UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Computación Paralela (CPA)

Práctica 2: Comparación de código genético

Curso 2018/19

Índice

1. Descripción del problema

En esta práctica trabajaremos en la paralelización de un programa que determina los mejores candidatos para la donación de órganos a una lista de pacientes.

En concreto, se tiene un fichero (`pacientes.txt`) con fragmentos de código genético de una serie de personas en espera de donaciones de órganos y, por otra parte, otro fichero (`donantes.txt`) con fragmentos de códigos genéticos pertenecientes a personas que han manifestado su deseo de ser donantes de órganos. Cada línea de estos ficheros es un fragmento genético, representado mediante una cadena de texto formada por las letras `A`, `G`, `C`, `T`.

El programa proporcionado procesa los dos ficheros que se le indique (el fichero de donantes y el fichero de pacientes), identificando para cada paciente el donante con mayor parecido genético, obteniendo así una lista de posibles donaciones. Posteriormente, el personal médico se encargará de revisar detalladamente la compatibilidad y viabilidad de cada caso propuesto por el programa.

Al programa se le indican como argumentos los ficheros de entrada y el fichero donde dejar la salida, como puede verse en este ejemplo de llamada:

```
$ ./simil donantes.txt pacientes.txt salida.txt
```

Puedes comprobar que el programa contiene una función (`distancia`) que determina cuán parecidos son dos fragmentos de código genético, para lo cual calcula la *distancia de Levenshtein*[1] entre dos cadenas de texto. La función tiene un coste cuadrático con respecto a la longitud de las cadenas que compara. Cabe decir que la longitud de los fragmentos genéticos no es fija, sino que varía dependiendo de cada paciente o donante.

Ejercicio 1: Modifica el programa para que se calcule y muestre el tiempo de ejecución de la llamada a la función `busca_mas_parecidos`. Pruébalo y familiarízate con el código fuente, prestando especial atención a esta función.

[1] `https://es.wikipedia.org/wiki/Distancia_de_Levenshtein`

2. Paralelización

En los programas a desarrollar en los ejercicios siguientes, deberás incluir el código necesario para que se muestre por pantalla el número de hilos con que se ejecuta el programa y el tiempo de ejecución invertido en la función `busca_mas_parecidos`.

Procura que las paralelizaciones que desarrolles se comporten exactamente igual que el programa secuencial. Esto quiere decir que la salida para iguales ficheros de entrada deberá ser idéntica a la producida por el programa secuencial. Presta especial atención a que en caso de donantes a igual distancia de un paciente, el código secuencial devuelve el donante de índice menor. Este comportamiento se consigue en el programa secuencial sin código extra para ello, pero puede requerir algún pequeño añadido en el código de alguna versión paralela.

Todos los tiempos que se muestren en el trabajo deben obtenerse en el clúster `kahan`.

Ejercicio 2: Paraleliza la lectura de datos de ambos ficheros a nivel de tareas (paralelismo de grano grueso). Es decir, modifica el código de forma que, si se ejecuta en varios hilos, se permita que un hilo cargue un fichero mientras al mismo tiempo otro hilo carga el otro fichero.

Ejercicio 3: Partiendo del código del ejercicio anterior, paraleliza el bucle más externo de la función que busca los códigos más parecidos. Para comprobar que funciona correctamente, usa el comando `cmp` o `diff` para comparar el fichero de salida del programa paralelo con el del programa secuencial. Por ejemplo:

```
$ cmp fich1 fich2
```

Si los ficheros son iguales, el comando anterior no mostrará nada.

Ejercicio 4: Partiendo del código del ejercicio 2, paraleliza el bucle interno de la función que busca los códigos más parecidos. Comprueba que funciona correctamente.

Ejercicio 5: Calcula los tiempos de ejecución, speed-up y eficiencia de las 2 versiones paralelas inmediatamente anteriores, ejecutando el programa con 6 hilos. Hazlo para distintas planificaciones, considerando (al menos) las siguientes:

- Planificación estática sin especificar tamaño de *chunk*.
- Planificación estática con tamaño de *chunk* de 1.
- Planificación dinámica con tamaño de *chunk* de 1.

Obtén tablas y gráficas con los resultados, y coméntalos. En particular, analiza la influencia de la planificación, identificando qué planificaciones obtienen mejores prestaciones y a qué puede deberse. Compara el comportamiento de ambas versiones, discutiendo si hay diferencia de prestaciones entre ellas o no y hasta qué punto ese comportamiento era el esperado de acuerdo con lo visto en las clases de teoría.

Ejercicio 6: Obtén tablas y gráficas con tiempos, speed-up y eficiencia que muestren cómo varían las prestaciones en función del número de hilos utilizado. Considera únicamente la versión paralela y planificación que mejor se comporten de acuerdo con el ejercicio anterior. En caso de que no haya diferencia significativa entre las versiones/planificaciones, elige una cualquiera de ellas.

Para limitar el número de ejecuciones, se recomienda usar potencias de 2 para los valores del número de hilos (2, 4, 8...). Llega hasta el número de hilos que consideres adecuado, teniendo en cuenta las características del clúster sobre el que ejecutas.

Comenta los resultados y extrae conclusiones.

Ejercicio 7: Modifica la versión paralela del ejercicio 3 para que el programa calcule y muestre por pantalla la longitud mínima, media y máxima de los fragmentos genéticos de pacientes (`tabla2`) que han tocado a cada hilo, así como del total. Recuerda que en lenguaje C se puede calcular la longitud de una cadena `s` con `strlen(s)`.

Ejemplo de ejecución (con 3 hilos):

```
Hilo 0 de 3: Longitud mín/med/máx: 621/880.7/1106
Hilo 2 de 3: Longitud mín/med/máx: 589/865.3/1090
Hilo 1 de 3: Longitud mín/med/máx: 615/880.6/1135
TOTAL: Longitud mín/med/máx: 589/875.5/1135
```

3. Entrega

Hay **dos tareas** en PoliformaT para la entrega de esta práctica:

- En una de las tareas debes subir un fichero **en formato PDF** con la memoria de la práctica. No se admitirán otros formatos.
- En la otra tarea debes subir un único archivo comprimido con todos los códigos fuentes de los algoritmos que hayas desarrollado. **Únicamente el código fuente, no envíes ejecutables** (ocupan mucho y no son necesarios porque se pueden generar a partir del fuente). El archivo debe estar comprimido en formato `.tgz` o `.zip`.

  Más concretamente, debes entregar los siguientes ficheros de código fuente, que deberán tener los nombres que se indican en la siguiente tabla:

Fichero	**Contenido**
`simil1.c`	Implementación del ejercicio 1
`simil3.c`	Implementación del ejercicio 3
`simil4.c`	Implementación del ejercicio 4
`simil7.c`	Implementación del ejercicio 7

  Si se entrega algún fichero de código fuente adicional, indica claramente en la memoria a qué corresponde el fichero.

  Comprueba que todos los ficheros compilen correctamente.

A la hora de realizar la entrega de la práctica, hay que tener en cuenta las siguientes recomendaciones:

- Hay que entregar una memoria descriptiva de los códigos empleados y los resultados obtenidos.
- Procura que la memoria tenga un tamaño razonable (ni un par de páginas, ni varias decenas).
- No incluyas el código fuente completo de los programas en la memoria. Sí puedes incluir, si así lo deseas, las porciones de código que hayas modificado.
- Pon especial cuidado en preparar una buena memoria del trabajo. Se trata de entregar una memoria. No queremos un libro, pero sí que tenga una estructura y que tenga algo de narrativa y no una mera exposición de resultados.