UNIVERSIDAD
DE SANTIAGO
DE CHILE

Departamento de Matemática y Ciencia de la Computación

Laboratorio 2

Comparacion entre Variables Globales y Locales en Implementaciones Recursivas.

Segundo Semestre 2016

Programación Avanzada 26106

Licenciatura en Ciencia de la Computación

Silvana Cerda Aravena
Alonso Maripi Vallejos
silvana.cerda@usach.cl / alonso.maripi@usach.c

1 Introducción

El objetivo del trabajo es construir un algoritmo usando el método Divide et Impera, este debe sumar una cantidad especifica de numeros definida por el usuario, durante la construción del algoritmo queremos definir la escalabilidad y tiempo que se demora en crear un arreglo desde el valor mínimo hasta el valor máximo permitido por nuestro computador, para la comparacioń se crearon dos algoritmos, uno trabaja el arreglo como variable global y el segundo trabaja con variables locales.

2 Procedimiento

Entendiendo el problema: Divide et Impera se caracteriza por dividir un problema en subproblemas para llegar de manera mas rápida a la solución, nuestro algoritmo que suma recibe un arreglo de numeros establecido por el usuario, este se va dividiendo hasta quedar 2 valores, estos se van sumando, de esta manera optimizamos tiempo y trabajo. Comenzamos con la contrución del algoritmo principal que posee variables globales, luego de este construimos el que posee las variables dentro de sus funciones y del int main.

2.1 Restricciones

El programa esta restringido para un valor es especifico, este valor limite depende de la capacidad de nuestro computador y de su sistema operartivo, en este caso nuestro sistema operativo de 64 bits soporta un arreglo de extencioń 19999998 .

3 Estructuras de Datos Utilizadas

Para este problema no se ocupan estructuras de datos.

4 Algoritmo

Algorithm RecSum
Input: a[0...n-1],l,r
Output: a[l]+a[r]

1 **if** $(l = r)$ **return** a[l]
2 **if** $(l + 1 = r)$ **return** a[l]+a[r]
3 $l1 \leftarrow l$
4 $r1 \leftarrow \lfloor \frac{l+r}{2.0} \rfloor - 1$
5 $l2 \leftarrow \lfloor \frac{l+r}{2.0} \rfloor$
6 $r2 \leftarrow r$
7 **return** $RecSum(a, l1, r1) + RecSum(a, l2, r2)$

Ambos algortimos poseen la misma estructura, solo se diferencian en la forma en que recibe los parametros y en la manera de llamarse recursivamente a si mismos.
El funcionamiento del algoritmo consiste en separar el largo del arreglo por la mitad y enviar las posiciones obtetidas recursivamente a si mismo con el fin de concretar ciclo por ciclo la suma de los elementos del arreglo por el lado izquierdo y derecho, que en si es el valor que retorna.

4.1 Análisis de Complejidad

El algoritmo propuesto tiene una complejidad de θ n, este es el costo por dividir el problema y combinar las soluciones.

5 Implementación

El algoritmo está implementado en lenguaje C como se muestra en la figura a continuación:

```c
/*
 * FILE: sum-G.c
 *
 * DESCRIPTION: computes a0 + a1 + ... + an
 *              using Divide et Impera approach
 *              with a global variable that save
 *              the result.
 *
 * AUTHOR: Ruben Carvajal Schiaffino
 * Edited by: Alonso Maripi Vallejos and Silvana Cerda Aravena.
 *
 * LAST REVISED: Santiago de Chile, 10/10/2016
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <sys/types.h>
#include <unistd.h>
#include <time.h>
#include <math.h>


int *a;


/*
 *
 */
int RecSum(unsigned int l, unsigned int r) {

     unsigned int l1, l2, r1, r2;

     if (l == r) return a[l];
     if (l + 1 == r) return a[l] + a[r];
     l1 = l;
     r1 = floor((float)((l + r)) / 2.0) - 1;
     l2 = floor((float)((l + r)) / 2.0);
     r2 = r;
     return RecSum(l1,r1) + RecSum(l2,r2);
}

int RecCounter(unsigned int l, unsigned int r,int cont) {

     unsigned int l1, l2, r1, r2;
     int A,B;

     if (l == r)
        return cont;
     if (l + 1 == r) return cont;
     l1 = l;
     r1 = floor((float)((l + r)) / 2.0) - 1;
     l2 = floor((float)((l + r)) / 2.0);
     r2 = r;
     A=RecCounter(l1,r1,cont+1);
     B=RecCounter(l2,r2,cont+1);
     if(A>B)
        return A;
     else
        return B;

}

/*
 *
 */
int main(int argc,char **argv) {

     unsigned int x ,n, i;
     int sum,cont;
     float E_cpu;
     clock_t cs, ce;
     long E_wall;
     time_t ts, te;
     FILE *save;

     save=fopen("sum-G.txt","a");
     x=atoi(argv[1]);
     for(n = 2;n <= x;n = n * 2){
        a = calloc(n,sizeof(int));
        for (i = 0; i < n; i = i + 1)
            a[i]=0;
        ts = time(NULL);
        cs = clock();
        sum = RecSum(0,n - 1);
        ce = clock();
        te = time(NULL);
```

5.1 Plataforma Computacional

El programa fue ejecutado en 2 computadores con las siguientes características:

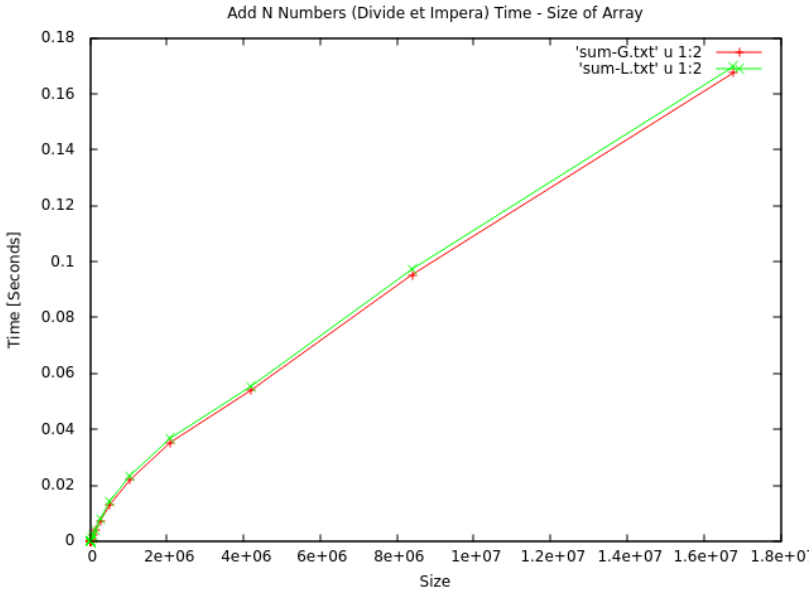
- **Computanor n1**
- **Procesador:** Intel(R) Core(TM) i7 2.9 GHz
- **Memoria RAM:** 8192MB RAM
- **Sistema Operativo:** Ubuntu Gnome 16.04

- **Computador n2**
- **Procesador:** Inter(R) Core(TM) i3-4005U 1.70 GHz
- **Memoria RAM:** 6144MB RAM
- **Sistema Operativo:** Ubuntu 15

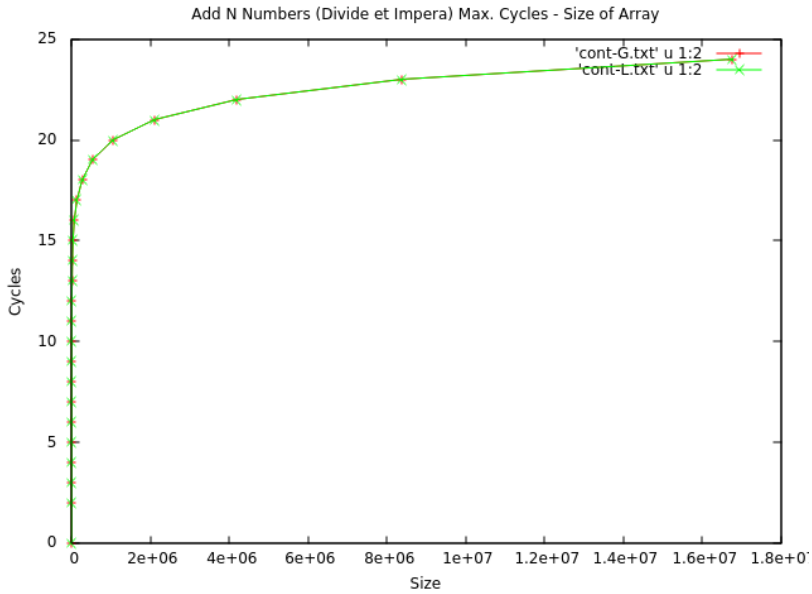
6 Resultados Experimentales

Para analizar la comparaciń entre el algortimo con variables globales y el algortimo con variables locales hemos graficado por medio de gnuplot el tiempo que tarda el programa en realizar los ciclos y en crear el arreglo en ambos computadores.

Computador n1

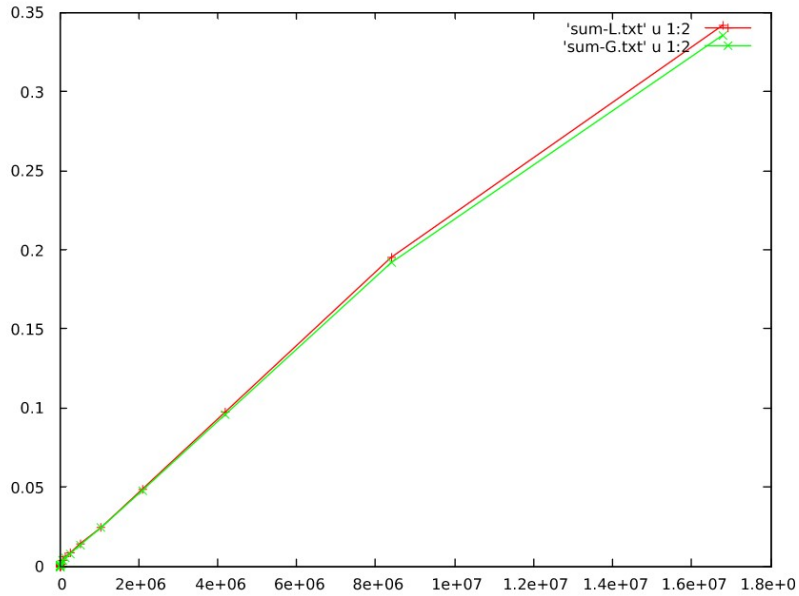


Analizando la gráfica podemos observar que la diferencia de tiempo es pequeña, en este caso el computador tarda menos tiempo utilizando el algoritmo con variables globales, esto se puede observar en la línea de color rojo.

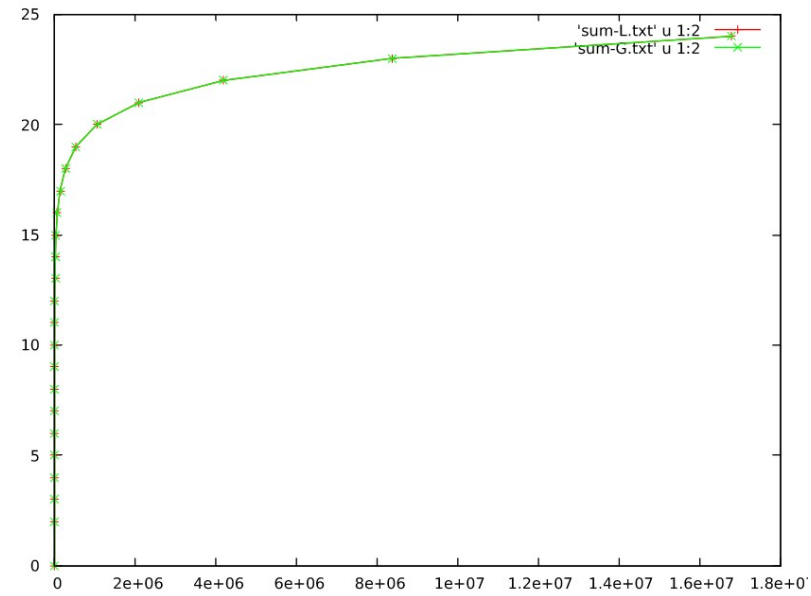


En cuanto los ciclos podemos observar que la diferencia es pareja entre ambos programas , por lo que realizan la misma cantidad de ciclos que correspondería a n-1 del n ingresado por el usuario.

Computador n2



Trabajando con el mismo valor máximo podemos observar que el algortimo con variables locales tarda un poco más .



Se cumple entre ambos programas que los ciclos son similares.