Aashish Dhungana

Suyog Joshi

Date: 12/13/2020

Homework 6

CS 471

Introduction

In this assignment we build a program that uses shared memory to differentiate and integrate. We modified our code from assignment 4 using OpenMP constructs to parallelize our serial code. To make sure our program compiles in serial mode we preform optimum parallelization in non-intrusive way. In this report we discussed briefly about our computation used in assignment 4 in 'Part I' and examine into more detail on our approach to assignment 6 in 'Part II'.

Part I

i) Approximating Differentiation and Integrals

To approximate differentiation; functions $x = x(r, s)$ and $y = y(r, s)$ are expressed as functions of r and s, where $\{(r, s) \in [-1, 1]^2\}$ and the chain rule is used as follows.

$$u_x = u_r r_x + u_s s_x$$

$$u_y = u_r r_y + u_s s_y$$

Given smooth mapping $(x, y) = (x(r, s), y(r, s))$, we use the reference element to compute

$$\frac{\partial \mathrm{u}}{\partial \mathrm{y}} = \frac{\partial \mathrm{r}}{\partial \mathrm{x}}\frac{\partial \mathrm{u}}{\partial \mathrm{r}} + \frac{\partial \mathrm{s}}{\partial \mathrm{y}}\frac{\partial \mathrm{u}}{\partial \mathrm{s}}$$

Gives the formula:

$$\begin{bmatrix} \mathrm{r_x} & \mathrm{r_y} \\ \mathrm{s_x} & \mathrm{s}_y \end{bmatrix} \begin{bmatrix} \mathrm{x_r} & \mathrm{x_s} \\ \mathrm{y_r} & \mathrm{y_s} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Taking the double integral over our domain of our reference element as a function of r and s:

$$\int_\Omega \mathrm{f(x,y)dx\,dy} = \int_{-1}^{1} \int_{-1}^{1} \mathrm{f(x(r,s), y(r,s))\, J(r,s)\, dr\, ds}$$

Where $\mathrm{J(r,s)} = \begin{vmatrix} \mathrm{x_r} & \mathrm{x_s} \\ \mathrm{y_r} & \mathrm{y_s} \end{vmatrix}$

ii)Error calculation

$$e_2(h_r,h_s) = (\int_{\Omega} (u_x(x,y) + u_y(x,y) - [(u_{exact})_x + (u_{exact})_y])^2 dxdy)^{1/2}$$

Part II

Parallelization is very effective in computing large and complex problem as it distributes the loops to each thread to complete. Similarly for this assignment we focus on timing of computing function error by adding "omp_get_wtime()" as well as "parallel do" function.

i)Functions:

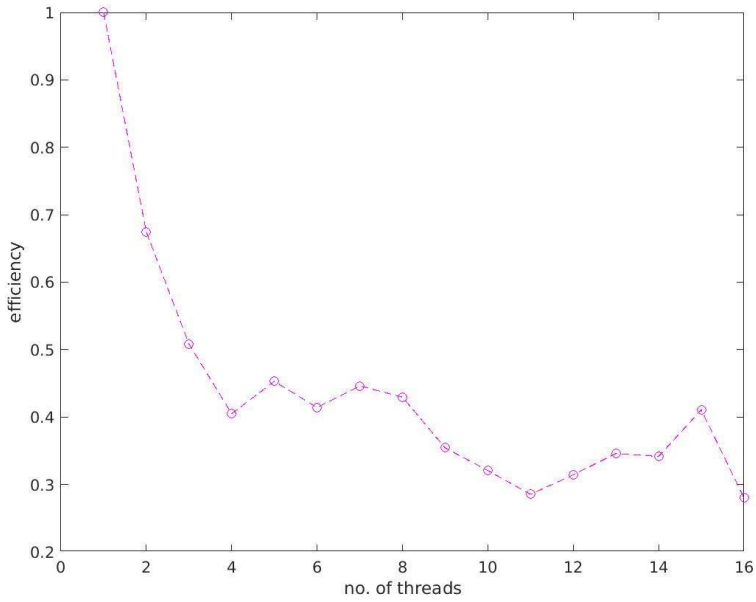
$$n_r = n_s = n$$
$$h_r = h_s = h = 2/n$$
$$\mathrm{x(r,s)} = \mathrm{r} + 0.1\mathrm{s}$$
$$y(r,s) = \mathrm{s}$$
$$u(x,y) = \sin(\mathrm{x}) \times \cos(\mathrm{y})$$
$$u_{x,exact} = \cos(x) \times \cos(\mathrm{y})$$
$$u_{y,exact} = -\sin(\mathrm{x}) \times \sin(\mathrm{y})$$

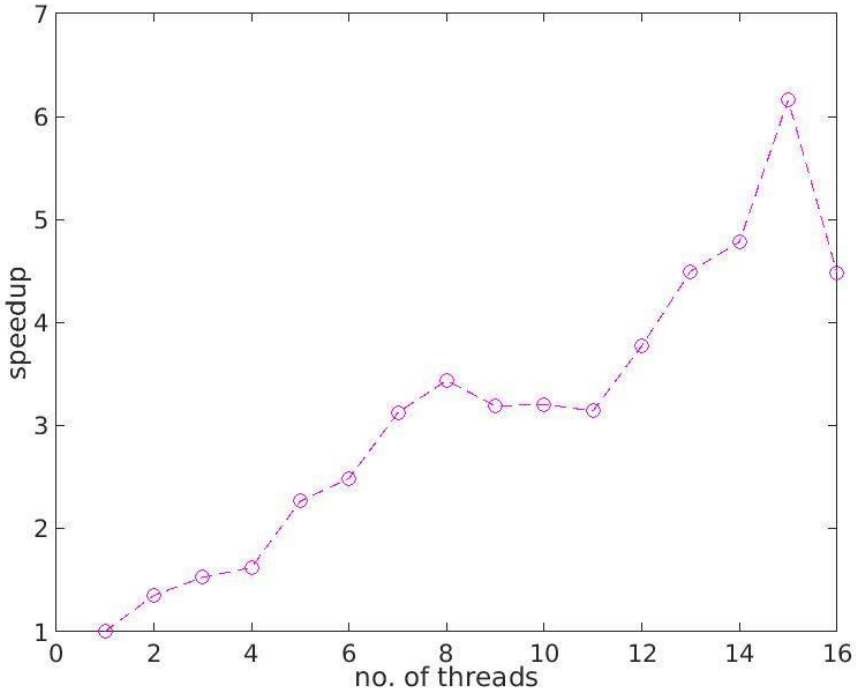
ii) Parallel code and serial code comparison

I compared parallel code and serial code in my local machine. I found the same error values for parallel with OpenMP and serial code.

iii) Timing Function

The timing function calculates the error for different mapping to get convergence. The figure below provides the level of precision of our mapping.



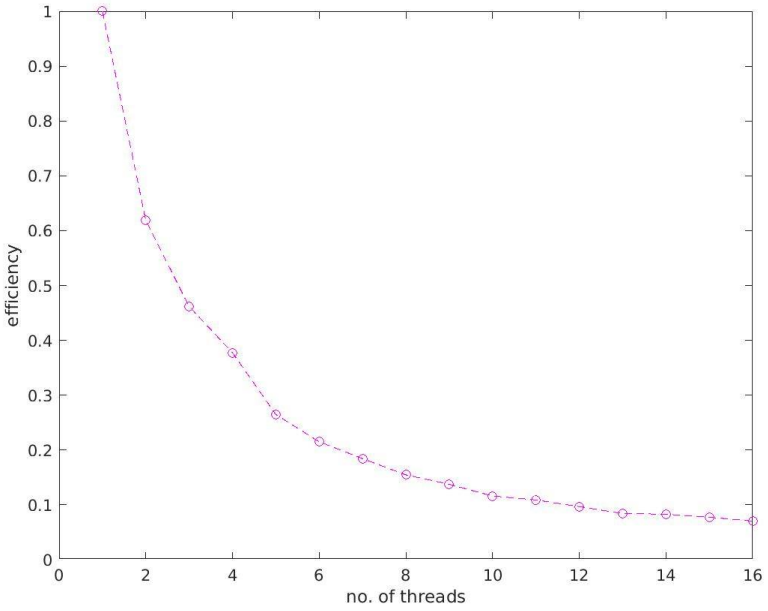


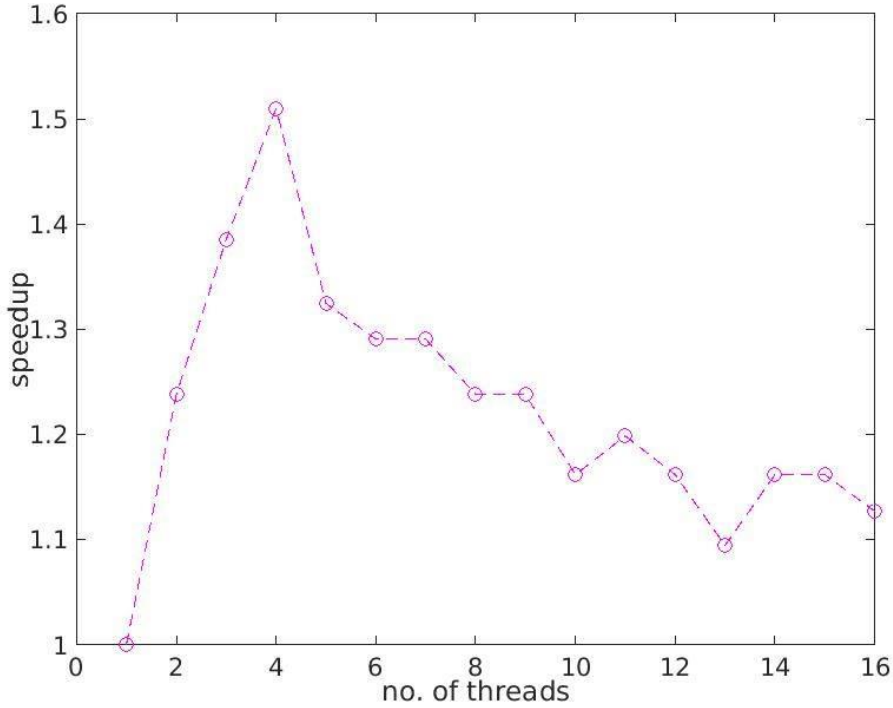
The speedup is ratio of the serial runtime to the time taken by the parallel algorithm to solve the same problem on l cores. The efficiency is defined as the ratio of speedup to the number of cores. We have computed and displayed the speedup and efficiency (with 1-16 cores) for fixed problem size below.

iv) Strong Scaling

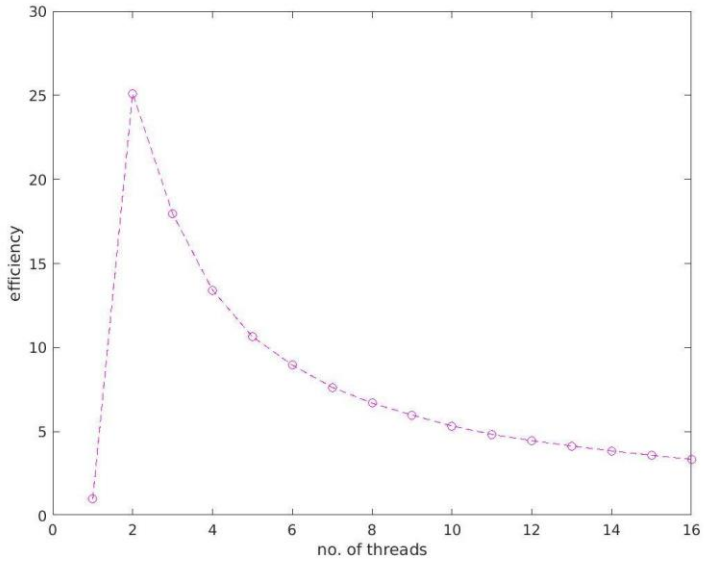
The grid size is same for different thread numbers in strong scaling, however we used two separate grids sizes in computation i.e ‘small 20 by 20’ and ‘large 800 by 800’.

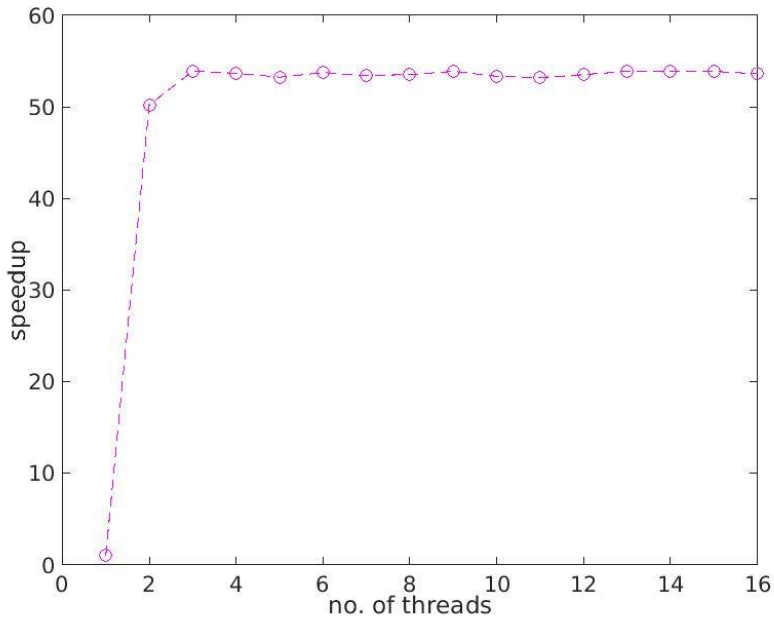
Small grid 20*20





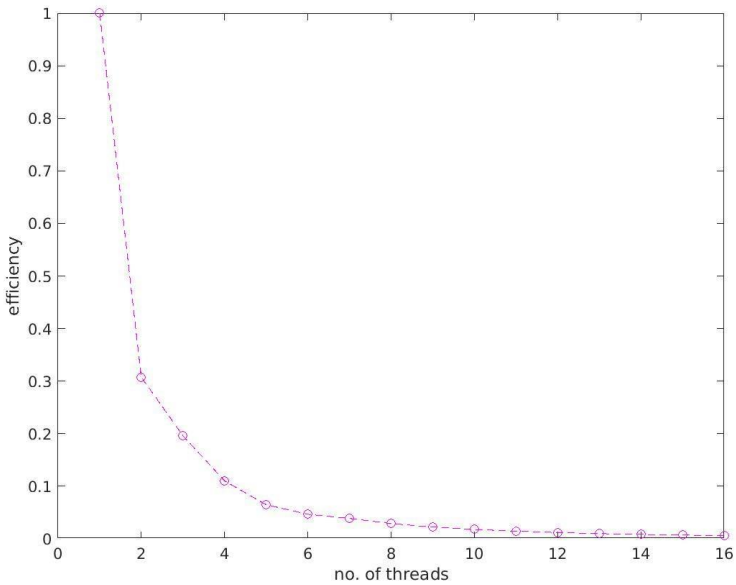
Large grid 800*800

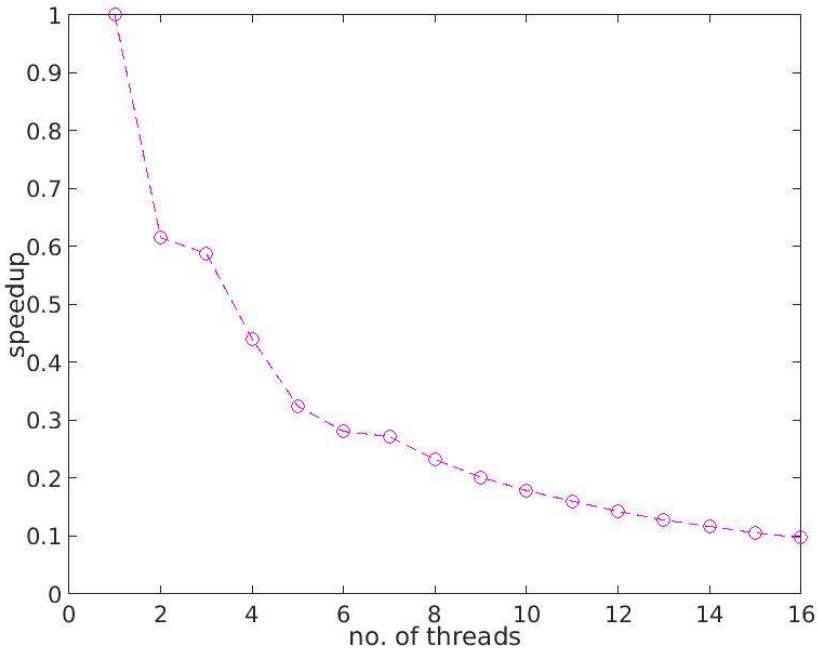




The above graph reflects change in efficiency and speed as number of threads increases. We can conclude efficiency decreases with increasing thread number regardless of the grid size. However, the speed remains fairly same on larger grids size with increasing thread, but we could not conclude exact relationship between speed and threads on smaller grid size.

Weak Scaling





The efficiency decreases and approaches to constant when there are many threads even if the grid size is changing. The speedup decreases with increasing number of threads.

Finally

We identified some sections to split up some processes for threads also we added parallel do's in our code significantly. We believe the speed of code increased with increasing the threads although there were some outliers.