Université de Pau et des Pays de l'Adour
Département de Mathématiques
2020-2021

M2-BigData : GPGPU
Chapter 13 – Exercice 3

Objectives

Write a more complex code and parallelize with OpenACC.

In this part we want to solve the 2D-Laplace equation with the iterative Jacobi solver. Iterative methods are a common technique to approximate the solution of elliptic PDEs, like the 2D-Laplace equation, within some allowable tolerance. In the case of our example we will perform a simple stencil calculation where each point calculates it value as the mean of its neighbors' values. The calculation will continue to iterate until either the maximum change in value between two iterations drops below some tolerance level or a maximum number of iterations is reached.

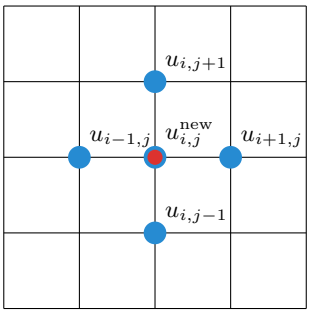
$$\begin{cases} \dfrac{du}{dt} = a\Delta u \text{ on } \Omega \times [0;T] \\ u(0,y,t) = 0 \\ u(1,y,t) = 0 \\ u(x,0,t) = 0 \\ u(x,1,t) = \sin(\pi x) \\ u(x,y,0) = u_0; \end{cases}$$

The equation models the heat distribution across a squared metal plate. We decompose the domain $\Omega = [0;1]^2$ as a cartesian grid (as an image with pixels). The temperature u is computed on each point of the grid (or pixels) using 4 neighboring points (from math finite difference method). The entire image is updated several times until reaching a final time T.

The formula is therefore for a current time $k\delta t$:

$$u_{i,j}^{\text{new}} = u_{i,j} + a\delta t \frac{u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i,j+1} + u_{i,j-1}}{h^2}$$

where u^{new} represents the solution a time $(k+1)\delta t$ computed from u, the solution at time $k\delta t$.



One need a specific handle for edges, for instance when $i = 0$, we need to compute $u_{0,j}$ but we do not have any $u_{-1,j}$ to use. Therefore values are computed only for $1 \leq i \leq N_x - 1$ and $1 \leq j \leq N_y - 1$. Values at boundary ($i = 0$, $j = 0$, $i = N_x$ and $j = N_y$) are directly given by values

from initialization. So the overall algorithm is the following :

Algorithme 1 : Iterative Jacobi solver
Données : 2D mesh
Résultat : u as approximation of the solution of the heat equation

```
1  //Initialize u^old ;
2  u^old ← u(x, y, 0) ;
3  pour k = 1 à N faire
4  |  // Computes new temperature field ;
5  |  pour i = 1 à N_x − 1 faire
6  |  |  pour j = 1 à N_y − 1 faire
7  |  |  |  u^new_{i,j} = u_{i,j} + aδt(u_{i+1,j} + u_{i−1,j} − 4u_{i,j} + u_{i,j+1} + u_{i,j−1})/h^2;
8  |  |  fin
9  |  fin
10 |  // Exhange old and new temperature field for next iteration ;
11 |  u ← u^new;
12 fin
```

Instructions

The given code works with two execution mode :

1. testing and debuging : `./3-heatEquation.exe` without any parameters
2. profiling and performance measurements : `./3-heatEquation.exe 1000 1000 100000 10000`

— From the given code, make all the changes and optimizations to perform the computations using OpenACC.
— Explain all your choices and justify all your OpenACC directives. Justifications can be made regarding profiling informations.
— Profile your last version and analyse what is the remaining bottleneck of the performances.