Memoria practica 2 de CPA
Louis-César Pagès
Noviembre 2018

Indice

1. Introducción

En esta prueba, vamos a ver, en primer lugar, las diferentes implementaciones propuestas y se expondrán y describirán para comprender la paralización con OpenMP de la función para encontrar los fragmentos más similares (busca_mas_parecidos). Después de eso, el tiempo de ejecución, la eficiencia y la aceleración se presentarán para analizar qué estrategias paralelas presentadas anteriormente y los horarios funcionan mejor, acompañados por una representación intuitiva de dichos resultados. Luego, teniendo en cuenta las consideraciones previas, se elige el programa y la estrategia que dieron mejores resultados y se comprueba su comportamiento comparando y analizando su eficiencia con un número creciente de subprocesos (adecuados según la arquitectura de hardware del clúster). Finalmente, se muestra y describe mi solución para el último ejercicio propuesto.

2. Definiciones

Speed-up indica la ganancia de velocidad que consigue el algoritmo paralelo con respecto a un algoritmo secuencial.

$$S(n,p) = t(n) /t(n,p) \quad (1)$$

Eficiencia mide el grado de aprovechamiento que un algoritmo paralelo hace de un computador paralelo.

$$E(n,p) = S(n,p)/p \quad (2)$$

3.Ejercicios

3.1Paralización de los ficheros

En este ejercicio se pretende medir el tiempo del bucle principal, para ello he añadido las siguientes lineas en la función main().

```
....
double t = omp_get_wtime();

    busca_mas_parecidos(nt1,tabla1,nt2,tabla2,ps);

    t = omp_get_wtime() - t;
    printf("Time needed to execute the function \"bucas_mas_parecidos\" is : %f seconds\n", t);

    escribe_resultado(salida,tabla1,nt2,ps,numeros);
...
```

3.2Estrategia 1 – Paralización de los ficheros

En este ejercicio debe compartir la carga de los dos ficheros : pacientes y donantes.

```
#pragma omp parallel sections
 {
      #pragma omp section
      err  = lee_lineas(argv[1],&nt1,&tabla1);

      #pragma omp section
      err2 = lee_lineas(argv[2],&nt2,&tabla2);
 }
```

3.3 Estrategia 2 – Paralización de la bucle mas externo

Paralización del bucle mas externo en la función busca_mas_desaparecido().

```
...
#pragma omp parallel for schedule(runtime) private(i,dm,d,im)
for ( j = 0 ; j < nt2 ; j++ ) {
  dm = MAX_LON + 1;
  for ( i = 0 ; i < nt1 ; i++ ) {
    d = distancia(tabla1[i],tabla2[j]);
    if( d < dm)
    #pragma omp critical
    if ( d < dm ) {
      dm = d;
...
```

Debe asegurarse que no hay problema de concurrencia cuando comprueba si el valor “d” es menor que el valor “dm”. Por lo tanto, “#pragma omp critical” se utiliza para evitar este problema.

3.4 Estrategia 3 – Paralización de la bucle interno

Paralización del bucle mas interno.

```
...
  #pragma omp parallel for schedule(runtime) private(d)
  for ( i = 0 ; i < nt1 ; i++ ) {
    d = distancia(tabla1[i],tabla2[j]);
    if( d < dm)
    #pragma omp critical
    if ( d < dm ) {
      dm = d;
...
```

4. Analiza

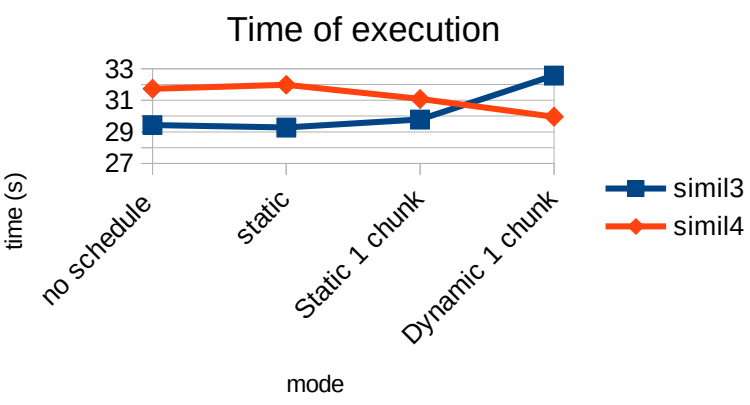
4.1 Analiza de los bucles.

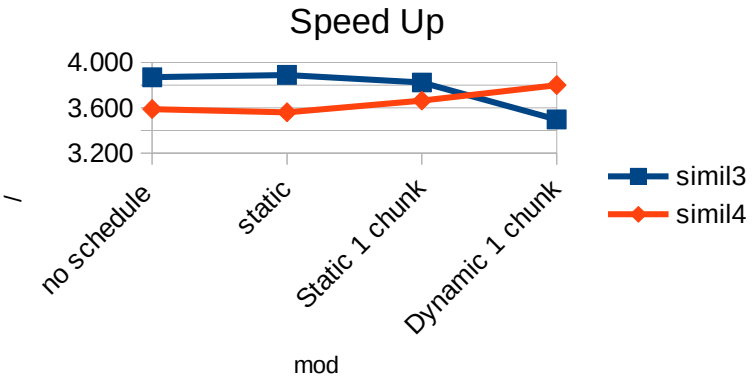
El código secuencial necesita 113.86 segundos para calcular la diferencia de cada fragmentos de código de los pacientes con los de donantes. Este tiempo sera el tiempo de referencia. Aquí tenemos las tablas y graficas de los resultados :

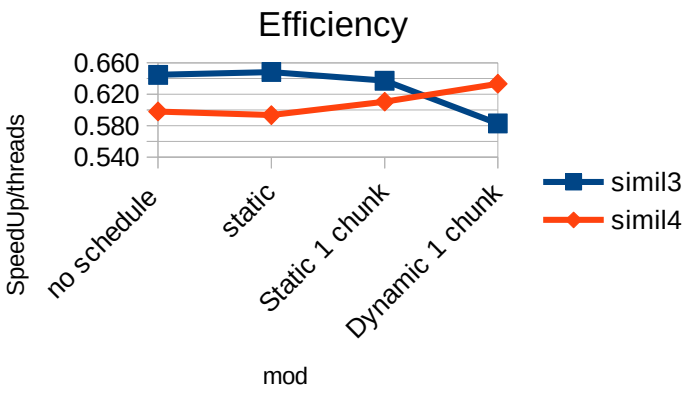
Time	simil3	simil4
no schedule	29.43	31.73
static	29.27	31.98
Static 1 chunk	29.78	31.08
Dynamic 1 chunk	32.56	29.96

SpeedUp	simil3	simil4
no schedule	3.869	3.588
static	3.890	3.560
Static 1 chunk	3.823	3.663
Dynamic 1 chunk	3.497	3.800

Efficiency	simil3	simil4
no schedule	0.645	0.598
static	0.648	0.593
Static 1 chunk	0.637	0.611
Dynamic 1 chunk	0.583	0.633







Las graficas muestran que la paralización del bucle más externo, en el programa “simil3”, sea más eficientes que la paralización del bucle interno sin planificación. Puede explicarse por la eficiencia general cuando, por un numero de elementos equivalente, la paralización del bucle mas externo sea más eficientes que el otro bucle. La inicialización de la región paralela, la planificación de los iteraciones (sin schedule) y la sincronizacion al fin de la región paralela cuestan mas en el bucle interno (overhead). Esto, se pone mejor con la planificación estática (schedule = static) que mejora la planificación entre las iteraciones. Esto solamente para el bucle externo. Sin embargo, está planificación resulta peor cuando el tamaño del chunk es 1 para el bucle externo y mejor pare el bucle interno.

La planificación dinámico con un chunk de tamaño 1 permite al bucle interno ser mas eficiente que el bucle externo. Es decir que cuando compartimos uno a uno de maniera dinámica (sin barreras entre los hilos) el calculo de la diferencia entre un código genético de un paciente con un de donante, cuales son repartidos entre los hilos, (bucle interno) es mas rápido.

Creo que esto se explica por la condición critica (critical) de saber si la diferencia es mas grande que la diferencia de referencia (dm). Por lo tanto, por cada hilo de la paralización del bucle externo va a tener más grande secciones “criticas” que bloquean el calculo de los otros hilos porque , a contrario del bucle interno, habrá un mayor numero de secciones criticas por cada hilos.

En segundo lugar, entre los programas utilizados con simil3.c, el que parecía funcionar mejor para mi era el estático, 0. Esto se debe probablemente al hecho de que cada hilo realiza las mismas iteraciones del bucle interno, por lo que no puede haber cargas de trabajo desequilibradas según el valor de j, lo que hace que un programa dinámico sea menos útil. Con respecto a la estática, 1, la ligera disminución del tiempo de ejecución puede deberse a la conmutación de contexto necesaria para proporcionar otra iteración al hilo que ha terminado con la iteración actual.

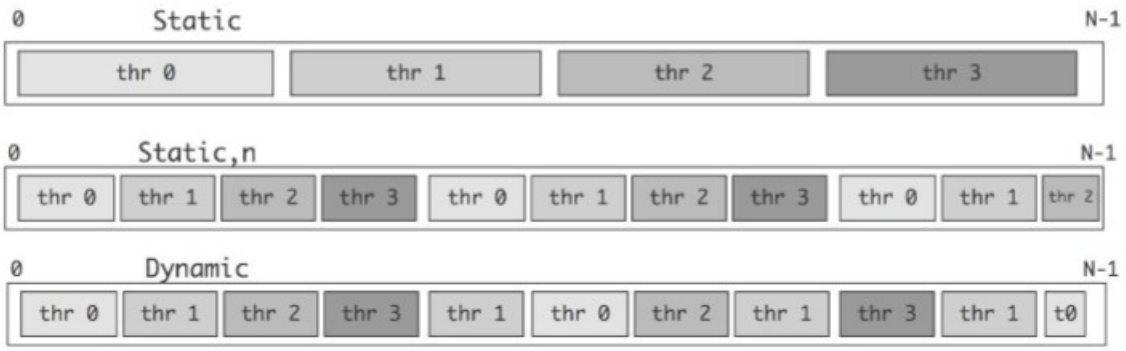
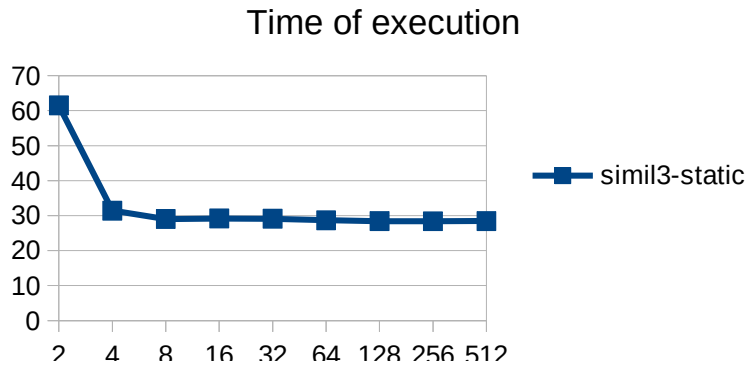


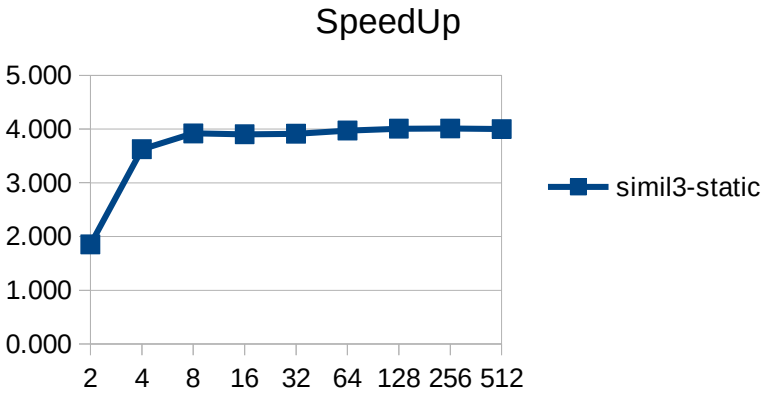
Figure 3 - Different assignments of iterations to each thread depending on the schedule

4.2 Eficiencia del “schedule”

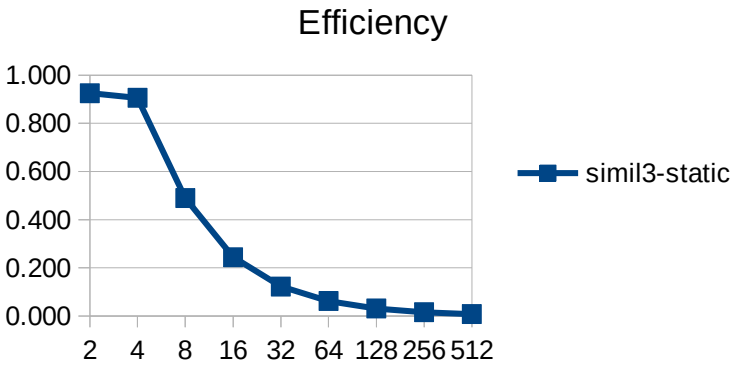
threads	simil3-static
2	61.51
4	31.42
8	29.04
16	29.17
32	29.1
64	28.67
128	28.41
256	28.39
512	28.46



SpeedUp	simil3-static
2	1.851
4	3.624
8	3.921
16	3.903
32	3.913
64	3.971
128	4.008
256	4.011
512	4.001



Efficiency	simil3-static
2	0.926
4	0.906
8	0.490
16	0.244
32	0.122
64	0.062
128	0.031
256	0.016
512	0.008



Podemos ver con estás graficas y está tablas que hay una gran diferencia entre el programa funcionando con 2 hilos y el funcionando con 4 hilos. El tiempo de ejecución se divide entre 2. Después, el aumento de hilos durante la ejecución no mejora de manera significante el tiempo de ejecución. La eficiencia se reduce rápidamente con el uso de mas hilos. Es decir que cada hilo creado por la ejecución no ayuda a reducir el calculo del programa.

5. Máximo, mínimo y promedio de los fragmentos.

Para recuperar el mínimo el máximo y el promedio del tamaño de los códigos genéticos de los pacientes, inicializamos variables y cogemos el mínimo y máximo después el primer bucle hace la suma de cada tamaño y cuenta el numero de fragmentos.

```
...
        size = strlen(tabla2[j]);
        if( size < (size_t) min){
            min = size;
        }
        else if ( size > max) max = size;
        ave += size;
        quo++;
...
```

Después, comprueba si el maximo y el minimo de cada hilo es el ultimo maximo o el ultimo minimo

```
...
    nh = omp_get_num_threads();
    printf("Hilo %d de %d : Longitud min/med/max : %d / %f / %d \n", omp_get_thread_num(), nh, min, ave/quo, max);
    if(max > max_g )
    #pragma omp critical
    if(max > max_g) max_g =max;

    if(min < min_g )
    #pragma omp critical
    if(min < min_g) min_g =min;

    ave_g += ave/quo;
...
```

En el exterior de la region paralela, annuncia el minimo, maximo y promedio genral de la funcion busca_mas_desparecido()

```
...

 }
    printf("\nTOTAL: Longitud min/med/max : %d / %f / %d \n\n", min_g, ave_g/nh,  max_g);
...
```

6. Conclusión y Extra-analizas.

En la primera estrategia, donde paralizo las lecturas de archivos, podemos observar que las operaciones de E / S no son muy significativas en un término de computación porque deben ser almacenadas por el sistema operativo. El almacenamiento en búfer grande siempre se puede intentar implementar (potencialmente con el productor / consumidor), o usar un archivo mapeado en memoria, pero las mejoras en el rendimiento no compensan el uso de este tipo de mecanismo (y el método lee_lineas () es mucho más simple).
Con respecto al primer análisis propuesto en este informe, donde tengo que la programación estática tiene un rendimiento ligeramente mejor, planteo ahora la pregunta de qué pasaría si estuviere utilizando un archivo irregular donde se extendió el código de ADN, en términos de longitud. Luego, podemos analizar y llegar a la conclusión de que con una programación estática, una de las hebras puede terminar mucho antes que otras, por lo que esta programación no sería tan eficiente como con los archivos originales.