Resolent l'equació 2D Laplace utilitzant el mètode iteratiu de Jacobi

1 Introducció

L'equació 2D de Laplace es important en diversos camps, especialment als camps de l'electromagnetisme, l'astronomia i la dinàmica de fluids (com es el cas de la propagació de calor).

$$\nabla^2\Phi(x,y) = \frac{\partial^2\Phi(x,y)}{\partial x} + \frac{\partial^2\Phi(x,y)}{\partial y} = 0$$

L'equació diferencial parcial, o EDP, es pot discretitzar, i aquesta formulació es pot utilitzar per aproximar la solució mitjançant diversos tipus de mètodes numèrics.

$$\frac{\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j}}{h^2} + \frac{\Phi_{i,j+1} - 2\Phi_{i,j} + \Phi_{i,j-1}}{h^2} \approx 0$$

$$\Phi_{i,j} \approx \frac{1}{4}[\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1}]$$

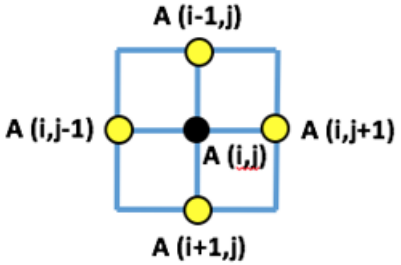

Figure 1: Punts a tenir en compte per al càlcul de l'stencil.

El mètode **iteratiu de Jacobi** és una manera de resoldre l'equació 2D-Laplace. Els mètodes iteratius són una tècnica comuna per aproximar la solució

de EDPs el·líptics, com l'equació 2D-Laplace, dins d'alguna tolerància permesa. En el cas del nostre exemple, realitzarem un simple calcul d'**stencil** on cada punt calcula el seu valor com la mitjana dels valors dels seus veïns. L'stencil està compost pel punt central i els quatre veïns directes. El càlcul s'itera fins que el canvi màxim de valor entre dues iteracions cau per sota d'algun nivell de tolerància o s'arriba a un nombre màxim d'iteracions.

$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

2 Descipció del codi

Suposarem una matriu 2D d'entrada, A, amb dimensions fixes **n** i **m**, definida com una variable global (fora de qualsevol funció). Les dades inicials determinen l'estat inicial del sistema. Assumim que tots els punts interiors de la matriu 2D són zero, mentre que l'estat límit, que és **fix** al llarg del procés d'iteració de Jacobi, es defineix de la següent manera:

```
// limit per a fronteres amunt i avall
For each j from 0 to m:
```
$$A_{0,j} = A_{n-1,j} = 0;$$

```
// frontera a dreta i esquerra
For each i from 0 to n:
```
$$A_{i,0} = sin(i * \pi/(n-1)) \;\; A_{i,m-1} = sin(i * \pi/(n-1) * e^{-\pi};$$

El bucle més extern que controla el procés d'iteració s'anomena **bucle de convergència**. Aquest bucle itera fins que la resposta ha convergit, assolint una tolerància d'error màxima o un nombre d'iteracions. Tingueu en compte que si es produeix o no una iteració de bucle depèn del valor d'error de la iteració anterior. A més, els valors de cada element d'A es calculen a partir dels valors de la iteració anterior.

El primer bucle niat dins del bucle de convergència hauria de calcular el nou valor per a cada element en funció dels valors actuals dels seus veïns. Observeu que és necessari emmagatzemar aquest nou valor en una matriu diferent, o matriu auxiliar, que anomenem **Anew**. Si cada iteració emmagatzema el nou valor en si mateix, llavors existeix una *dependència de dades* entre els elements, ja que l'ordre en que cada element es calcula afecta la resposta final. Al emmagatzemar en una matriu temporal o auxiliar (Anew) ens assegurem que tots els valors es calculen **utilitzant l'estat actual de A abans s'actualitzi**. Com a resultat, *cada iteració del bucle és completament independent una de l'altra.* Aquestes iteracions de bucle es poden executar amb seguretat en qualsevol ordre i el resultat final serà el mateix.

Un segon bucle ha de calcular un valor d'error màxim entre els errors de cadascun dels punts de la matriu 2D. El valor d'error de cada punt de la matriu 2D es defineix com l'arrel quadrada de la diferència entre el nou valor (en Anew) i l'antic (en A). Si la quantitat màxima de canvi entre dues iteracions està dins d'alguna tolerància, el problema es considera convergent i el bucle exterior terminarà.

El tercer bucle aniat fa una actualització dels valors d'A amb els valors calculats a Anew. Si és l'última iteració del bucle de convergència, A tindrà els valors finals. Si el problema encara no ha convergit, llavors els valors d'A serviran com a dades d'entrada de la següent iteració.

Per últim, cada deu iteracions del bucle convergència s'ha d'imprimir l'error real a la pantalla per comprovar que l'execució avança correctament i l'error convergeix.

3 Recomanacions per a la solució amb MPI

Com sabeu, de vegades, un problema pot ser massa gran (en mida o complexitat) per abordar-lo en un sol node. En aquest cas, podem desenvolupar una solució distribuïda paral·lela que faci ús de múltiples nodes per resoldre el problema en un temps raonable. En aquesta pràctica, ens centrarem en la paral·lelització del problema de Laplace utilitzant MPI com a mecanisme de comunicació i sincronització.

Teniu un programa per resoldre l'equació de Laplace disponible en versió seqüencial. Suposeu que utilitzarem N processos per resoldre el problema de forma distribuida. D'acord amb les caràcteristiques del problema, cadascun d'aquests processos haurà de dur a terme una part del cómput sobre una porció de les dades, tenint en compte els intercanvis necessaris de dades i la sincronització entre processos. Els processos que hauràn de comunicar-se depenen de com es reparteixen les dades entre ells.

L'ús de memòria per a cada procés es pot reduir proporcionalment al nombre de processos amb l'addició d'algunes files extres en cada procés per emmagatzemar les dades (files de frontera) rebudes dels seus veïns. Si l'algorisme seqüencial triga un temps T a completar-se, podem esperar que el temps d'aquesta versió distribuïda podria ser T/N + el tempsd de sobrecàrrega de comunicació. Val la pena assenyalar que la quantitat addicional de memòria utilitzada i la sobrecàrrega de comunicació augmentarien el seu impacte en el rendiment de les aplicacions a mesura que augmenta el nombre de processos utilitzats.

L'avaluació del rendiment d'una aplicació paral·lela normalment consisteix a analitzar l'escalabilitat de l'aplicació (canviant la quantitat de dades a processar o el nombre de recursos utilitzats) i explicar les causes dels resultats observats. Això vol dir que hauríeu d'executar el programa utilitzant diferents mides d'entrada (diferent mida de matriu o diferents proves) i variant el nombre de recursos i calcular l'acceleració i l'eficiència (**escalabilitat forta**) i fer el mateix variant tant la mida d'entrada com el nombre de recursos (**escalabilitat feble**). També podeu intentar quantificar la sobrecàrrega de comunicació, la

relació comunicació/computació o les funcions que consumeixen més temps, per tal d'explicar el comportament de l'aplicació.

Feu una anàlisi de rendiment del vostre programa utilitzant les eines d'anàlisi de rendiment disponibles. **A TENIR EN COMPTE:** utilitzar perf en una aplicació MPI no té sentit (ja que només ens proporciona les dades d'un procés), utilitzeu MPI_Wtime.

4 Entorn de proves i treball experimental

Recordeu que heu de fer les execucions utilitzant el gestor de cues d'SLURM. Podeu preparar dos scripts d'SLURM.

- Per testar els resultats - compilació i execució seqüencial per a unes dades d'entrada petites (aquesta versió us donarà els resultats correctes); compilació i execució paral·lela per a unes dades d'entrada petites per comprovar que la versió dona els resultats correctes.
- Per a l'experimentació final i recollir els temps d'execució - compilació i execució seqüencial (per fer servir com a referència del temps seqüencial); compilació i execució paral·lela: diferent nombre de processos i diferents tamanys per a les dades d'entrada per analitzar el rendiment.

IMPORTANT: podeu utilitzar els dos clústers, aolins i Wilma per provar i executar. Recordeu que heu de fer les execucions mitjançant el gestor de treballs SLURM.