UNIVERSIDAD DE LOS ANDES
Facultad de Ingeniería y Ciencias Aplicadas
Sistemas Operativos y Redes 2018-20

Tarea 1 - Parte 1

Hugo de la Fuente
María Pía Sánchez

1. Descripción del Algoritmo *Parallel Binsearch*.

El algoritmo de busqueda binario (binary search o binsearch) consiste en una busqueda dentro de un arreglo de numero ordenados. Para encontrar un numero se siguen los siguientes pasos:

1. Se verifica si el numero localizado en la mitad del arreglo es el buscado.
2. Si no es el numero buscado, se verifica si es mayor o menor.
3. Si es mayor, se utilizará la mitad superior para seguir buscando.
4. Si es menor, se utilizará la mitad inferior para seguir buscando.
5. Si el *target* es encontrado terminar el algoritmo.
6. Si no, volver a iniciar.

```
int target; // numero buscado
bool serial_binsearch(unsigned int array[], int n) {
    int done = false;
    int low = 0, mid, high = n-1;   // se definen cuales son los indices
                                    //inicial, medio y final
    while (low <= high && !done) {
         mid = (low + high) / 2;  // se define la mitad del arreglo
                                  //o sub-arreglo para buscar el target
         if (array[mid] < target) {
            low = mid + 1;  // el sub arreglo partiria de la mitad + 1
        }
         else if (array[mid] > target) {
            high = mid - 1;   // el sub arreglo terminaria en la mitad - 1
        }
        else if (array[mid] == target)  {
            done = true;      // se encuentra el target
            break;
        }
     }
     return done;           //se retorna true o false
```

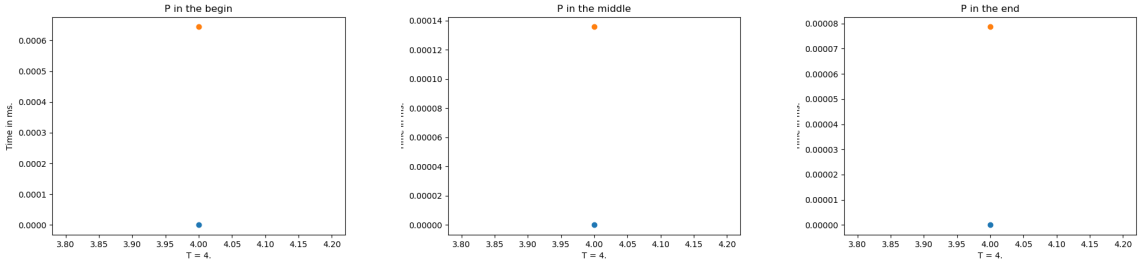
```
}
```

Para implementar el algoritmo usando threads, lo que se hizo fue darle a cada thread una porcion del arreglo, definiendo un limite de busqueda para cada thread, usando la función de la libreria unistd.h *sysconf(_SC_NPROCESSORS_ONLN)*, se puede saber cuántos cores se pueden usar dado el hardware donde se esta ejecutando el proceso principal. Luego para definir los parametros a usar por cada thread se hace uns estructura que contiene esta información y luego pasarsela al thread. El algortimo no tiene muchos cambios respecto a su version serial ya que el flujo de este algoritmo será exactamente el mismo pero repartido paralelamente entre cada thread.

2. Análisis de Resultados.

Para obtener los tiempos de ejecución de la función serial y paralela de *binsearch*, se utilizó el método *clock_gettime*. Los resultados fueron los siguientes:

1. Para $T = 4$:

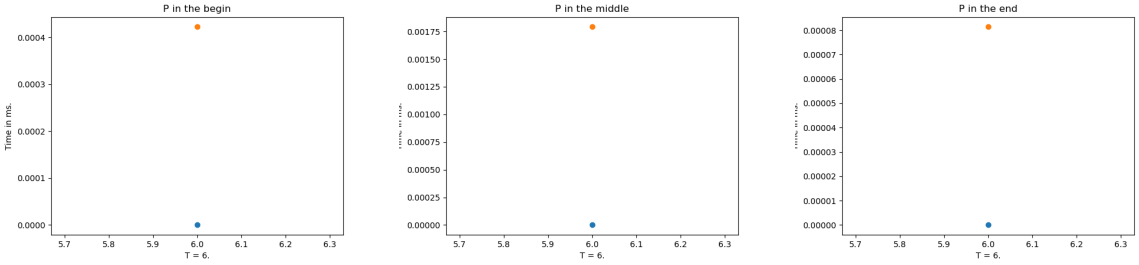


(a) $P = 0$ (b) $P = 5.000$ (c) $P = 9.999$

Figura 1: $T = 4$

Azul: *serial_binsearch*. Naranjo: *parallel_binsearch*

2. Para $T = 6$:

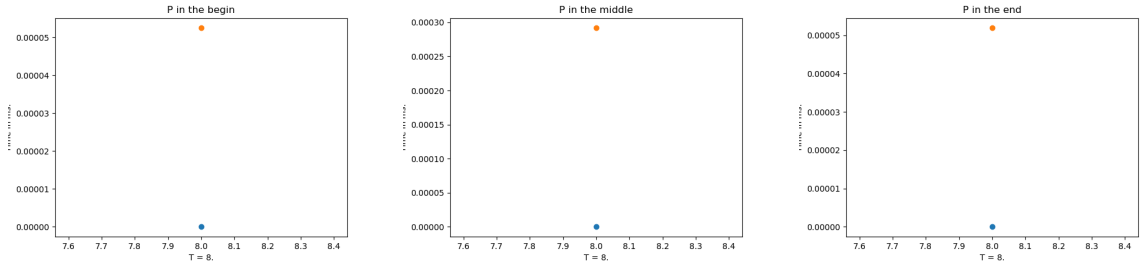


(a) $P = 0$ (b) $P = 500.000$ (c) $P = 999.999$

Figura 2: $T = 6$

Azul: *serial_binsearch*. Naranjo: *parallel_binsearch*

3. Para $T = 8$:



(a) $P = 0$ (b) $P = 50.000.000$ (c) $P = 99.999.999$

Figura 3: $T = 8$

Azul: *serial_binsearch*. Naranjo: *parallel_binsearch*

Como se puede observar, *serial_binsearch* es mucho más rápido que parallel binsearch. Esto es debido a que el serial del algoritmo es $O(log(1))$, es decir, su tiempo es constante. Sin embargo, esto también debiera ocurrir para la versión paralela de la búsqueda binaria. Esta diferencia entre la versión serial y la paralela puede darse por lo siguiente: para que haya sincronización entre los *threads*, se requieren instrucciones que deben seguir que puede gastar mucho tiempo de CPU. Otra posibilidad es que se haya medido el tiempo de CPU y no el de ejecución en el código.

3. Conclusiones.

Dados el análisis, se puede observar que el paralelizar el algoritmo serial, como lo hicimos, no mejora el rendimiento de la búsqueda binaria.

4. Problemas Encontrados y Limitaciones.

Los principales problemas encontrados en esa tarea fueron:

1. La creación de threads para implementar el *parallel_binsearch*, ya que el thread exige una forma de pasarle un metodo y sus parametros,

2. Luego para valores de T ¿6 el programa terminaba repentinamente (y apocos ciclos para llegar a los 100 que exige el enunciado) señalando un segmentation fault (core dumped), Claudio publicó una solución al error ya que este ocurria por mal manejo de memoria.