Universidad de Costa Rica

Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI0117 – PROGRAMACIÓN PARALELA Y CONCURRENTE
Profesor Jeisson Hidalgo

Tarea 04 - OMP y MPI

Estudiante:
Emmanuel D. Solís - B97670
emmanuel.solispomares@ucr.ac.cr

28 de julio de 2021

Índice

I - S - 2021 - RRF - Programación Paralela y Concurrente - 002

Tarea04: goldbach_omp_mpi

En las tareas anteriores usted escribió programas que reciben números enteros en la entrada estándar y calculan las sumas de Goldbach de forma serial (tarea01), concurrente con Pthreads (tarea02), y comparó su rendimiento (tarea03). Estas soluciones aprovechan los recursos concurrentes de una máquina, pero no escalan, ni aprovechan varias computadoras disponibles en una organización o un clúster.

En esta tarea su objetivo es distribuir el cálculo de sumas de Goldbach entre varias máquinas usando dos tecnologías: OpenMP y MPI. Su programa recibirá los números en la entrada estándar en el mismo formato, y deberá producir los resultados en el mismo orden, de tal forma que pase los casos de prueba.

Concurrencia declarativa [30%]

Utilice la tecnología de paralelismo de datos OpenMP para paralelizar el cálculo de sumas. Dadas las facilidades de esta tecnología, utilice una unidad de descomposición más granular que los números. Asegúrese de que su implementación pasa los casos de prueba, y no genera diagnósticos del linter.

Nota: Si toma como código base la tarea02 o tarea03, deberá reemplazar el código de Pthreads por OpenMP. Si usa como código base su solución serial (tarea01), asegúrese de aplicarle las optimizaciones seriales que realizó en tareas posteriores y de que éstas pasen los casos de prueba.

Comparación Pthreads-OpenMP [20%]

Como hubo un cambio en la implementación de la concurrencia, compare el rendimiento de su implementación con OpenMP contra la de Pthreads siguiendo el mismo procedimiento que aplicó en la tarea03, utilizando tantos hilos como núcleos de CPU hay en el sistema con el caso de prueba `023`. Calcule el Speedup y la eficiencia. Cree un gráfico combinado con esas dos secuencias. Agréguelo a un documento de análisis de resultados donde incluya una discusión no mayor a 500 palabras.

Distribución [30%]

Para hacer su solución escalable y aprovechar las computadoras de un clúster, distribuya el cálculo de sumas de Goldbach utilizando la tecnología MPI. Tome en cuenta que su programa debe leer los datos de la entrada estándar y no de archivos ni argumentos de línea de comandos. Descomponga la unidad de trabajo a distribuir, ésta puede tener mayor granularidad que la unidad usada para OpenMP. Idee un mapeo eficiente entre las unidades de trabajo y los procesos, de tal forma que los recursos del clúster se aprovechen lo más equitativamente posible. Asegúrese de que su solución distribuida con MPI pase los casos de prueba usando todos los núcleos con OpenMP, y que no genere diagnósticos del linter.

Comparación OpenMP-MPI [20%]

Compare el rendimiento de su solución distribuida usando el caso de prueba `023` en el clúster de Arenal. Cree un proceso por cada nodo secundario del clúster, el cual cree tantos hilos de ejecución como núcleos de CPU hay disponibles en la máquina donde corre.

Mida la duración de la versión distribuida, calcule el incremento de velocidad

y la eficiencia. Agregue los resultados a su gráfico, de tal forma que en el eje-x se pueda comparar la versión concurrente con Pthreads, con OpenMP, y finalmente la distribuida con MPI/OpenMP. Agregue a su discusión un análisis de los resultados que obtuvo con la versión distribuida.

1. Paralelismo usando Open MP

Por medio de un paralelismo implementado al descomponer el calculo de las sumas de cada número es que se implementará la tecnología de *Open MP*, es decir que el calculo de las sumas de Goldbach de cada números, lo que vendrían siendo los ciclos for en el código, serán las partes que llevarán código de *Open MP*.

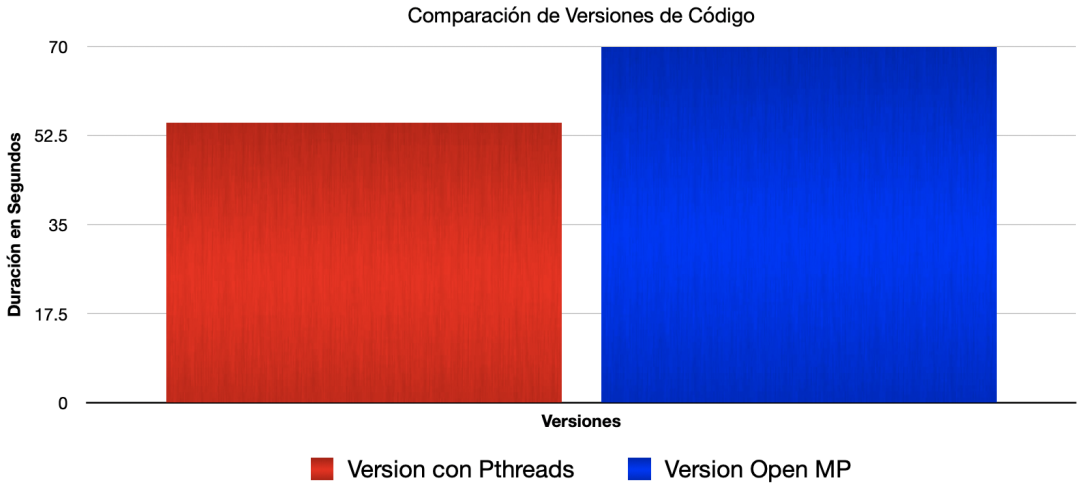
2. Distribución de tareas con MPI

Si bien se han mejorado los tiempos con la implementación del paralelismo de tareas con Open MP aún se podría mejorar usando los recursos disponibles en un sistema como el cluster donde tenemos varios nodos con recursos que podemos aprovechar por medio de distintos procesos, es por ello que esta implementación se realizó distribuyendo los números solicitados por distintos procesos de la aplicación.

3. Analisis de resultados

3.1. OpenMP contra Pthreads

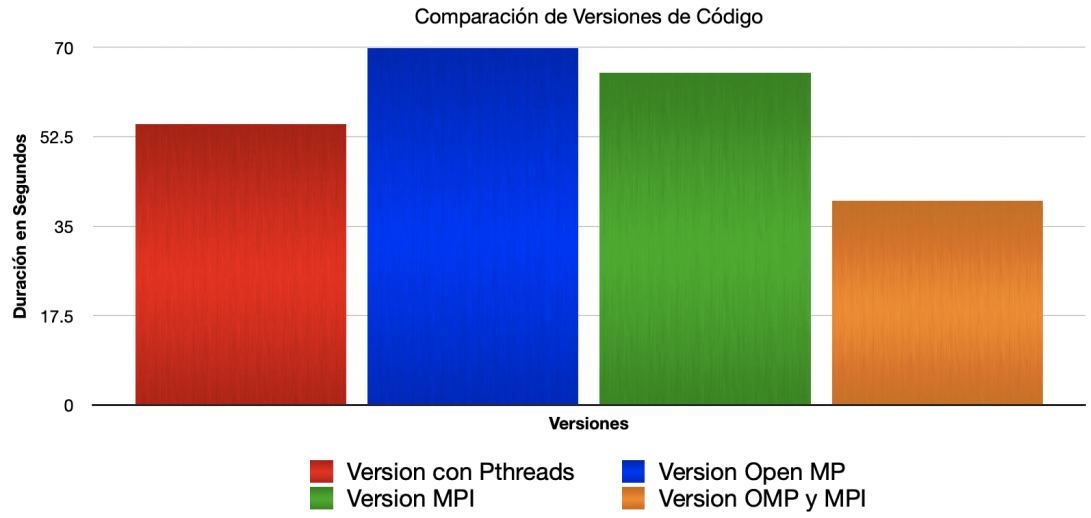
Como base fundamental mencionamos que la tecnología de OpenMP por debajo lo que utiliza son pthreads sin embargo de una forma más generica para que se pueda usar una tecnología declarativa, es por ello que las posibilidades que ofrecen los pthreads no son aprovechadas en su máxima capacidad. Dicho eso podemos ver el siguiente grafico donde hay una comparación entre ambas implementaciones.

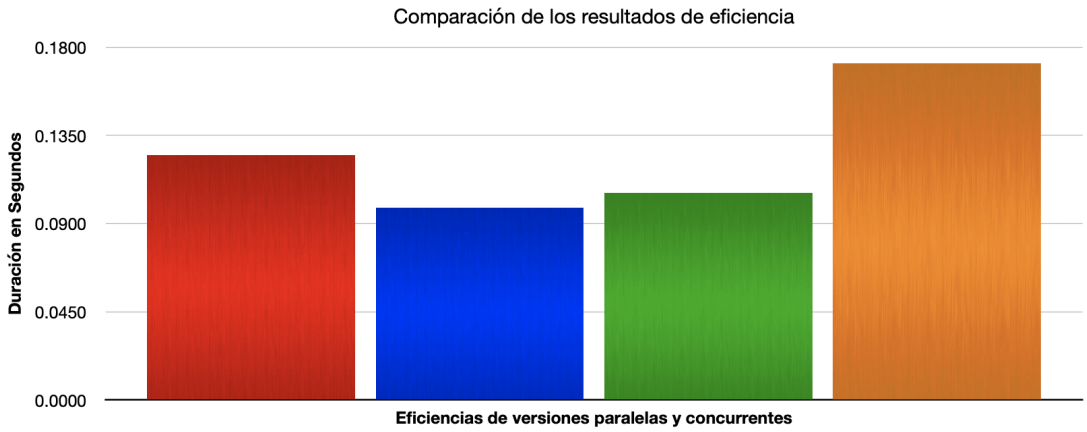


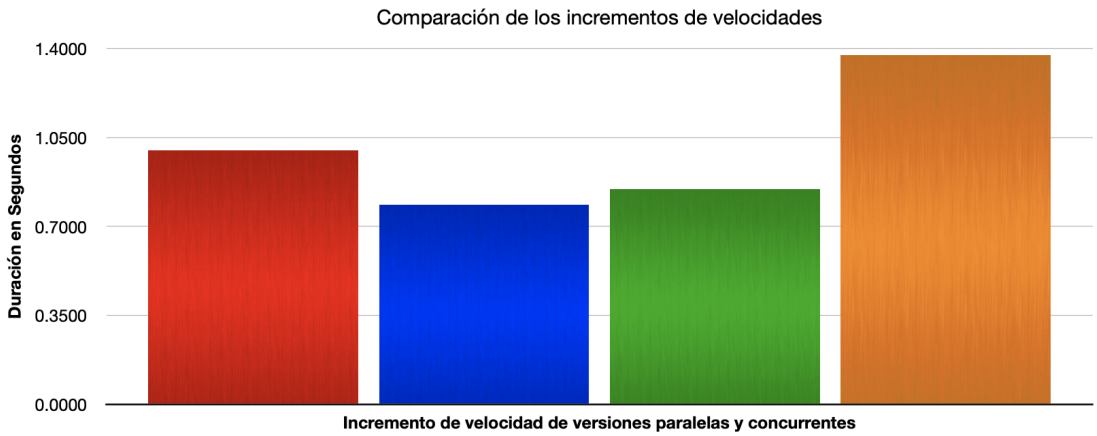
3.2. OpenMP contra Hibrido

Una vez ya implementada la solución de Open MP podemos agregarle la herramienta extra de MPI que nos permite distribuir los cálculos de varios números. Esta es una optmización importante porque distribuye mucho mejor el uso de recursos y disminuye los tiempos de espera significativamente. Podemos ver los resultados de los tiempos de ejecución en el siguiente gráfico:

Como resultado final de los calculos de incremento de velocidad y la eficiencia tenemos lo siguiente:


Comparación de Versiones de Código
70
52.5
35
17.5
0
Duración en Segundos
Versiones
Version con Pthreads
Version Open MP
Version MPI
Version OMP y MPI


Comparación de los resultados de eficiencia
0.1800
0.1350
0.0900
0.0450
0.0000
Duración en Segundos
Eficiencias de versiones paralelas y concurrentes


Comparación de los incrementos de velocidades
1.4000
1.0500
0.7000
0.3500
0.0000
Duración en Segundos
Incremento de velocidad de versiones paralelas y concurrentes

4. Conclusiones

Era de esperarse que el uso de pthreads presentara un mejor desempeño contra OpenMP pues nosotros teniamos mayor control sobre cómo deseabamos utilizar las cosas y sobre cómo implementarlas. Por tal razón este fue mejor.

Respecto a las comparaciones de las cuatro versiones y basados en el párrafo anterior podemos saber que MPI nos permite tener una mejor distribución de las tareas pero que tan bien se ejecuten cada una de estas dependera de nuestro código, y que tan bien implementado se encuentre. Aunque Pthreads implementado manualmente es mejor que OpenMP, obtuvo mejores resultados la versión híbrida debido a que aprovechaba el paralelismo de datos y la distribución de tareas, usando dos tecnologías de paralelismo importantes.