Javier Falcón 2016-5265
Manuel Molina 2016-5468

Resolución del problema

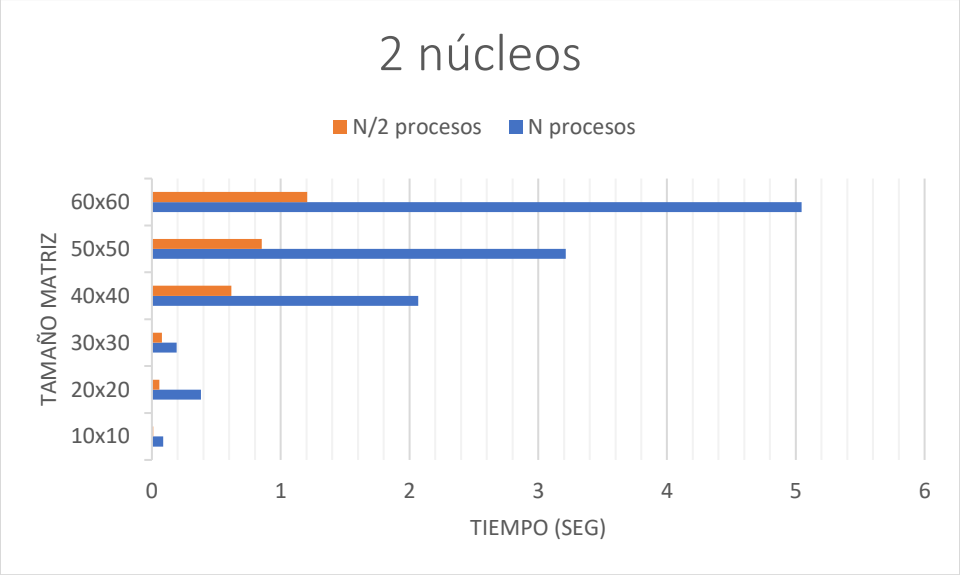
Al ser nuestra primera asignación utilizando programación paralela por paso de mensajes, nos encontramos con un reto más difícil de lo que esperábamos. Los objetivos eran claros: dado un número de procesos a utilizar, se debía realizar un programa que hiciera una correcta repartición de las tareas relacionadas a la multiplicación de matrices entre la cantidad de procesos indicada. Para esto, el *proceso maestro* debía encargarse de calcular las particiones y enviarla a cada *proceso hijo*. Es justo en este momento cuando comenzamos a tomar el tiempo. Las particiones las realizamos de la siguiente manera: el tamaño de cada matriz dividido entre el número total de procesos (sin contar el *maestro*) nos daría la cantidad de filas a multiplicar por cada proceso *hijo*. Acto seguido, el proceso maestro envía las secciones de la matriz que le corresponde computar a cada proceso *hijo*. Cabe destacar que en este tipo de programación paralela no se utiliza el concepto de memoria compartida, por lo que cada *hijo* recibe una copia del programa con su memoria particular. Dicho esto, una vez computadas las multiplicaciones, los hijos proceden a mandarle sus resultados al proceso *maestro*, el cual se encontraba a la espera de dichas computaciones. Finalmente, se unen todos los mensajes recibidos y, con la matriz resultado lista, de deja de tomar el tiempo. De esta manera, obtendremos el tiempo de ejecución para multiplicar matrices utilizando paralelismo por paso de mensajes.

Javier Falcón 2016-5265
Manuel Molina 2016-5468

Comparación de resultados

Para la medición y prueba del programa, el código fue ejecutado en dos máquinas con procesadores diferentes, una de 2 *cores* y la otra de 6 *cores*. Para cada una de las pruebas se ejecutó una matriz de tamaño superior a la anterior, comenzando en una matriz 10x10 y aumentando de 10 en 10 su tamaño. Consideramos interesante computar cada matriz con un número de procesos distintos, con el objetivo de observar los efectos de la granularidad del programa. Primero se utilizaron N procesos, donde N viene dado por el tamaño de la matriz y, finalmente, probamos con N/2 procesos. Los resultados fueron los siguientes:

- **Máquina de 2 cores:**

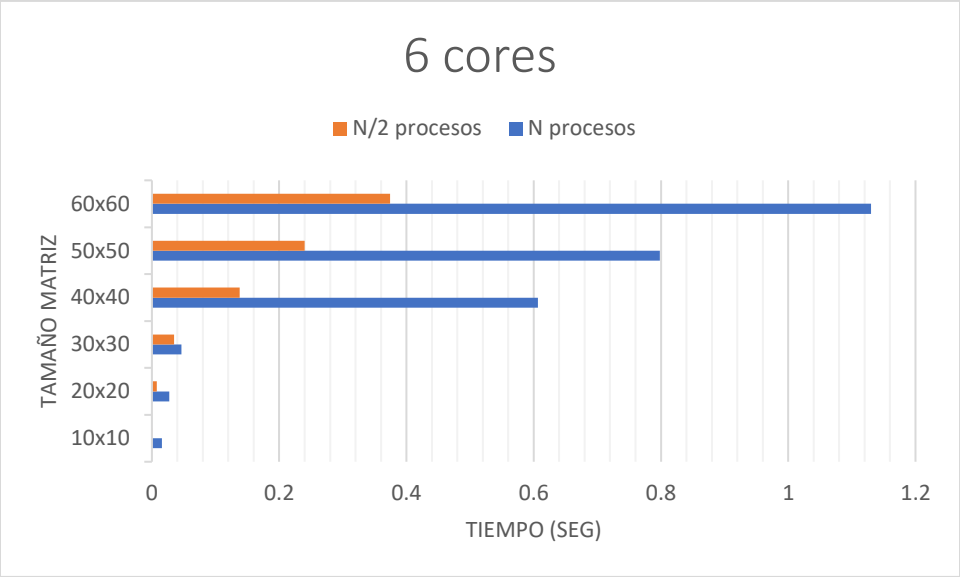


Gráfica 1 Tiempos de ejecución en una máquina de 2 cores

En la **Gráfica 1** podemos observar el comportamiento de un algoritmo de multiplicación de matrices paralelo por paso de mensajes. En interesante notar cómo los tiempos de ejecución en N procesos es muy superior al tiempo de ejecución utilizando N/2 procesos. Esta diferencia es causada por los efectos de la granularidad. Como N procesos representan una menor granularidad, implica que existirá una necesidad mayor de

comunicación entre procesos que cuando se tiene una mayor granularidad, como el caso de N/2 procesos.

- **Máquina de 6 cores:**



Gráfica 2 Tiempos de ejecución en una máquina de 6 cores

La **Gráfica 2** nos arroja un comportamiento similar a la **Gráfica 1,** una mayor cantidad de procesos implica más comunicación y, por lo tanto, más tiempo de ejecución. Dicho esto, lo interesante de este resultado es poder apreciar cómo afecta la disponibilidad de núcleos de procesamiento en el tiempo de ejecución. La multiplicación para las mismas matrices del mismo tamaño fue mucho más rápida en una computadora de 6 núcleos que en la computadora de 2 núcleos. Esto es gracias a que tendremos un mejor *speed-up* de procesamiento ya que más procesos pueden ejecutarse al mismo tiempo cuando se tiene una cantidad mayor de procesadores.

Finalmente, a modo de conclusión, a pesar de la dificultad para implementar el código utilizando el API de MPI, resultó interesante observar dos aspectos:

1. Confirmamos que una menor granularidad nos arroja tiempos de ejecución más altos debido a que cada la computación se limita a una mayor comunicación en lugar de a

una mayor computación, lo cual indica que es importante encontrar un balance en cuanto a la cantidad de procesos que se abren para computar.

2. Observamos cómo un mayor número de elementos de procesamiento nos ofrece un tiempo de ejecución mucho más rápido, confirmando los conceptos teóricos vistos en clase.