UNIVERSIDAD EAFIT
INGENIERÍA DE SISTEMAS
STO0263 TÓPICOS ESPECIALES EN TELEMÁTICA, 2020-1
GRUPOS 031 Y 032

PROYECTO 4–HPC : Área bajo la curva mediante la regla del trapecio compuesto

Agustín Nieto García	**Kevin Arley Parra Henao**	**Santiago Castrillon**
201710009010	201710093010	201710148010
anietog1@eafit.edu.co	kaparrah@eafit.edu.co	scastrillg@eafit.edu.co

Github del proyecto:
https://github.com/anietog1/telem-s7/tree/hpc

Carpeta del drive:
https://drive.google.com/drive/folders/1e9DVnkxQEHF6vjnENCptLxVmslLVt9ha?usp=sharing

0. Ideas preliminares de proyecto 4

Hallar la integral de una función de forma analitica no siempre es posible, debido a la complejidad que pueden llegar a tener ciertas funciones. Es por esto que han surgido métodos numéricos que aproximan la solución de una integral definida, con cierto grado de error, que puede ser tolerable de acuerdo a la aplicación que se busque. Se pretenden en este caso de estudio utilizar la regla del trapecio compuesto en programas con paralelismo, para aproximar a una integral definida, de forma rápida.

1. Problema o caso de estudio a resolver

Según el teorema fundamental del cálculo, la integral definida de una función se puede resolver como sigue:

Sea $f(x)$ una función continua y definida en el intervalo $[a,b]$ y sea $F(x)$ una función primitiva de $f(x)$, entonces:

$$I = \int_a^b f(x)\ dx\ = F(b) - F(a)$$

Sin embargo, no siempre es posible encontrar la primitiva F(x). Por ejemplo, en la siguiente integral:

$$I = \int_0^1 e^{x^2}\, dx$$

Este tipo de integrales suele resolverse utilizando métodos numéricos de integración, que permiten aproximar la integral de una función. Uno de estos es el método del trapecio o regla del trapecio.

Regla del trapecio

Hace parte de un conjunto de métodos agrupados en lo que se conoce como Newton-Cotes [1]. En estos métodos se trata de aproximar la integral de la función como la integral de un polinomio.

$$I = \int_a^b f(x)\ dx \approx \int_a^b P_n(x) dx$$

Donde Pn es un polinomio de grado n, conocido como polinomio de interpolación, puesto que toma los mismos valores de la función en puntos elegidos. La regla del trapecio corresponde al caso donde se tiene un polinomio de grado n = 1.

Lo que se hace acá es que se traza una línea que une los puntos (a, f(a)) y (b,f(b)), formándose así un trapecio cuya área será aproximada al área bajo la curva:

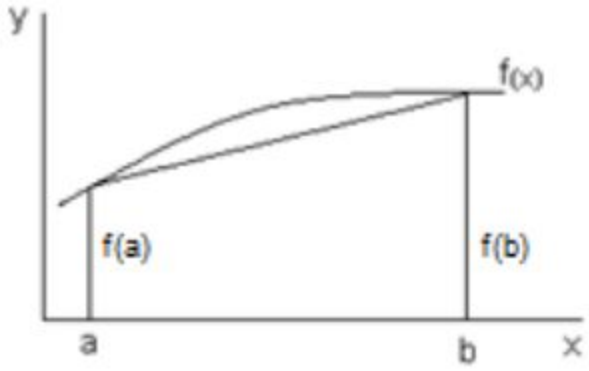


Figura 1: Representación gráfica de la regla del trapecio.
Fuente: [1]

Por tanto se tiene que:

$$I = \int_a^b f(x)\ dx \approx \int_a^b P_1(x)dx = \int_a^b (a_0 + a_1 x)\ dx = (b-a)\ \frac{f(a)+f(b)}{2}$$

conocida como *Regla del Trapecio*.

Sin embargo, para obtener una mejor aproximación, se suele dividir el intervalo en subintervalos de integración. A esto se le conoce como regla del trapecio compuesto [2]:

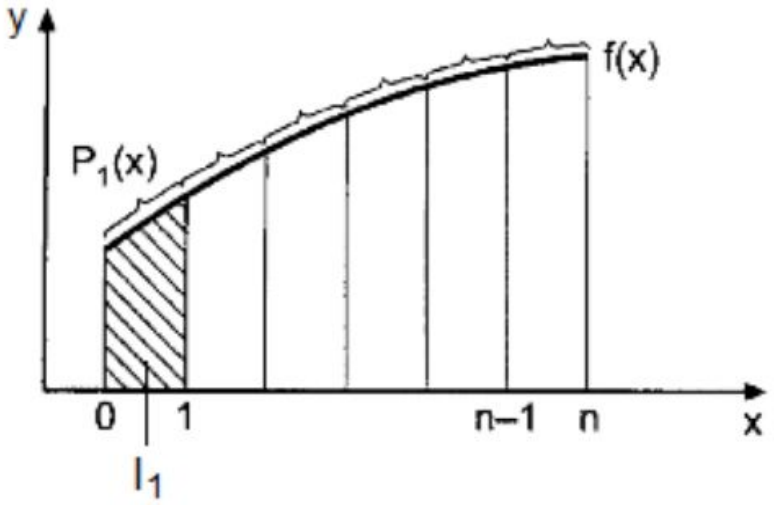


Figura 2: Representación gŕafica de la regla del trapecio compuesta.
Fuente: [1]

Partiendo de esta asunción, se puede realizar el desarrollo matemático que concluye en la siguiente fórmula:

$$\int_a^b f(x)dx \sim \frac{b-a}{n}\left[\frac{f(a)+f(b)}{2}+\sum_{k=1}^{n-1} f\left(a+k\frac{b-a}{n}\right)\right]$$

La figura presenta la formulación de la regla del trapecio compuesto [3]. En este trabajo se pretende implementar estrategias de procesamiento paralelo y HPC, para implementar este método y resolver la integral definida de una función.

Dominio: Dada una función a integrar, se tiene que el dominio del problema es el intervalo para la integral definida, y la cantidad de pasos a usar. En nuestro caso de prueba se tiene que el intervalo es [0,10], con 4*10^9 pasos.

Rango: Un valor x, tal que x pertenece a los reales y es el valor aproximado de la integral definida en el intervalo dado.

2. Objetivos y alcance

Realizar integración por medio del método del trapecio para una cantidad de segmentos masiva en poco tiempo de cómputo, utilizando técnicas de High Performance Computing (HPC) para optimizar al máximo el uso de los procesadores en la máquina en la cual se ejecutará el algoritmo diseñado.

3. Requerimientos técnicos

Cluster con el compilador Intel, OpenMP y MPI para la realización del proyecto y las distintas implementaciones en serial, multithreading y cluster.

4. Plan de trabajo

- Definir y Entender bien el problema a resolver. (entregable)

A continuación se presentan el diseño e implementación de los algoritmos en el lenguaje de programación C++. Se presentan las partes más relevantes, los detalles completos se pueden ver en el repositorio del proyecto.

- Diseñar e Implementar un algoritmo secuencial (dependiendo de la opción) en C o python en forma secuencial, tomar el tiempo de procesamiento (Ts)

```
double fs(const double x)
{
    return exp(x);
}

 double_type trapezoid(double_type x0, double_type xn, n_type n) {
   double_type h = (xn - x0) / n;
   double_type acum = 0;

   for(n_type i = 1; i < n; ++i) {
     acum += f(x0 + h * i);
   }

   return (h / 2) * (f(x0) + 2 * acum + f(xn));
 }
```

El tiempo de procesamiento que se tuvo para este programa fue:

```
1
2  ################################################################################
3  # Colfax Cluster - https://colfaxresearch.com/
4  #      Date:           Sat May 30 12:34:48 PDT 2020
5  #    Job ID:           115770.c008
6  #      User:           u42832
7  # Resources:           neednodes=1:xeonphi,nodes=1:xeonphi,walltime=00:02:00
8  ################################################################################
9
10 Time(sec): 115.530155
11 Integral of exp(0.000000, 10.000000, 40000000001): 22025.465795
12
13 ################################################################################
14 # Colfax Cluster
15 # End of output for job 115770.c008
16 # Date: Sat May 30 12:36:44 PDT 2020
17 ################################################################################
18
```

Ts = 115.53 s

- Diseñar e implementar el algoritmo paralelo en el paradigma o combinación de ellos, según sea el caso (OpenMP, MPI). Hallar el tiempo de procesamiento paralelo con diferentes recursos (Tp(P)) y hallar el SpeedUp y Eficiencia.

Algoritmo inicial con OpenMp, corriendo en una sola máquina:

```
#pragma omp declare simd simdlen(8)
double f(const double x) {
  return exp(x);
}


double trapezoid(double x0, double xn, long long n) {
  double h = (xn - x0) / n;
  double acum = 0;

#pragma omp parallel for simd simdlen(8) reduction(+: acum)
  for(long long i = 1; i < n; ++i) {
    acum += f(x0 + h * i);
  }

  return (h / 2) * (f(x0) + 2 * acum + f(xn));
}
```

Tp = 0.58 s
Speed Up (S(P)) = Ts / (Tp / P)
S(1) = 115.53 / (0.58 / 1) = 199.19
Eficiencia (E(P)) = S(P) / P
E(1) = 199.19 / 1 = 199.19

- Implementar el algoritmo paralelo según sea el caso (OpenMP, MPI)

Algoritmo con MPI y 4 nodos:

```
#pragma omp declare simd
double f(const double x)
{
    return exp(x);
}
```

```
double_type trapezoid(double_type x0, n_type start, n_type end, double_type h)
{
  double_type acum = 0.0;
#pragma omp parallel for reduction(+ \
                                   : acum)
  for (n_type i = start; i < end; ++i)
  {
    acum += f(x0 + h * i);
  }
  double_type total = 0.0;
  MPI_Allreduce(&acum, &total, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
  return total;
}

/* Starts MPI processes ... */
MPI_Init(&argc, &argv);              /* starts MPI */
MPI_Comm_rank(MPI_COMM_WORLD, &myid); /* get current process id */
MPI_Comm_size(MPI_COMM_WORLD, &p);    /* get number of processes */

l_num = (n / p) * myid;
u_num = (n / p) * myid + (n / p);
h = (b - a) / n;
if (l_num == 0)
{
  l_num = 1;
}
MPI_Barrier(MPI_COMM_WORLD);
const double t0 = omp_get_wtime();
my_result = trapezoid(a, l_num, u_num, h);
```

Tm = 0.233222 s
Speed Up (S(P)) = Ts / (Tm / P)
S(4) = 115.53 / (0.233222 / 4) = 1981,65
Eficiencia (E(P)) = S(P) / P
E(4) = 1981,65 / 4 = 495,41

- Determinar un balance entre partes del algoritmo que se desarrollarán en OpenMP, coprocesamiento y partes con MPI.

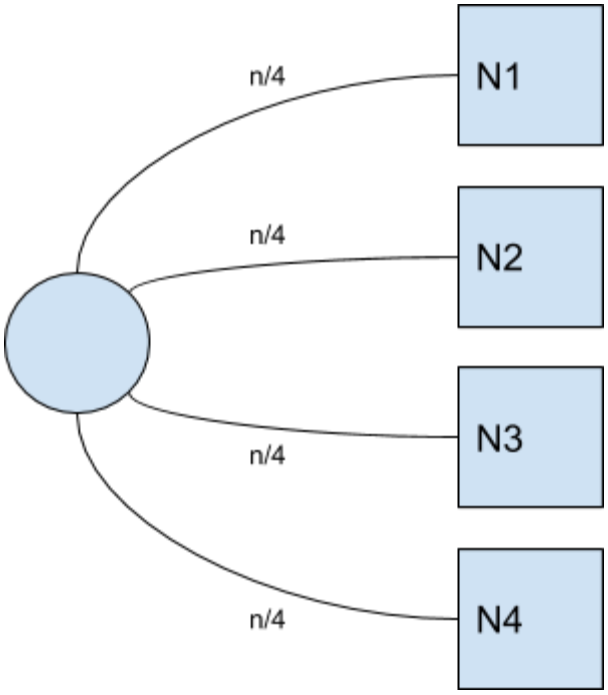
En el algoritmo que se tiene con MPI, también utilizamos openMP, combinando así el procesamiento en cluster con el multithreading.

- Aplicar la técnica de diseño de algoritmo PCAM (particionamiento, comunicación,aglomeración,mapeo a UE), en donde defina claramente el mecanismo de particionamiento (funcional y/o datos), mecanismos de comunicación entre tareas, optimización o aglomeración y finalmente el mapeo a unidades de ejecución (cores de procesamiento).

En este trabajo, se utiliza los pasos de la metodología PCAM principalmente en la solución del problema con MPI. Este metodología fue propuesta por Foster en el libro *Design and building parallel programs.* Se tienen las cuatro etapas señaladas por la metodología [4], adaptadas al problema en cuestión:

Particionamiento:

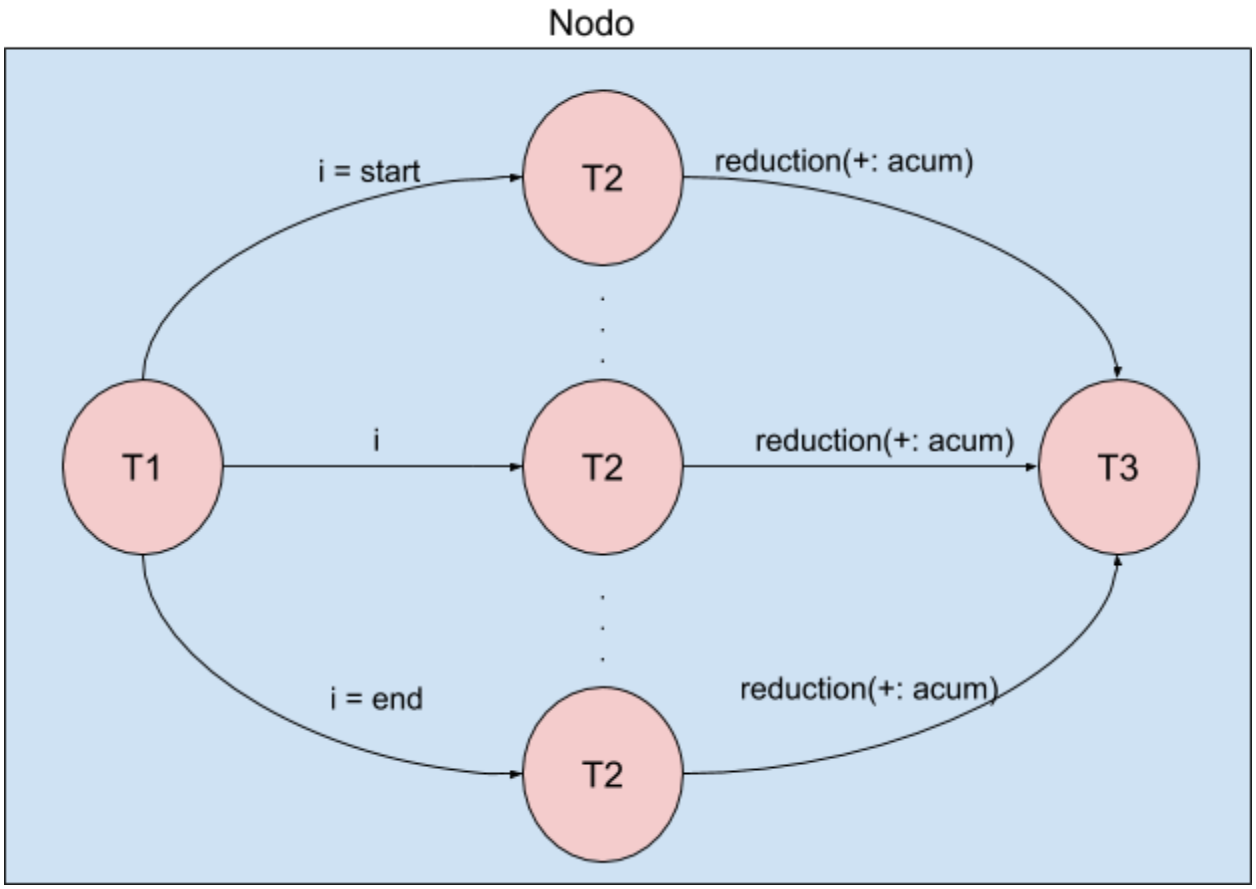
En este caso, se particiona la tarea por datos, sobre los cuales se ejecuta la misma función y se hace la final una acumulación de los resultados parciales. Se divide en N, donde N son la cantidad de nodos. En cada nodo se ejecuta también en multithreading, dependiendo de los hilos disponibles o definidos por el usuario. Dejaremos la solución final con los hilos disponibles, pues si se solicitan hilos y no hay disponibles los suficientes, el proceso tendrá que esperar y tardará más. Sin embargo se harán pruebas para mirar cómo afecta distintos hilos a la solución OpenMP.



Cada nodo realiza n/4 iteraciones para resolver la integral.

Comunicación:

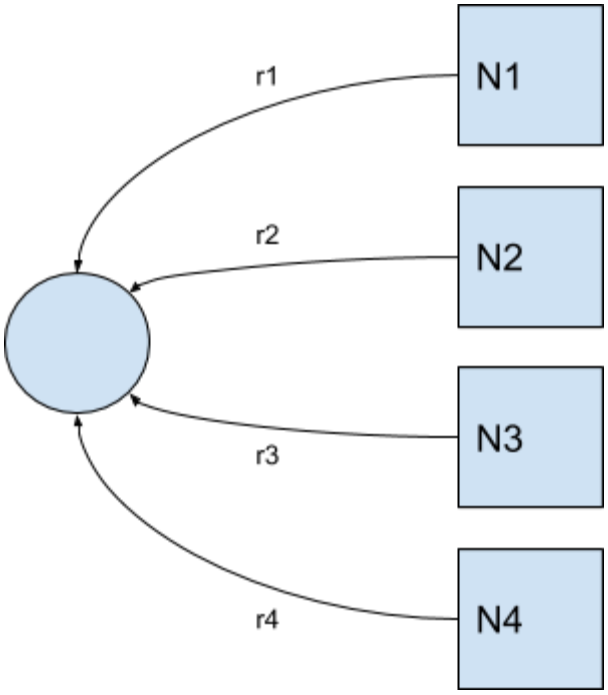
Los nodos se comunican usando la red de alta velocidad del cluster, lo que permite tener tiempos de comunicación pequeños. Se utiliza el protocolo MPIAllReduce para obtener el resultado final, que toma todos los resultados parciales y utiliza el operador suma para proveer el total de la integral en el intervalo dado. Los hilos se comunican a través de memoria compartida, que lo define simd en la función a integrar en este caso. Debido a que solo se comunica un número, que es el resultado parcial de cada nodo, y el tiempo de comunicar esto es menor que el tiempo de procesamiento, consideramos el problema como débilmente acoplado.



Dentro de cada nodo, se tienen hilos de procesamiento de acuerdo a los hilos disponibles en el sistema en un momento dado. Cada hilo operará en un subconjunto del conjunto recibido por el nodo.

Aglomeración:

Cada nodo de ejecución tendrá un subconjunto de los n pasos a realizar, y evaluará la función utilizando la regla del trapecio compuesto para obtener un acumulado parcial, en un subintervalo del intervalo inicial [a,b]. Este resultado parcial se sumará con los demás para obtener la integral total.



Cada nodo retorna un resultado parcial (ri, con i = 1-4), de la integral. Para resolverla totalmente, se suman estos resultados y se aplica otro cálculo definido por el método del trapecio.

Mapeo:

Se tendrán N = 4 nodos, para evaluar la integral definida de la función exp(x) en el intervalo [a=0, b=10], con n = 4*10^9 pasos. El resultado que se obtuvo para el algoritmo MPI es de 22025.465795, como integral definida.

- Realizar el análisis del SpeedUp y Eficiencia para varios escenarios de recursos.

Primero teníamos esto, que simplemente se encarga de paralelizar el bucle, pero no se vectorizaba debido a una dependencia y el SIMD no era explícito.

```
double_type trapezoid(double_type x0, double_type xn, n_type n)
{
  double_type h = (xn - x0) / n;
  double_type acum = 0;
#pragma omp parallel for reduction(+ \
                                   : acum)
  for (n_type i = 1; i < n; ++i)
  {
    acum += f(x0 + h * i);
  }

  return (h / 2) * (f(x0) + 2 * acum + f(xn));
}
```

```
LOOP BEGIN at main.cc(7,1) inlined into main.cc(24,24)
   remark #15344: loop was not vectorized: vector dependence prevents vectorization
   remark #15346: vector dependence: assumed FLOW dependence between at (11:5) and at (11:5)
LOOP END
```

```
[u42832@c008 openmp]$ cat process_openmp.o115769

########################################################################
# Colfax Cluster - https://colfaxresearch.com/
#      Date:           Sat May 30 12:34:35 PDT 2020
#    Job ID:           115769.c008
#      User:           u42832
# Resources:           neednodes=1:xeonphi,nodes=1:xeonphi,walltime=00:02:00
########################################################################

Time(sec): 1.582817
Integral of exp(0.000000, 10.000000, 4000000000l): 22025.465795

########################################################################
# Colfax Cluster
# End of output for job 115769.c008
# Date: Sat May 30 12:34:37 PDT 2020
########################################################################
```

También se probó con otras alternativas que no dieron tan buenos resultados. Por ejemplo, en el siguiente logramos vectorizar el ciclo además de hacerlo paralelo. Dio peor, debido posiblemente a overhead causado por una mala vectorización por defecto.

```
double_type trapezoid(double_type x0, double_type xn, n_type n)
{
  double_type h = (xn - x0) / n;
  double_type acum = 0;
#pragma omp parallel for simd reduction(+ \
                                        : acum)
  for (n_type i = 1; i < n; ++i)
  {
    acum += f(x0 + h * i);
  }

  return (h / 2) * (f(x0) + 2 * acum + f(xn));
}
```

```
LOOP BEGIN at main.cc(7,1) inlined into main.cc(24,24)
   remark #15410: vectorization support: conversion from int to float will be emulated
   remark #15305: vectorization support: vector length 8
   remark #15399: vectorization support: unroll factor set to 2
   remark #15309: vectorization support: normalized vectorization overhead 0.034
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 110
   remark #15477: vector cost: 52.870
   remark #15478: estimated potential speedup: 1.940
   remark #15484: vector function calls: 1
   remark #15487: type converts: 1
   remark #15488: --- end vector cost summary ---
   remark #15489: --- begin vector function matching report ---
   remark #15490: Function call: f(double) with simdlen=8, actual parameter types: (vecto
   remark #15492: A suitable vector variant was found (out of 2) with xmm, simdlen=2, unm
   remark #15493: --- end vector function matching report ---
LOOP END

LOOP BEGIN at main.cc(7,1) inlined into main.cc(24,24)
<Remainder loop for vectorization>
   remark #15410: vectorization support: conversion from int to float will be emulated
   remark #15305: vectorization support: vector length 8
   remark #15309: vectorization support: normalized vectorization overhead 0.096
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END
```

```
########################################################################
# Colfax Cluster - https://colfaxresearch.com/
#      Date:           Sat May 30 17:17:00 PDT 2020
#    Job ID:           115877.c008
#      User:           u42513
# Resources:           neednodes=1:xeonphi,nodes=1:xeonphi,walltime=00:02:00
########################################################################

Time(sec): 1.826674
Integral of exp(0.000000, 10.000000, 4000000000): 22025.465795

########################################################################
# Colfax Cluster
# End of output for job 115877.c008
# Date: Sat May 30 17:17:02 PDT 2020
########################################################################
```

Del siguiente modo teníamos errores y no vectorizaba debido a que no reconocía los tipos de datos.

```
double_type trapezoid(double_type x0, double_type xn, n_type n)
{
  double_type h = (xn - x0) / n;
  double_type acum = 0;
#pragma omp simd reduction(+: acum)
  for (n_type i = 1; i < n; ++i)
  {
    acum += f(x0 + h * i);
  }

  return (h / 2) * (f(x0) + 2 * acum + f(xn));
}
```

```
c(23,24)
 loop control variable i was found, but loop iteration count cannot be computed before e

"simd"
eplaced In Loop: 1
```

Al poner todo con tipos de datos del compilador sí vectorizó pero da timeout posiblemente dado que el speedup estimado por el compilador es de 0.4 únicamente.

```
double trapezoid(double x0, double xn, long long n)
{
  double h = (xn - x0) / n;
  double acum = 0;
#pragma omp simd reduction(+: acum)
  for (long long i = 1; i < n; ++i)
  {
    acum += f(x0 + h * i);
  }

  return (h / 2) * (f(x0) + 2 * acum + f(xn));
}

int main()
```

```
LOOP BEGIN at main.cc(8,20) inlined into main.cc(23,19)
   remark #15410: vectorization support: conversion from int to float will be emula
   remark #15305: vectorization support: vector length 2
   remark #15309: vectorization support: normalized vectorization overhead 0.124
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 109
   remark #15477: vector cost: 60.500
   remark #15478: estimated potential speedup: 0.420
   remark #15484: vector function calls: 1
   remark #15487: type converts: 1
   remark #15488: --- end vector cost summary ---
   remark #15489: --- begin vector function matching report ---
   remark #15490: Function call: f(double) with simdlen=2, actual parameter types:
   remark #15492: A suitable vector variant was found (out of 2) with xmm, simdlen=
   remark #15493: --- end vector function matching report ---
   remark #25015: Estimate of max trip count of loop=1073741823
LOOP END
```

Hicimos otro que intentará solventar todos esos problemas pero el speedup ganado se perdía reservando memoria, ya que para nuestro caso de prueba se requiere alrededor de 32GB de RAM:

```
double trapezoid(double x0, double xn, long long n)
{
  double h = (xn - x0) / n;
  double acum = 0;
  double *x = new double[n];

#pragma omp parallel for simd
  for (long long i = 1; i < n; ++i)
  {
    x[i] = x0 + h * i;
  }

#pragma omp parallel for simd reduction(+: acum)
  for (long long i = 1; i < n; ++i)
  {
    acum += f(x[i]);
  }

  delete[] x;

  return (h / 2) * (f(x0) + 2 * acum + f(xn));
}
```

```
################################################################################
# Colfax Cluster - https://colfaxresearch.com/
#      Date:           Sat May 30 17:44:39 PDT 2020
#    Job ID:           115889.c008
#      User:           u42513
# Resources:           neednodes=1:xeonphi,nodes=1:xeonphi,walltime=00:02:00
################################################################################

Time(sec): 3.992006
Integral of exp(0.000000, 10.000000, 4000000000): 22025.465795

################################################################################
# Colfax Cluster
# End of output for job 115889.c008
# Date: Sat May 30 17:44:44 PDT 2020
################################################################################

[u42513@c008 openmp]$
```

```
LOOP BEGIN at main.cc(9,1) inlined into main.cc(33,19)
   remark #15388: vectorization support: reference x has
   remark #15410: vectorization support: conversion from
   remark #15305: vectorization support: vector length 8
   remark #15399: vectorization support: unroll factor se
   remark #15309: vectorization support: normalized vecto
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainde
   remark #15449: unmasked aligned unit stride stores: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 9
   remark #15477: vector cost: 2.120
   remark #15478: estimated potential speedup: 3.890
   remark #15487: type converts: 1
   remark #15488: --- end vector cost summary ---
LOOP END
```

```
LOOP BEGIN at main.cc(15,1) inlined into main.cc(33,19)
   remark #15388: vectorization support: reference at (18:1
   remark #15305: vectorization support: vector length 4
   remark #15399: vectorization support: unroll factor set
   remark #15309: vectorization support: normalized vectori
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15448: unmasked aligned unit stride loads: 1
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 106
   remark #15477: vector cost: 52.250
   remark #15478: estimated potential speedup: 1.890
   remark #15484: vector function calls: 1
   remark #15488: --- end vector cost summary ---
   remark #15489: --- begin vector function matching report
   remark #15490: Function call: f(double) with simdlen=4,
   remark #15492: A suitable vector variant was found (out
   remark #15493: --- end vector function matching report -
LOOP END
```

Finalmente, con solo hacer un cambio se obtuvo mejoras significativas. Nos dimos cuenta de que la vectorización siempre era de 2 o de 4 pero sabíamos que se podía vectorizar de a 8 en nuestras máquinas, así que lo forzamos a hacer eso:

```
double trapezoid(double x0, double xn, long long n)
{
  double h = (xn - x0) / n;
  double acum = 0;
#pragma omp parallel for simd simdlen(8) reduction(+: acum)
  for (long long i = 1; i < n; ++i)
  {
    acum += f(x0 + h * i);
  }

  return (h / 2) * (f(x0) + 2 * acum + f(xn));
}
```

```
#pragma omp declare simd simdlen(8)
double f(const double x)
{
    return exp(x);
}
```

```
#pragma omp declare simd simdlen(8)
double f(const double x);
```

```
################################################################################
# Colfax Cluster - https://colfaxresearch.com/
#      Date:           Sat May 30 18:16:58 PDT 2020
#    Job ID:           115894.c008
#      User:           u42513
# Resources:           neednodes=4:flat,nodes=4:flat,walltime=00:02:00
################################################################################

Time(sec): 0.268544
Integral of exp(0.000000, 10.000000, 4000000000): 22025.465795

################################################################################
# Colfax Cluster
# End of output for job 115894.c008
# Date: Sat May 30 18:17:01 PDT 2020
################################################################################
```

```
LOOP BEGIN at main.cc(7,1) inlined into main.cc(23,19)
   remark #15410: vectorization support: conversion from int to float
   remark #15305: vectorization support: vector length 8
   remark #15399: vectorization support: unroll factor set to 2
   remark #15309: vectorization support: normalized vectorization over
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 110
   remark #15477: vector cost: 15.370
   remark #15478: estimated potential speedup: 6.660
   remark #15484: vector function calls: 1
   remark #15487: type converts: 1
   remark #15488: --- end vector cost summary ---
   remark #15489: --- begin vector function matching report ---
   remark #15490: Function call: f(double) with simdlen=8, actual par
   remark #15492: A suitable vector variant was found (out of 2) with
   remark #15493: --- end vector function matching report ---
LOOP END

LOOP BEGIN at main.cc(7,1) inlined into main.cc(23,19)
<Remainder loop for vectorization>
   remark #15410: vectorization support: conversion from int to float
   remark #15305: vectorization support: vector length 8
   remark #15309: vectorization support: normalized vectorization over
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END
```

- Implementar el algoritmo en la plataforma MPI dispuesta para el curso. (en el cluster del curso MOOC de Intel o en el cluster mpi del DCA, 192.168.10.40 con su usuario de la vpn y clave asignada)

Salida del programa en serial:

```
[u42832@c008 serial]$ cat process_serial.o115839

################################################################################
# Colfax Cluster - https://colfaxresearch.com/
#      Date:           Sat May 30 16:36:08 PDT 2020
#    Job ID:           115839.c008
#      User:           u42832
# Resources:           neednodes=1:xeonphi,nodes=1:xeonphi,walltime=00:02:00
################################################################################

Time(sec): 115.500642
Integral of exp(0.000000, 10.000000, 4000000000): 22025.465795

################################################################################
# Colfax Cluster
# End of output for job 115839.c008
# Date: Sat May 30 16:38:05 PDT 2020
################################################################################
```

Salida del programa con OpenMp en una sola máquina:

```
################################################################################
# Colfax Cluster - https://colfaxresearch.com/
#      Date:           Sat May 30 19:09:58 PDT 2020
#    Job ID:           115905.c008
#      User:           u42513
# Resources:           neednodes=1:xeonphi,nodes=1:xeonphi,walltime=00:02:00
################################################################################

Time(sec): 0.575400
Integral of exp(0.000000, 10.000000, 4000000000): 22025.465795

################################################################################
# Colfax Cluster
# End of output for job 115905.c008
# Date: Sat May 30 19:09:59 PDT 2020
################################################################################
```

Salida del programa en MPI con varias 4 nodos:

```
##########################################################################
# Colfax Cluster - https://colfaxresearch.com/
#      Date:           Sat May 30 19:10:39 PDT 2020
#    Job ID:           115906.c008
#      User:           u42513
# Resources:           neednodes=4:flat,nodes=4:flat,walltime=00:02:00
##########################################################################

Time(sec): 0.233222
Integral of exp(0.000000, 10.000000, 4000000000): 22025.465795

##########################################################################
# Colfax Cluster
# End of output for job 115906.c008
# Date: Sat May 30 19:10:42 PDT 2020
##########################################################################
```

- Realizar el análisis de rendimiento con varios números de procesadores (1: secuencial, paralelo: 2, 3 y 4 procesadores). SpeedUp y Eficiencia. Hacer conclusiones.

Los recursos que podemos controlar son el número de nodos para MPI. El número máximo es 4 nodos. Ahora analizaremos con 2 y 3 nodos el SpeedUp y la eficiencia:

```
[u42832@c008 mpi]$ make queueN N=2
echo 'cd $PBS_O_WORKDIR ; mpirun -machinefile $PBS_NODEFILE ./app' | qsub -l nodes=2:flat -N process_mpi_N_2
115890.c008
[u42832@c008 mpi]$ make queueN N=3
echo 'cd $PBS_O_WORKDIR ; mpirun -machinefile $PBS_NODEFILE ./app' | qsub -l nodes=3:flat -N process_mpi_N_3
115891.c008
[u42832@c008 mpi]$
```

N = 2

```
1
2  ##########################################################################
3  # Colfax Cluster - https://colfaxresearch.com/
4  #      Date:           Sat May 30 19:11:16 PDT 2020
5  #    Job ID:           115907.c008
6  #      User:           u42513
7  # Resources:           nednodes=2:flat,nodes=2:flat,walltime=00:02:00
8  ##########################################################################
9
10 Time(sec): 0.354924
11 Integral of exp(0.000000, 10.000000, 4000000000): 22025.465795
12
13 ##########################################################################
14 # Colfax Cluster
15 # End of output for job 115907.c008
16 # Date: Sat May 30 19:11:19 PDT 2020
17 ##########################################################################
18
```

Tm2 = 0.35 s
Speed Up (S(P)) = Ts / (Tm2 / P)
S(2) = 115.53 / (0.354924 / 2) = 660.17
Eficiencia (E(P)) = S(P) / P
E(2) =660.17 / 2 = 330.09

N = 3

```
1
2  ##########################################################################
3  # Colfax Cluster - https://colfaxresearch.com/
4  #      Date:           Sat May 30 19:12:58 PDT 2020
5  #    Job ID:           115908.c008
6  #      User:           u42513
7  # Resources:           nednodes=3:flat,nodes=3:flat,walltime=00:02:00
8  ##########################################################################
9
10 Time(sec): 0.346949
11 Integral of exp(0.000000, 10.000000, 4000000000): 22025.465740
12
13 ##########################################################################
14 # Colfax Cluster
15 # End of output for job 115908.c008
16 # Date: Sat May 30 19:13:01 PDT 2020
17 ##########################################################################
18
```

Tm3 = 0.346949 s
Speed Up (S(P)) = Ts / (Tm3 / P)
S(3) = 115.53 / (0.346949 / 3) = 990.26
Eficiencia (E(P)) = S(P) / P
E(3) = 990.26 / 3 = 330.09

Vemos que el *Speed Up* es menor en el caso de realizar la ejecución con 2 nodos y 3 nodos comparados con la ejecución en 4 nodos, por este caso se utilizaron 4 nodos para la implementación final de la respuesta.

Nos apoyamos en el curso sobre High Performance Computing por Intel, disponible en coursera en el siguiente enlace:
https://www.coursera.org/learn/parallelism-ia/

Adicionalmente, se presentan las referencias académicas en la siguiente sección.

5. Referencias

[1] *ing.unne.edu.ar*. [Online]. Available: http://www.ing.unne.edu.ar/assets/pdf/academica/departamentos/computacion/comp/IN.pdf. [Accessed: 02- Jun- 2020].

[2] F. Correa Zabala, *Métodos numéricos*, 1st ed. Medellín: Fondo Editorial Universidad EAFIT, 2010.

[3] "Regla del trapecio", *es.wikipedia.org*, 2020. [Online]. Available: https://es.wikipedia.org/wiki/Regla_del_trapecio. [Accessed: 02- Jun- 2020].

[4] I. Foster, *Designing and building parallel programs*. Reading, Mass.: Addison-Wesley, 1995.