Computación Paralela (MC-8836)

Instituto Tecnológico de Costa Rica
Escuela de Computación
I semestre, 2022
Tarea Programada 4

Instrucciones[1]

- Debe crear una cuenta en `https://gitlab.com` y crear un repositorio **privado** llamado `mc-8836` para las tareas del curso. Si no lo he hecho ya, envíe un correo con su usuario de `GitLab` al profesor `esteban.meneses@acm.org` y al asistente `alexsaenz@estudiantec.cr`.
- Asegúrese que los usuarios `emeneses` (Esteban Meneses) y `sr.alexe` (Álex Sáenz) existen como colaboradores de su repositorio GIT en `GitLab` con permisos de acceso al menos de *Reporter*.
- Los fuentes para esta tarea están en el directorio `hw4` del repositorio GIT del profesor en `GitLab`:
  `git clone https://gitlab.com/emeneses/mc-8836.git`
- Las interfaces funcionales del código fuente provisto deben respetarse.
- El único mecanismo de entrega de esta asignación es por medio de su repositorio GIT en `GitLab`.
- Debe crear una carpeta `hw4` en su repositorio GIT. Ese directorio debe contener los siguientes archivos:
  • Archivo `jacobi.cpp` con el código que implementa el algoritmo paralelo de interacción de partículas.
  • Archivo `jacobi_sequential.cpp` con el código secuencial de interacción de partículas.
  • Archivo `report.pdf` con un reporte resumiendo los resultados del programa (debe ser un archivo en formato PDF).
- El código paralelo debe correr en la supercomputadora `Kabré` del Centro Nacional de Alta Tecnología (CeNAT). Para poder compilar programas MPI debe ejecutar el comando:
  `module load gcc/7.2.0`
  `module load mpich/3.2.1-gcc-7.2.0`
  También puede copiar estos comandos en el archivo `~/.bashrc` y se ejecutarán automáticamente.
- Se recomienda utilizar trabajos interactivos para el desarrollo de la tarea:
  `salloc --partition=nu --time=02:00:00`
- Se debe utilizar trabajos en lote para recoger información de desempeño:
  `sbatch file.slurm`
- La fecha límite de entrega de esta asignación es el **Viernes 6 de mayo a las 11:55pm**. Entregas extemporáneas **no** serán aceptadas.
- La presente asignación es estrictamente **individual**.

[1]Cada estudiante debe mostrar el estándar más alto de integridad académica. Presentar las ideas de otra persona sin el debido permiso es una clara violación de la política de integridad académica de la universidad y es una práctica que debilita las habilidades para resolver problemas en cualquier ámbito. Cualquier infracción a esta regla será llevada hasta sus últimas consecuencias.

Método de Jacobi

El método de Jacobi en análisis numérico es un método iterativo usado para resolver sistemas de ecuaciones lineales del tipo $Ax = b$. El nombre del algoritmo se debe al matemático alemán Carl Gustav Jakob Jacobi (1804-1851). La base del método consiste en construir una sucesión convergente definida iterativamente. El límite de esta sucesión es precisamente la solución del sistema. Para efectos prácticos, si el algoritmo se detiene después de un número finito de pasos se llega a una aproximación del valor de x de la solución del sistema. Tenemos entonces que:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

para $i = 1, 2, 3, ...$ y donde k es el contador de iteración[2].

El método de Jacobi se puede implementar como un *iterative stencil loop* (ISL, por sus siglas en inglés). Un ISL es una clase de mecanismos para la solución numérica de procesamiento de datos que actualiza los elementos de un arreglo de acuerdo a un patrón fijo, llamado *stencil*. Estos patrones se encuentran principalmente en simulaciones computacionales, por ejemplo en dinámica de fluido computacional en el contexto de aplicaciones científicas y de ingeniería. La Figura 2(a) presenta un stencil de siete puntos en una malla tridimensional regular[3].

Una de las aplicaciones del método de Jacobi, implementado como un ISL, es la simulación de transferencia de calor. La Figura 1 muestra una visualización[4] de la transferencia de calor de una placa caliente en la base hacia una colección de placas alineadas perpendicularmente sobre la base.

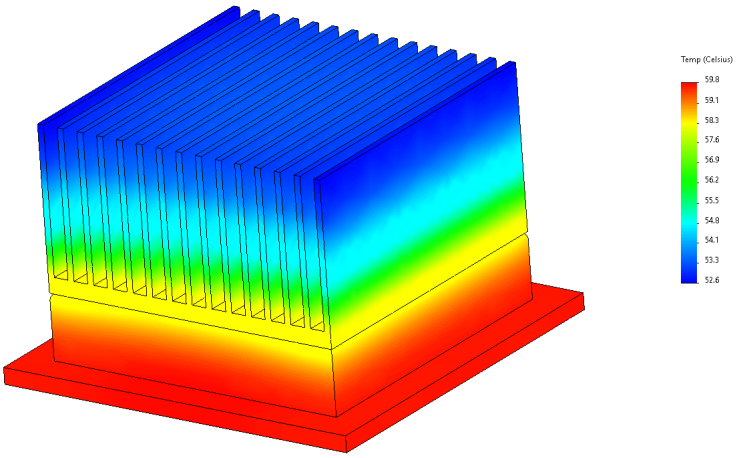
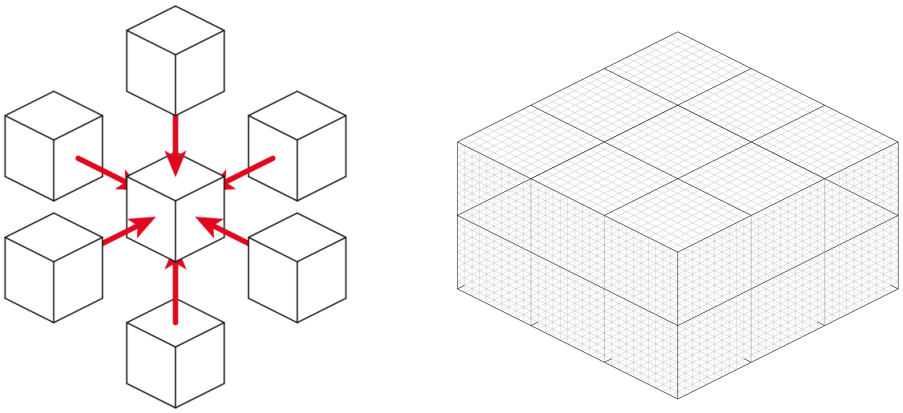


Figura 1: Transferencia de calor en un espacio tridimensional.

La simulación de esta asignación simula la transferencia de calor en una malla regular tridimensional usando el paradigma de paso de mensajes. La malla está ilustrada en la Figura 2(b). Cada celda contiene una valor de temperatura, que será inicializada de forma arbitraria (pero fija en el código). Las celdas están agrupadas en bloques más grandes (cada uno llamado *chunk* en el código). Cada bloque se asocia a un único *rank* de MPI. Todos los bloques tienen la misma dimensión. La comunicación que existe entre los ranks es para intercambiar las celdas que están en la frontera entre bloques. Cada rank tiene entonces seis vecinos, dos por cada dimensión. Esto quiere decir que si los bloques tienen cada uno dimensiones $D_x D_y D_z$, entonces

[2]Tomado de Wikipedia
[3]Tomado de Wikipedia
[4]Tomado de Solidworks

un rank deberá intercambiar las dos caras exteriores de su bloque de dimensión $D_x D_y$ con sus vecinos en el eje z. Algo equivalente sucederá con los restantes vecinos en las otras dimensiones. De esta forma, en cada iteración un bloque envía seis mensajes a sus vecinos. Los bloques están continuos en el espacio, lo que quiere decir que un bloque en un extremo está conectado con el bloque en el otro extremo de la misma dimensión.



(a) *Stencil* de siete puntos (b) Espacio de simulación

Figura 2: Algoritmo de simulación de transferencia de calor.

El algoritmo es iterativo y cada iteración de la simulación está compuesta de los siguientes pasos:

1. Cada rank declara las seis recepciones de los mensajes que sus vecinos le enviarán (o sea, se utiliza la función `MPI_Irecv`).
2. Cada rank envía los mensajes a sus seis vecinos.
3. Cada rank ejecuta el código de Jacobi sobre su bloque de celdas.
4. Se ejecuta una operación colectiva para determinar el error mayor entre todos los ranks.
5. Se ejecuta una operación colectiva para determinar el tiempo de iteración promedio.

El programa debe tomar por lo menos tres parámetros en la línea de comandos:
`mpiexec -np <P> -bind-to core ./jacobi <NX> <NY> <NZ> [<iteraciones>]`
donde los valores `NX`, `NY` y `NZ` representan el número de bloques en todo el sistema en las dimensiones respectivas. La multiplicación de estos valores debe ser siempre igual al número total de ranks `P`.

Deberá tomar como base su algoritmo paralelo e implementar una versión secuencial del mismo usando la misma interfaz funcional (deberá recibir los mismos parámetros en la línea de comandos que la versión paralela). La implementación interna del algoritmo queda a discreción del estudiante.

Informe

Debe crear un informe con las siguientes secciones:

1. Una estrategia general del esfuerzo de paralelización. ¿Qué patrones de programación paralela usó en la solución? ¿Qué operaciones de MPI parecieron las más apropiadas para el programa?
2. Análisis de *speedup*. Presente un gráfico con el *speedup* de su solución paralela usando 2, 4, 8, 16, 32, 64, 128 y 256 *ranks*. Cuando ejecute en más de 64 ranks asegúrese de usar múltiples nodos. Este

gráfico debe tener el número de *ranks* en el eje `x` y el *speedup* en el eje `y`. Use el tiempo de ejecución del programa secuencial para calcular el *speedup*. Debe escoger un tamaño de problema *interesante* para su análisis[5].

3. Análisis de eficiencia. Presente un gráfico con la eficiencia de su solución paralela usando 2, 4, 8, 16, 32, 64, 128 y 256 *ranks*. Cuando ejecute en más de 64 ranks asegúrese de usar múltiples nodos. Este gráfico debe tener el número de *ranks* en el eje `x` y la eficiencia en el eje `y`. Use el tiempo de ejecución del programa secuencial para calcular la eficiencia. Debe escoger un tamaño de problema *interesante* para su análisis.

Calificación

Se evaluará la correctitud, claridad y eficiencia del código escrito, así como el apego a las directrices establecidas. La división de la evaluación es la siguiente:

- Programación método de Jacobi: 70 %
- Informe con resultados: 30 %

[5]Posiblemente quiera modificar los valores `DIMX,DIMY,DIMZ` en el código para tener tamaños de problema manejables.