UNIVERSIDAD TECNOLÓGICA METROPOLITANA
UTEM del Estado de Chile

Proyecto 2 Computación paralela.

Suma y multiplicación de dos números muy grandes MPI

Alumnos: *Diego Higuera*
Marco Guzmán
Docente: *Mario Fernández*
Fecha: *01/07/2017*

Tabla de contenido

Información del proyecto. **3**

Propósito y justificación del proyecto. **3**

Descripción del proyecto. **3**

Requerimientos. **4**

Requerimientos del proyecto 4

Objetivos **4**

Requisitos de aprobación del proyecto **4**

Recursos. **5**

Implementación **5**

Estudio del problema. 5

Metodologías para la suma y multiplicación. 6

Patrón de diseño paralelo presenta este problema. **7**

Análisis de resultados. **8**

Conclusiones. **9**

Referencias. **10**

Anexos. **11**

Algoritmo en C de suma secuencial. 11

Algoritmo en C de suma paralela. 13

Algoritmo en C de multiplicación secuencial. 17

Algoritmo en C de multiplicación paralela. 20

Información del proyecto.

El proyecto consiste en implementar soluciones algorítmicas a través de paradigmas secuenciales y paralelos de operaciones aritméticas, en concreto de números poseedores de extensa longitud. se trabajarán los siguientes números.

a = 627338399484900393837444599599383746458588474883837444059030384

b = 725739330343929902282028828202020282927116282893939405405950039398

Propósito y justificación del proyecto.

Este proyecto tiene como propósito analizar el rendimiento que se tiene al ejecutar un “programa” tanto en desempeño secuencial, paralelo y en clusters . Dado que este tiene como fin entender el funcionamiento básico de MPI en programación paralela.

Descripción del proyecto.

Este proyecto trata de encontrar una solución secuencial, paralela y paralela en cluster usando las librerías MPI, al siguiente enunciado:

Dados a y b, enteros muy grandes (no entran en ningún tipo de datos básico). Escriba un programa paralelo para hallar la suma y el producto de a y b.

1. Describir un algoritmo paralelo para resolver el problema en el menor tiempo posible Utilizando los recursos distribuidos en la sala de computación paralela.

2. ¿Qué patrón de diseño paralelo presenta este problema?

3. Implemente el algoritmo en C/C++

4. Determine la eficiencia del algoritmo.

Requerimientos.

A Continuación se mencionan los requerimientos principales del proyecto.

Requerimientos del proyecto

Los programas contenedores de las soluciones tanto secuenciales como paralelas deben estar escritas y compiladas bajo el lenguaje c o c++, las soluciones paralelas deben funcionar bajo las librerías de MPI como herramienta de comunicación entre nodos.

La implementación debe ser probada y en un sistema paralelo, Cluster donde al menos haya dos terminales donde deban dividirse las tareas.

Objetivos

Objetivo	Indicador	Unidad
Alcance		
Desempeño ejecución secuencial.	Tiempo de ejecución.	[Segundos]
Desempeño ejecución paralelo.	Tiempo de ejecución.	[Segundos]
Desempeño ejecución paralelo en cluster.	Tiempo de ejecución.	[Segundos]

Requisitos de aprobación del proyecto

Poseer un estudio empírico, basado en datos reales tomados del software desarrollado.

Recursos.

Recurso	Descripción
Computadores	Necesarios para ejecutar el programa tanto local como en cluster. por eso es importante disponer de más de uno.
Red ethernet	Necesario para el cluster.
SO Linux	Usado por más comodidad y confiabilidad.
Librería MPI	Necesaria para la comunicación de los nodos.
Acceso a internet	Para descargar todo lo necesario para preparar los códigos y computadores.

Implementación

Estudio del problema.

Como primer paso se estudiaron métodos matemáticos para la operatoria aritmética con números grandes, para así separar estas en pasos que pueden ser llevados en código.

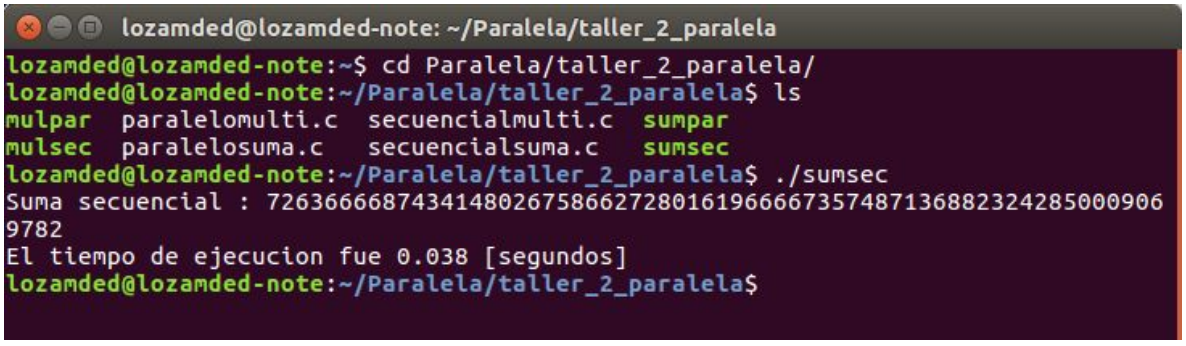
Metodologías para la suma y multiplicación.

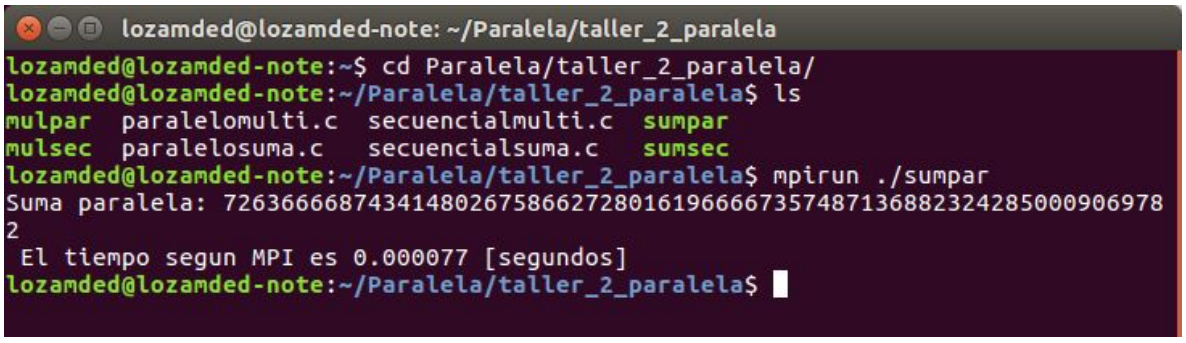
Sumar dos número es algo trivial en casi en cualquier lenguaje de programación en el caso de C y C++ las expresiones son simples, dada la variables numéricas a y b puedo establecer operatorias básicas con ellas, por ejemplo “suma = a + b” y “producto = a * b” pero cuando a y b son números compuestos por más de 60 dígitos como este caso una variable entera se queda corta en cuanto a su almacenamiento en bits, por ende la estrategia a utilizar es cambiar el enfoque trabajando en lugar de números muy grandes, utilizar cadenas de caracteres que representan el número.

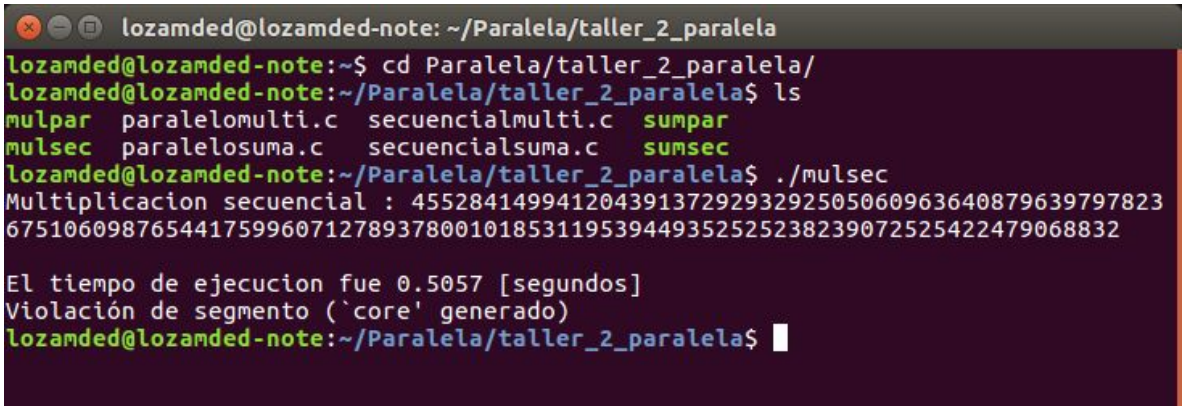
En cuanto a la programación.

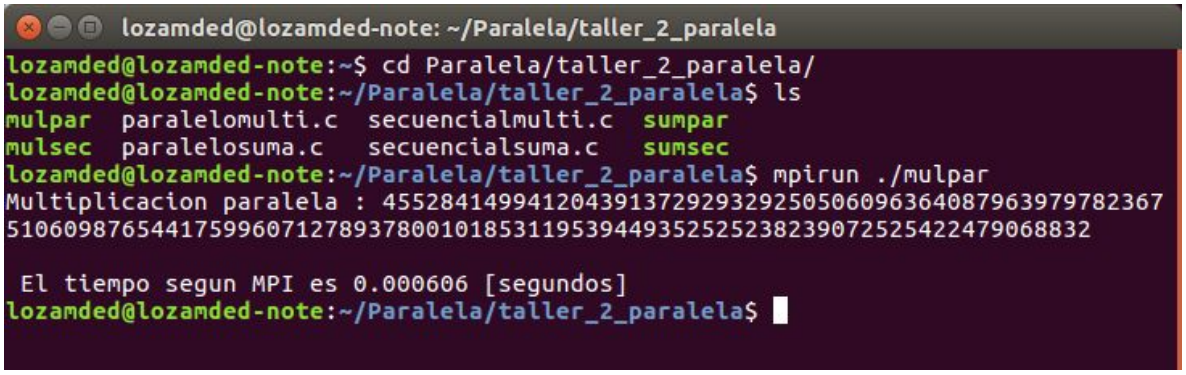
La lógica es simple, se utilizarán dos arreglos s1 y s2 (estos varían de nombre según el código), para ir trabajando los números se necesitará hacer conversiones entre tipos de dato string y número entero (int) para esto se recurrirá a utilizar atajos en el código ASCII como método de conversión por ejemplo el número en texto 5 es equivalente a 53 en código ASCII por ende si a este le restamos 48 obtendremos el número 5 pero en valor entero, asi mismo si este número le sumamos 48 volvemos a tener el valor 53 qué representa el 5 en código ASCII.

.







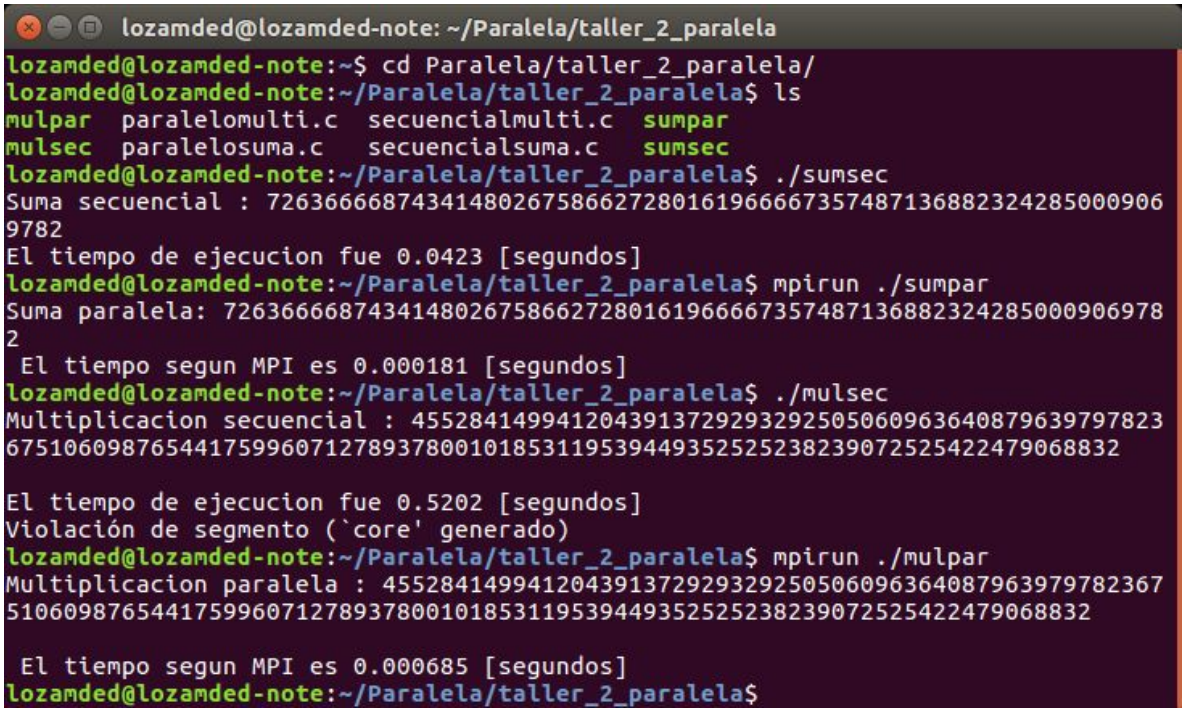


Patrón de diseño paralelo presenta este problema.

Para el diseño paralelo se notó un patrón importante, el cual era que todas las variable, las cuales se usarán para sumar o multiplicar , estas siempre están almacenadas en un array, por lo tanto es ya conocido como trabajar estos en MPI por lo que se vio en clases. En este proyecto en particular se usaron Scatter y Gather, ya que al tener los dos datos datos almacenado en un array como un string se pueden trabajar de tal forma que al momento de ejecutarlos estos se pueden ejecutarse en distintos procesadores separándolo en tamaño nuevo cada arreglo.

Por lo tanto nuestros diseño fue similar en cuanto a la multiplicación y la suma, ya que usamos los mismos principios. Separar un arreglo en diversos de un mismo tamaño para después juntar todos en una solución en orden con gather.

Análisis de resultados.



A más recursos disponibles mejor es el tiempo de ejecución, es decir cada núcleo tiene menos que trabajar.

Tiempo promedio de ejecución [segs]	Secuencial	Paralela
Suma	0.00450	0.000215
Multiplicacion	0.5541	0.000721

Conclusiones.

Como se pudo observar con los datos obtenidos, el tiempo de ejecución va disminuyendo a medida que se aumenta la cantidad de recursos que se asigna de manera efectiva, en este caso fue mediante a las librerías MPI Scatter y Gather, ya que al trabajar de acuerdo a la cantidad de procesadores estos tiene la capacidad de ejecutar un segmento del programa a trabajar, en otras palabras a medida que se asignen otros núcleos el tiempo de ejecución baja ya que están en función de estos que se solicitan a otros equipos en cuanto al sistema de cluster usado.

Referencias.

http://forum.codecall.net/topic/cc-adding-two-big-number/

https://stackoverflow.com/questions/122/multiplying-two-large-numbers

https://stackoverflow.com/questions/5322056/how-to-convert-an-ascii-character-into-an-int-in-c

Anexos.

Algoritmo en C de suma secuencial.

```
#include<stdio.h>

#include<string.h>

#include<time.h>


int main()

{


  //Variables para medir tiempo

  clock_t tiempo_i;

  clock_t tiempo_f;

  double m_seconds;


tiempo_i = clock(); //tiempo inicial


int num1[255], num2[255], sum[255];

int l1, l2;


char s1[255] = "627338399484900393837444599599383746458588474883837444059030384";

char s2[255] =
"725739330343929902282028828202020282927116282893939405405950039398";


/* convert character to integer*/
```

```
for (l1 = 0; s1[l1] != '\0'; l1++)
num1[l1] = s1[l1] - '0';
for (l2 = 0; s2[l2] != '\0'; l2++)
num2[l2] = s2[l2] - '0';

int carry = 0;
int k = 0;
int i = l1 - 1;
int j = l2 - 1;

for (; i >= 0 && j >= 0; i--, j--, k++) {
sum[k] = (num1[i] + num2[j] + carry) % 10;
carry = (num1[i] + num2[j] + carry) / 10;
}

if (l1 > l2) {
while (i >= 0) {
sum[k++] = (num1[i] + carry) % 10;
carry = (num1[i--] + carry) / 10;
}

} else if (l1 < l2) {
while (j >= 0) {
sum[k++] = (num2[j] + carry) % 10;
```

```
carry = (num2[j--] + carry) / 10;

}

} else {

if (carry > 0)

sum[k++] = carry;

}


printf("Suma secuencial : ");

for (k--; k >= 0; k--)

printf("%d", sum[k]);


tiempo_f = clock();//tiempo final

m_seconds = (double) (tiempo_f - tiempo_i)/CLOCKS_PER_SEC;//tiempo de ejecución.

printf("\nEl tiempo de ejecucion fue %.16g [segundos]\n", m_seconds * 100.0);


return 0;

}
```

Algoritmo en C de suma paralela.

```
#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include <time.h>


int main(int argc, char *argv[]) {
```

```
int num_procs,my_id;

int cantidad ;

int recibo2[66];

int respuesta[1020];


int num1[255], num2[255], sum[255];

char s1[255]="627338399484900393837444599599383746458588474883837444059030384";

char
s2[255]="725739330343929902282028828202020282927116282893939405405950039398";

int l1, l2;


int max=66;


MPI_Status stat;

MPI_Init(&argc, &argv);

// MPI_Comm_size(comm,&gsize):

MPI_Comm_rank(MPI_COMM_WORLD, &my_id);

MPI_Comm_size(MPI_COMM_WORLD, &num_procs);


double MPI_Wtime();

  double t1 = MPI_Wtime();


cantidad = max / num_procs;

int recibo[cantidad];
```

```
        for (l1 = 0; s1[l1] != '\0'; l1++)
        num1[l1] = s1[l1] - '0';

        for (l2 = 0; s2[l2] != '\0'; l2++)
        num2[l2] = s2[l2] - '0';

int carry = 0;
int k = 0;
int i = l1 - 1;
int j = l2 - 1;

MPI_Scatter(num2,cantidad,MPI_INT,&recibo,cantidad,MPI_INT,0,MPI_COMM_WORLD);

for (; i >= 0 && j >= 0; i--, j--, k++) {
sum[k] = (num1[i] + num2[j] + carry) % 10;
carry = (num1[i] + num2[j] + carry) / 10;
}
if (l1 > l2) {

while (i >= 0) {
sum[k++] = (num1[i] + carry) % 10;
carry = (num1[i--] + carry) / 10;
}

} else if (l1 < l2) {
```

```
while (j >= 0) {
sum[k++] = (num2[j] + carry) % 10;
carry = (num2[j--] + carry) / 10;
}
} else {
if (carry > 0)
sum[k++] = carry;
}

MPI_Gather(sum,cantidad,MPI_INT,recibo2,cantidad,MPI_INT,0,MPI_COMM_WORLD);
double t2 = MPI_Wtime();
  double tf = t2 - t1;

MPI_Finalize();

printf("Suma paralela: ");
for (k--; k >= 0; k--)
printf("%d", recibo2[k]);

  printf("\n El tiempo segun MPI es %f [segundos]\n", tf);

return 0;
}
```

Algoritmo en C de multiplicación secuencial.

```
#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#include<string.h>

#include<time.h>

#define MAX 1000


int main()

{

  //Variables para medir tiempo

  clock_t tiempo_i;

  clock_t tiempo_f;

  double m_seconds;


  char a[MAX] = "627338399484900393837444599599383746458588474883837444059030384";

  char b[MAX] =
"725739330343929902282028828202020282927116282893939405405950039398";

  char c[MAX];

  char temp[MAX];

  int la=strlen(a)-1;

  int lb=strlen(b)-1;


  char mul[MAX]; //static
```

```
int i,j,k=0,x=0,y;
long int r=0;
long sum = 0;

for(i=0;i<=la;i++)
{
        a[i] = a[i] - 48;

}

for(i=0;i<=lb;i++)
{
        b[i] = b[i] - 48;
}

for(i=lb;i>=0;i--)
{
        //printf("k antes del for: %d ",k);
        r=0;
        for(j=la;j>=0;j--)
        {
        temp[k++] = (b[i]*a[j] + r)%10;
        //printf("temp[%d] = %d",k,temp[k]);
        //printf("k en el primer for: %d ",k);
        r = (b[i]*a[j]+r)/10;
```

```
        }
        temp[k++] = r;
        // printf("k al salir del for %d ",k);
        x++;
        for(y = 0;y<x;y++)
        {
        temp[k++] = 0;
        // printf("temp[%d] = %d",k,temp[k]);
        //printf("k en el segundo for %d \n",k);
        }
}

k=0;
r=0;
for(i=0;i<la+lb+2;i++){
        sum =0;
        y=0;
        for(j=1;j<=lb+1;j++){
        if(i <= la+j){
        sum = sum + temp[y+i];
        }
        y += j + la + 1;
        }
        c[k++] = (sum+r) %10;
        r = (sum+r)/10;
```

```
}
c[k] = r;
j=0;
for(i=k-1;i>=0;i--){
        mul[j++]=c[i] + 48;
}
mul[j]='\0';

printf("Multiplicacion secuencial : ");
printf("%s\n",mul);

tiempo_f = clock();//tiempo final
m_seconds = (double) (tiempo_f - tiempo_i)/CLOCKS_PER_SEC;//tiempo de ejecución.
printf("\nEl tiempo de ejecucion fue %.16g [segundos]\n", m_seconds * 100.0);

return 0;
}
```

Algoritmo en C de multiplicación paralela.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>
#include "string.h"
#define MAX 100000
```

```
int main(int argc, char *argv[]) {
  int num_procs,my_id;
  int cantidad ;
  char recibo2[MAX];
  int respuesta[MAX];
  char a[MAX] = "627338399484900393837444599599383746458588474883837444059030384";
  char b[MAX] = "725739330343929902282028828202020282927116282893939405405950039398";
  char c[MAX];
  char temp[MAX];

  MPI_Status stat;
  MPI_Init(&argc, &argv);
  // MPI_Comm_size(comm,&gsize):
  MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
  MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

  double MPI_Wtime();
    double t1 = MPI_Wtime();

  int la;
  la=strlen(a)-1;
  int lb;
  lb=strlen(b)-1;
  char mul[MAX];
  int i,j,k=0,x=0,y;
  long int r=0;
```

```
long sum = 0;


cantidad = 200 / num_procs;

int recibo[MAX];


for(i=0;i<=la;i++)

{

        a[i] = a[i] - 48;

}


for(i=0;i<=lb;i++)

{

        b[i] = b[i] - 48;

}


MPI_Scatter(b,cantidad,MPI_INT,&mul,cantidad,MPI_INT,0,MPI_COMM_WORLD);


for(i=lb;i>=0;i--)

{

        r=0;

        for(j=la;j>=0;j--)

        {

        temp[k++] = (b[i]*a[j] + r)%10;

        //printf("temp[%d] = %d",k,temp[k]);

        //printf("k en el primer for: %d ",k);

        r = (b[i]*a[j]+r)/10;
```

```
            }
            temp[k++] = r;
  // printf("k al salir del for %d ",k);
            x++;
            for(y = 0;y<x;y++)
            {
            temp[k++] = 0;
            // printf("temp[%d] = %d",k,temp[k]);
            //printf("k en el segundo for %d \n",k);
            }
}

k=0;
r=0;
for(i=0;i<la+lb+2;i++){
            sum =0;
            y=0;
            for(j=1;j<=lb+1;j++){
            if(i <= la+j){
            sum = sum + temp[y+i];
            }
            y += j + la + 1;
            }
            c[k++] = (sum+r) %10;
            r = (sum+r)/10;
}
```

```
c[k] = r;

j=0;

for(i=k-1;i>=0;i--){

        mul[j++]=c[i] + 48;

}

mul[j]='\0';



 MPI_Gather(mul,cantidad,MPI_CHAR,recibo2,cantidad,MPI_CHAR,0,MPI_COMM_WORLD);


 double t2 = MPI_Wtime();

  double tf = t2 - t1;


 MPI_Finalize();


 printf("Multiplicacion paralela : ");

 printf("%s\n",recibo2);


  printf("\n El tiempo segun MPI es %f [segundos]\n", tf);


 return 0;

}
```