %f %d", a_ptr, b_ptr, n_ptr);
        for (dest = 1; dest < p; dest++){
            tag = 0;
            MPI_Send(a_ptr, 1, MPI_FLOAT, dest, tag,
                MPI_COMM_WORLD);
            tag = 1;
            MPI_Send(b_ptr, 1, MPI_FLOAT, dest, tag,
                MPI_COMM_WORLD);
            tag = 2;
            MPI_Send(n_ptr, 1, MPI_INT, dest, tag,
                MPI_COMM_WORLD);
        }
    } else {
        tag = 0;
        MPI_Recv(a_ptr, 1, MPI_FLOAT, source, tag,
            MPI_COMM_WORLD, &status);
        tag = 1;
        MPI_Recv(b_ptr, 1, MPI_FLOAT, source, tag,
            MPI_COMM_WORLD, &status);
        tag = 2;
        MPI_Recv(n_ptr, 1, MPI_INT, source, tag,
                MPI_COMM_WORLD, &status);
    }
} /* Get_data */


/*********************************************************************/
float Trap(
          float  local_a   /* in */,
          float  local_b   /* in */,
          int    local_n   /* in */,
          float  h         /* in */) {

    float integral;   /* Store result in integral  */
    float x;
    int i;

    float f(float x); /* function we're integrating */

    integral = (f(local_a) + f(local_b))/2.0;
    x = local_a;
    for (i = 1; i <= local_n-1; i++) {
        x = x + h;
        integral = integral + f(x);
    }
    integral = integral*h;
    return integral;
} /*  Trap  */


/*********************************************************************/
float f(float x) {
    float return_val;
    /* Calculate f(x). */
    /* Store calculation in return_val. */
    return_val = x*x;
    return return_val;
} /* f */
```