Ejercicio de paralelización con MPI

Pablo Renero Balgañón, Antonio Román López

Índice

1. Introducción

1.1. Objeto del documento

El objeto del presente documento es mostrar los resultados obtenidos en el proceso de análisis, optimización y paralelización del código secuencial proporcionado en la asignatura con el fin de obtener el mínimo tiempo de cómputo posible.

1.2. Metodología

La metodología seguida para la realización de la práctica ha sido la siguiente:

- Análisis del código secuencial.
- Paralelización con *MPI*
- Optimización de aspectos secuenciales del código.

1.3. Realización

La primera idea de la que partimos consistía en usar un scatter al principio del código secuencial para enviar a los diferentes procesos una división del layer para que hicieran el cómputo normal sobre sus respectivos trozos y finalmente hacer uso de gather para unir todos los trozos de layer de nuevo y a partir de ellos calcular los máximos.

Tras la realización de esta primera versión , analizando el código, comprendimos que el scatter que realizábamos no servía para nada, ya que únicamente enviaba layer que es un vector lleno de 0 y que no es necesario para la creación de los trozos de layer por lo que lo eliminamos.

Continuando este análisis llegamos a la conclusión de que gather tampoco era necesario pues solo necesitábamos conocer el máximo, no los valores del vector al completo y planteamos utilizar un reduce con el uso del tipo predefinido MPI_FLOAT_INT y la función MPI_MAXLOC ademas de guardar los resultados en un struct.

1.4. Problemas posteriores

Después de llevar a cabo la paralelización del programa como se plantea arriba, nos dimos cuenta de que la división que hacíamos en los diferentes trozos para los diferentes procesos solo servía para calcular correctamente el problema si el layer era exactamente divisible entre el numero de procesos pero fallaba si no lo era. Para solucionar esto balanceamos el reparto de cargas haciendo que el layer de los n primeros procesos fuera una unidad más grande siendo n el resto de hacer la división. Para acceder a la posición que en concreto de cada vector, es decir, para llevar a cabo la traducción de la posición global a la posición en su propio proceso calculamos el desplazamiento correspondiente a una posición de un proceso concreto como a partir de:

```
desplazamiento = (rank > layer_size % size) ? rank * (layer_size / size)
+ layer_size % size : rank * (layer_size / size) + rank
```

Tras estos cálculos nos dimos cuenta de que el problema que teníamos entonces se encontraba en la fase de relajación ya que los extremos del vector de un proceso afectaban a los dos colindantes por lo que tuvimos la necesidad de enviarlos entre procesos, en este caso mediante el uso de MPI_Isend y MPI_Irecv. También tuvimos que tener en cuenta un problema parecido en el cálculo de los máximos pero en este caso los solucionamos simplemente calculando el máximo local sin tener en cuenta sus contiguos, por lo que no fue necesaria ninguna comunicación extra.

Después de estos arreglos conseguimos obtener una versión del código funcional y paralelizada que pasó las pruebas del tablón.

2. Optimizaciónes finales

Tras todo el proceso el proceso de paralelización y con una versión ya válida nos dispusimos realizar algunas optimizaciones secuenciales en el código. Las dos optimizaciones principales, a parte del function inlining fueron la realización del cálculo previo de todos los valores de las raíces en vez de calcularlos en cada golpe de bucle, sustituir el bucle de copia de layer_copy por un intercambio de punteros y, como en la anterior práctica, el cálculo de un inicio y fin en los que la energía afectaba para no tener que recorrer toda la longitud del layer en cada impacto. Finalmente y tras todos estos pasos conseguimos una versión del código que se ejecutaba en el leaderboard en un tiempo inferior a 15 segundos.

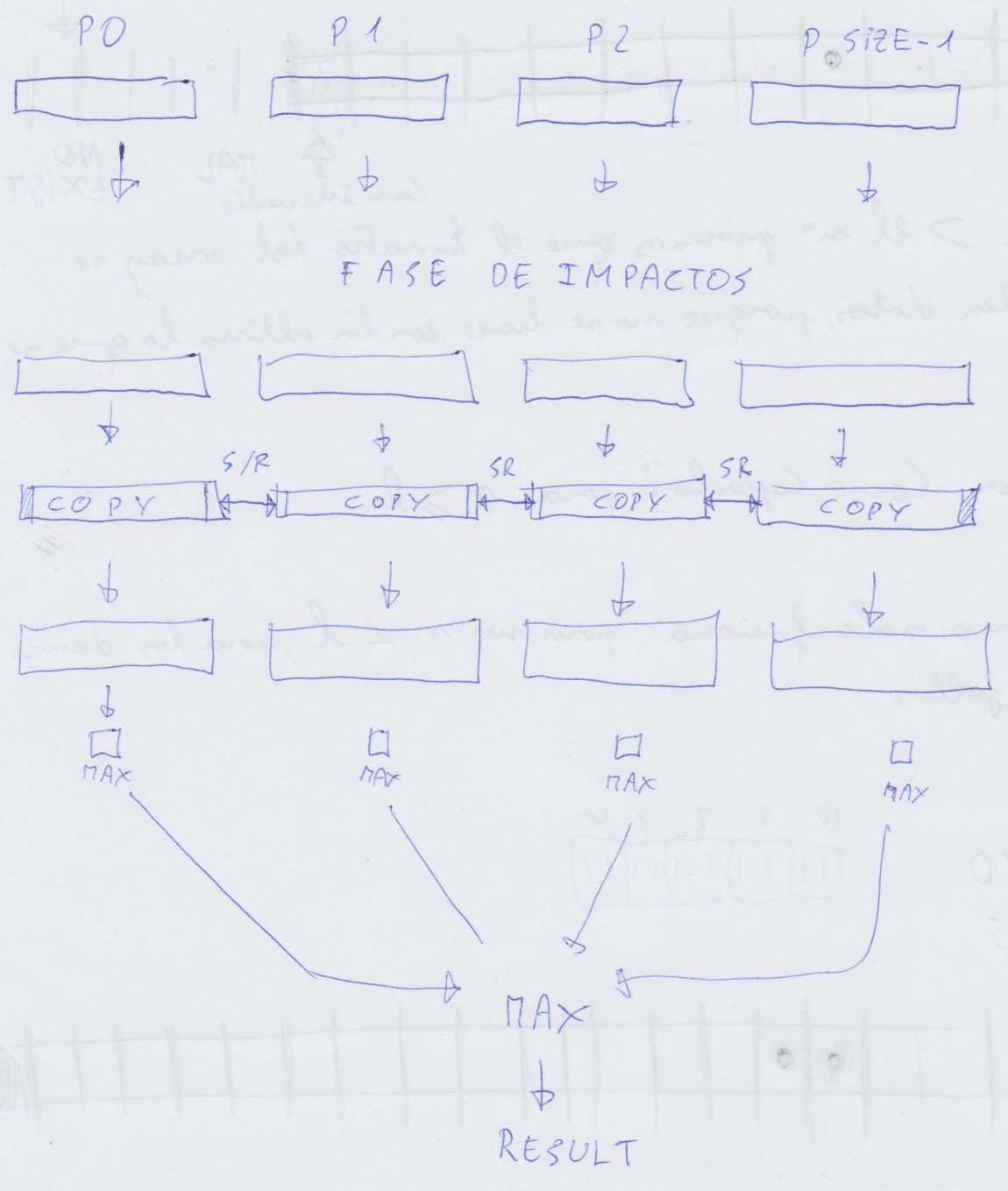

Figura 1: Foto del proceso planteado para el código paralelo.