Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y
Cs. de la Computación

Nicolás Araya
Jonathan Venegas

Tarea 1

503627-1 Introducción a la computación paralela
Cecilia Hernández

Introducción

El presente informe tiene como objetivo realizar una implementación del problema de Suma de Prefijos o Scan. Para ello se utilizará el modelo PRAM para obtener una implementación paralela. También se propondrá una implementación híbrida que utiliza parte del algoritmo secuencial así como también en paralelo. Para el desarrollo del presente proyecto se utilizará C++ junto con la librería OpenMP.

1. Algoritmos implementados

Algoritmo 1:

Cada procesador primero copia un elemento a un vector auxiliar. Luego en paralelo, en un primer paso, cada procesador lee de memoria compartida el elemento vecino, computa la suma de lo que tiene con lo que acaba de leer, y escribe el resultado en el vector auxiliar, luego nuevamente en forma paralela se almacena en el vector de salida los valores obtenidos anteriormente . En el siguiente paso, cada procesador vuelve a realizar la suma con el elemento a distancia 2^i, y luego continua con los pasos siguientes hasta llegar a log(n).

Pseudocodigo:

```
aux[];
for(i=0; i < nDatos; i++) do_in_parallel{
       aux[i] = salida[i];
}

for (i = 0; i < log(nDatos); i++){
       for (j = 1<<i; j < nDatos; i++)do_in_parallel{
              aux[j] += salida[j<<i];
       }

       for (j = 1<<i; j < nDatos; i++)do_in_parallel{
              Salida[j] = aux[j];
       }
}
```

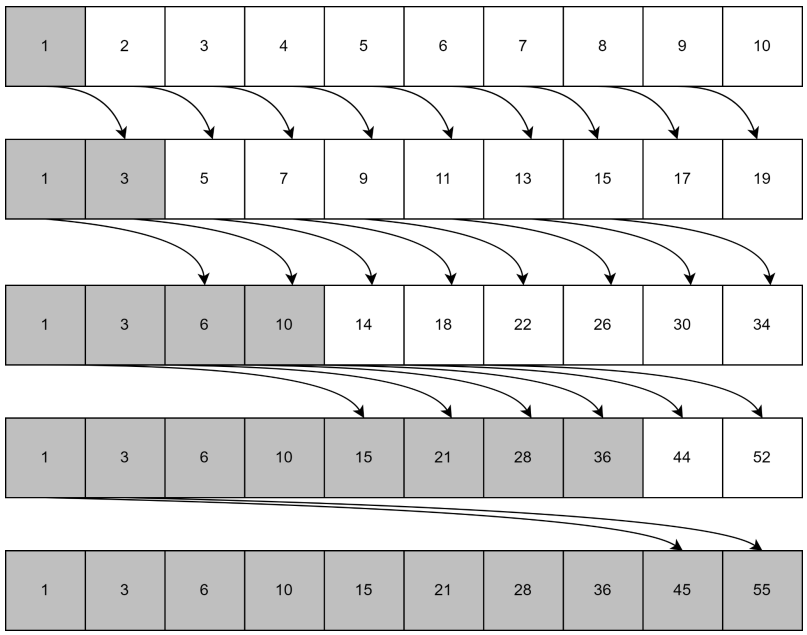

Figura 1: Diagrama Algoritmo 1

Algoritmo 2:

El arreglo es subdividido de la manera más equitativa posible para cada hebra. Luego cada hebra en paralelo se encarga de realizar la suma de prefijos de manera secuencial para su parte del subarreglo. Después, de manera secuencial, se actualiza el valor del mayor número del subarreglo con el mayor número del subarreglo anterior. Finalmente, cada hebra actualiza su subarreglo, incrementando cada elemento (excepto el elemento mayor) con el mayor del subarreglo anterior.

Pseudocodigo:

```
void secuencial(int* A[], l, r){
       for(i = l+1; i < r; i++) A[i] += A[i-1];
}

...

A[]
for (i = 0; i < nThreads; i++) do_in_parallel{
       secuencial(A[],(nDatos/nThreads)*i,(nDatos/nThreads)*(i+1));
}

for (i = 2; i < nThreads; i++){
       A[(nDatos/nThreads * i) - 1] = A [(nDatos/nThreads * (i-1)) - 1]
}
for(i = 0; i < nThreads; i++) do_in_parallel {
       secuencial(A[], ((nDatos/nThreads)*i ) - 1,((nDatos/nThreads)*(i+1))-1 )
}
```

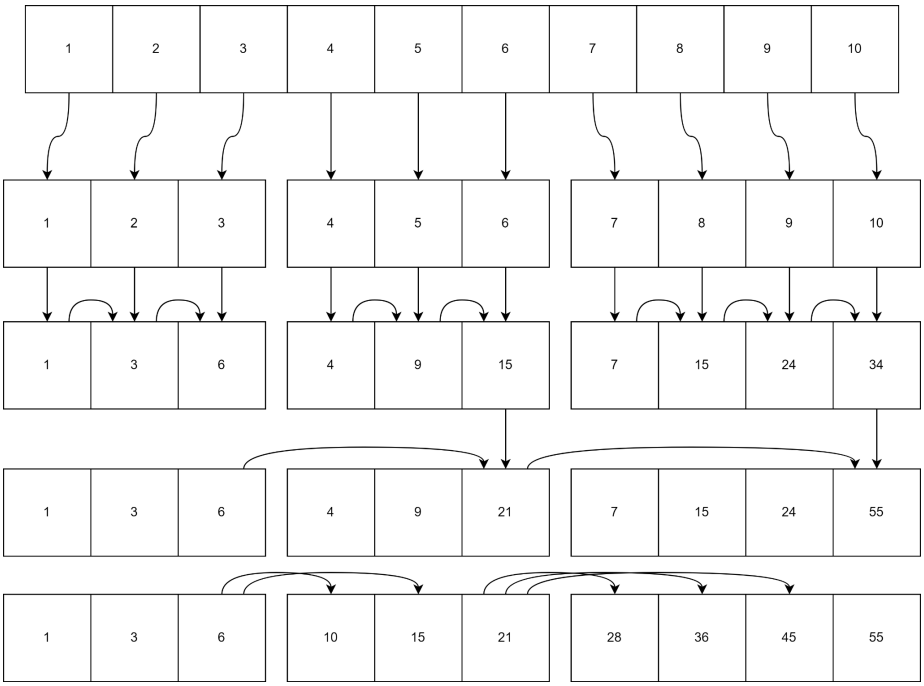

Figura 2: Diagrama Algoritmo 2

2. Análisis del modelo PRAM

Para el siguiente análisis, se considera como máquina de ejecución el servidor Chome, con 80 hebras y el algoritmo 1, ya que su implementación si es completamente paralela y teóricamente mejor.
También se considera 1 millón de elementos, es decir, $N = 1.000.000$

$S = T(N,1) \ / \ T(N,p)$

$T(N,1) = N \ log(N)$

$T(N,p) = \frac{N}{p} log(N)$

$S = p$

$E = 1$

$C = N \ log(N)$

3. Analisis Experimental

Algoritmo 1.

N° hebras	1	2	3	4	5	6
Tiempo (ms) paralelo	569	426	389	371	358	355
Tiempo (ms) secuencial	18	18	18	18	18	18

Tabla 1: Tiempo de ejecución Algoritmo 1, con 1.000.000 de elementos.

N° hebras	20	40	60	80
Tiempo (ms) paralelo	242	247	208	230
Tiempo (ms) secuencial	18	18	18	18

Tabla 2: Tiempo de ejecución Algoritmo 1, con 1.000.000 de elementos, en Chome.

Algoritmo 2.

N° hebras	1	2	3	4	5	6
Tiempo (ms) paralelo	18	24	22	22	22	23
Tiempo (ms) secuencial	18	18	18	18	18	18

Tabla 3: Tiempo de ejecución Algoritmo 2, con 1.000.000 de elementos.

N° hebras	20	40	60	80
Tiempo (ms) paralelo	14	14	17	19
Tiempo (ms) secuencial	18	18	18	18

Tabla 4: Tiempo de ejecución Algoritmo 2, con 1.000.000 de elementos, en Chome.

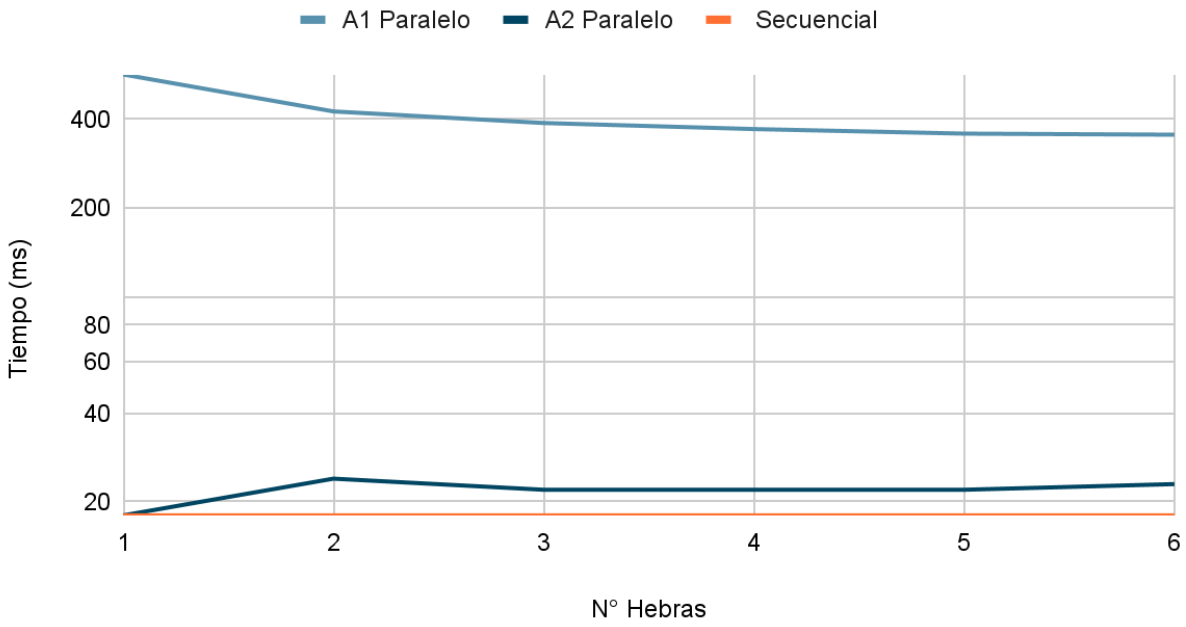

Gráfico 1: Comparativa de tiempos de ejecución.

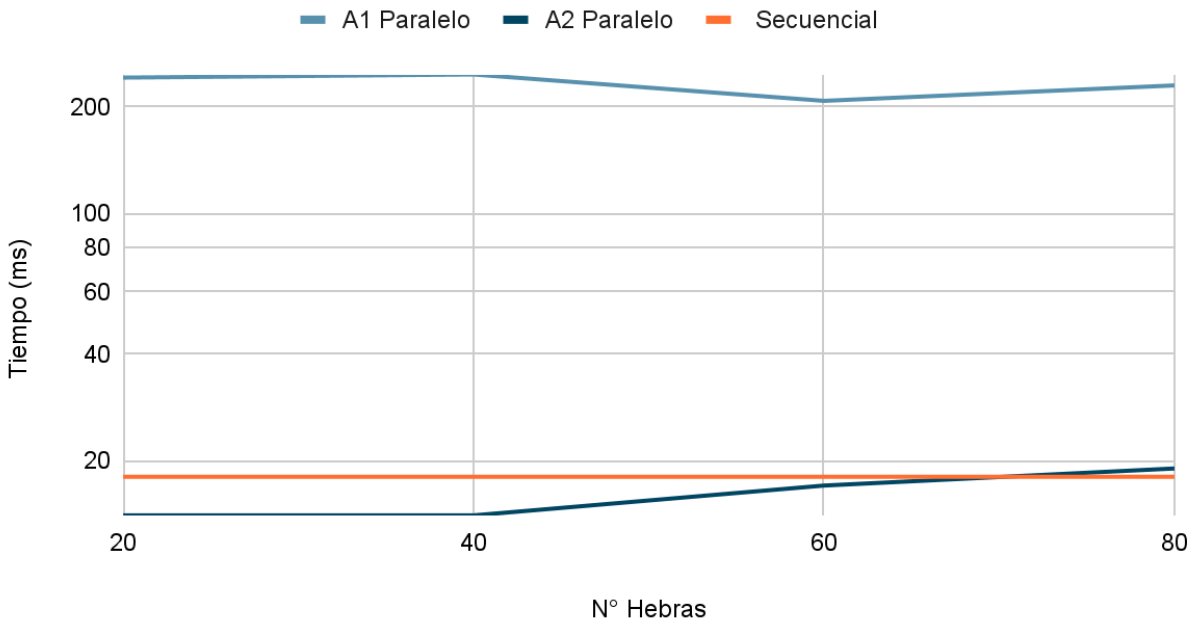

Gráfico 2: Comparativa de tiempos de ejecución en Chome.

Conclusión.

Gracias a los datos expuestos anteriormente, se puede concluir que las soluciones implementadas en paralelo en la práctica no son notablemente mejores que el algoritmo secuencial.

En la implementación del Algoritmo 1 esto puede deberse al hecho de que se considera idealmente que la mayor eficiencia se logra cuando el número de procesadores es igual al número de elementos del arreglo, lo cual es algo difícil de lograr. Además los análisis teóricos de los algoritmos no tienen en consideración los tiempos de acceso a memoria y sincronización.

Por otro lado, la implementación del Algoritmo 2 consigue mejores resultados, acercándose bastante a la solución secuencial, pero sin alcanzar a ser notoriamente mejor.