Práctica No. 2: Análisis de Eficiencia y Escalabilidad

El objetivo de esta práctica es comparar el comportamiento de un algoritmo paralelo versus una versión secuencial. El problema que nos interesa es un algoritmo secuencial que cuenta el número de apariciones de un número en un arreglo muy grande.

Para simular una búsqueda en un arreglo mayor, lo que haremos es buscar N veces en la misma estructura de datos.

Versión Secuencial

Un ejemplo de este programa en versión secuencial lo puede copiar desde:
http://ldc.usb.ve/~yudith/docencia/ci-6842/Practica2/

```
//--------------------------------------------------------------------+
//                                                                    |
// CuentaSec.c: Cuenta el numero de veces que aparece un numero en    |
//              un vector muy grande.                                 |
//--------------------------------------------------------------------+
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>

#define MAX_ENTERO      1000
#define NUM_VECTORES    10000 // Simula vector todavia mayor sin ocupar espacio
                              // de memoria
#define NUM_BUSCADO     8

int main (int argc, char *argv[]) {
      struct timeval t0, tf, t;
      int i, j, laCardinalidad, numVeces;
      int *vector;
      if (argc != 2) {
            printf ("Uso: cuentaSec cardinalidad \n");
            return 0;
      }
      laCardinalidad = atoi(argv[1]);
      assert (laCardinalidad > 0);

      assert((vector=(int *)malloc(sizeof(int)*laCardinalidad))!=NULL);
      for (i=0; i<laCardinalidad; i++) {
            vector[i] = random() % MAX_ENTERO;
      }
      assert (gettimeofday (&t0, NULL) == 0);
      numVeces = 0;
      for (i=0; i<NUM_VECTORES; i++)
            for (j=0; j<laCardinalidad; j++)
                  if (vector[j] == NUM_BUSCADO) numVeces++;
      assert (gettimeofday (&tf, NULL) == 0);
      timersub (&tf, &t0, &t);
      printf ("Veces que aparece el %d = %d\n", NUM_BUSCADO, numVeces);
      printf ("Tiempo de proceso: %ld:%3ld(seg:mseg)\n",
                t.tv_sec, t.tv_usec/1000);
      return 0;
}
```

El programa se invoca con un parámetro "*cardinalidad*" que determina el tamaño del arreglo. Si fijamos una cardinalidad de 200.000, tendremos un arreglo de ese tamaño donde se colocan números generados al azar comprendidos entre 0 y 1.000 (MAX_ENTERO). En este arreglo se busca (10.000 veces) el número indicado en la constante NUM_BUSCADO.

Antes de construir una versión paralela de este programa, ejecuten la versión secuencial variando el tamaño del problema. Para ello:

- Generar el ejecutable de CuentaSec.c

- Ejecutar el programa variando la cardinalidad desde 100.000 hasta 1.000.000. Anotar en la Tabla 1 el tiempo de ejecución de cada corrida. Es conveniente ejecutarlo más de una vez y tomar el promedio de los tiempos.

Comprobar el efecto de la optimización de código por parte del compilador. Para ello, generar el ejecutable compilando con la opción de optimización del gcc.

- Generar el ejecutable de CuentaSec.c con la opción de optimización.

- Ejecutar el programa cuentaSec variando la cardinalidad desde 100.000 hasta 1.000.000. Anotar en la Tabla 2 el tiempo de ejecución para cada una de las corridas realizadas. Si hubo alguna mejora en el tiempo de ejecución, justifique que pudo ocurrir.

Versión Paralela

Generar una versión paralela sencilla de este programa en la que el maestro reparte el trabajo entre sí mismo y el resto de los procesos. El reparto consistirá en dividir el arreglo en tantos trozos como procesos. Un esqueleto de este programa, al que denominaremos CuentaPar.c, puede conseguirlo en http://ldc.usb.ve/~yudith/docencia/ci-6842/Practica2/.

```
//------------------------------------------------------------------+
//                                                                  |
// CuentaPar.c: Cuenta aparaciones de un numero en un arreglo muy   |
//               grande. Version paralela simple                    |
//                           ESQUELETO                              |
//------------------------------------------------------------------+

#include <assert.h>
#include <sys/time.h>
#include <stdlib.h>
#include <stdio.h>

#include "mpi.h"

#define MAX_ENTERO      1000
#define NUM_VECTORES    10000 // Simula vector todavia mayor
#define NUM_BUSCADO     8
```

```
//--------------------------------------------------------------------
void esclavo(void) {

}

//--------------------------------------------------------------------
void maestro (int NumProcesos, int Cardinalidad) {
      int i, totalNumVeces;
      int *vector;
      struct timeval t0, tf, t;

      // Inicializar el vector

      assert((vector =(int *)malloc(sizeof(int)*Cardinalidad))!=NULL);
      for (i=0; i<Cardinalidad; i++)
            vector[i] = random() % MAX_ENTERO;

      assert (gettimeofday (&t0, NULL) == 0);

      // Repartir trabajo

      // Computar mi trozo

      // Recoger resultados

      assert (gettimeofday (&tf, NULL) == 0);

      timersub(&tf, &t0, &t);
      printf ("Numero de veces que aparece el %d = %d\n",
                  NUM_BUSCADO, totalNumVeces);
      printf ("tiempo total = %ld:%3ld\n", t.tv_sec, t.tv_usec/1000);
}

//--------------------------------------------------------------------
int main( int argc, char *argv[] ) {
      int yo, numProcesos;

      if (argc != 2) {
            printf ("Uso: cuentaPar cardinalidad \n");
            return 0;
      }
      laCardinalidad = atoi(argv[1]);

      assert (laCardinalidad > 0);
      setbuf (stdout, NULL);
      MPI_Init (&argc, &argv);
      MPI_Comm_rank (MPI_COMM_WORLD, &yo);
      MPI_Comm_size (MPI_COMM_WORLD, &numProcesos);
      if (yo == 0) maestro(NumProcesos,laCardinalidad);
      else esclavo();
      MPI_Finalize();
      return 0;
}
```

- Completar el código del programa CuentaPar.c
- Generar el ejecutable correspondiente usando la librería de mpi
- Ejecutar el programa y anotar en la Tabla 3 el tiempo de ejecución. Ejecútelo más de una vez y tome el promedio de los tiempos.
- Calcular la aceleración obtenida con respecto a la versión secuencial y analizar el resultado. (Graficar resultados)
- Calcular la eficiencia obtenida. (Graficar resultados)
- Analizar los resultados obtenidos.

Funciones MPI a revisar

MPI_Send
MPI_Recv
MPI_Reduce

Entrega

Enviar vía correo electrónico el código de su programa paralelodel desarrollado, debidamente documentado y un breve informe con los resultados obtenidos y un análisis de los mismos.

Tabla 1. Versión Secuencial

Cardinalidad	**Sin Optimización (seg:mseg)**	**Con Optimización O3 (seg:mseg)**
100.000		
200.000		
400.000		
600.000		
800.000		
1.000.000		

Tabla 2. Versión Paralela

Cardinalidad	**P=2 (seg:mseg)**	**P=4 (seg:mseg)**	**P=8 (seg:mseg)**	**P=16 (seg:mseg)**
100.000				
200.000				
400.000				
600.000				
800.000				
1.000.000				

Tabla 3. Aceleración

Cardinalidad	P=2 (seg:mseg)	P=4 (seg:mseg)	P=8 (seg:mseg)	P=16 (seg:mseg)
100.000				
200.000				
400.000				
600.000				
800.000				
1.000.000				

Tabla 4. Eficiencia

Cardinalidad	P=2 (seg:mseg)	P=4 (seg:mseg)	P=8 (seg:mseg)	P=16 (seg:mseg)
100.000				
200.000				
400.000				
600.000				
800.000				
1.000.000				