P. Hadjidoukas

Set 7 - MPI I

Issued: May 3, 2023

Question 1: 2D Diffusion and MPI

Heat flow in a medium can be described by the diffusion equation

$$\frac{\partial\rho(\boldsymbol{r},t)}{\partial t} = D\nabla^2\rho(\boldsymbol{r},t), \tag{1}$$

where $\rho(\boldsymbol{r},t)$ is a measure for the amount of heat at position $\boldsymbol{r}$ and time t and the diffusion coefficient D is constant.

Let's define the domain Ω in two dimensions as $\{x,y\} \in [-1,1]^2$. Equation 1 then becomes

$$\frac{\partial\rho(x,y,t)}{\partial t} = D\left(\frac{\partial^2\rho(x,y,t)}{\partial x^2} + \frac{\partial^2\rho(x,y,t)}{\partial y^2}\right). \tag{2}$$

Equation 2 can be discretized with a central finite difference scheme in space and explicit Euler in time to yield:

$$\frac{\rho_{r,s}^{(n+1)} - \rho_{r,s}^{(n)}}{\delta t} = D\left(\frac{\rho_{r-1,s}^{(n)} - 2\rho_{r,s}^{(n)} + \rho_{r+1,s}^{(n)}}{\delta x^2} + \frac{\rho_{r,s-1}^{(n)} - 2\rho_{r,s}^{(n)} + \rho_{r,s-1}^{(n)}}{\delta y^2}\right) \tag{3}$$

where $\rho_{r,s}^{(n)} = \rho(-1+r\delta x, -1+s\delta y, n\delta t)$ and $\delta x = \frac{2}{N-1}$, $\delta y = \frac{2}{M-1}$ for a domain discretized with $N \times M$ gridpoints.

We use open boundary conditions

$$\rho(x,y,t) = 0 \quad \forall\, t \geq 0 \,\text{and}\, (x,y) \notin \Omega \tag{4}$$

and an initial density distribution

$$\rho(x,y,0) = \begin{cases} 1 & |x,y| < 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

a) Implement the OpenMP parallelization of the 2D diffusion equation. Parallelize the routines that initialize and advance the system.

   The parallel code can be found in `diffusion2d_omp.c`.

b) Implement the MPI parallelization of the 2D diffusion equation by filling in all parts of the code marked by `TODO:MPI`. Decompose the domain using tiling decomposition scheme (described in the lecture notes). (i.e. distribute the rows evenly to the MPI processes).

- *Note 1:* Study and become familiar with the provided OpenMP version of the code.
- *Note 2:* Do not use non-blocking communication (which has not been discussed yet).

The parallel code can be found in `diffusion2d_mpi.c`. Significant points: MPI initialization, getting the rank, the number of processes, MPI finalization, MPI send/receive operations.

A series of snapshots of the simulation in time are shown in the figure below.

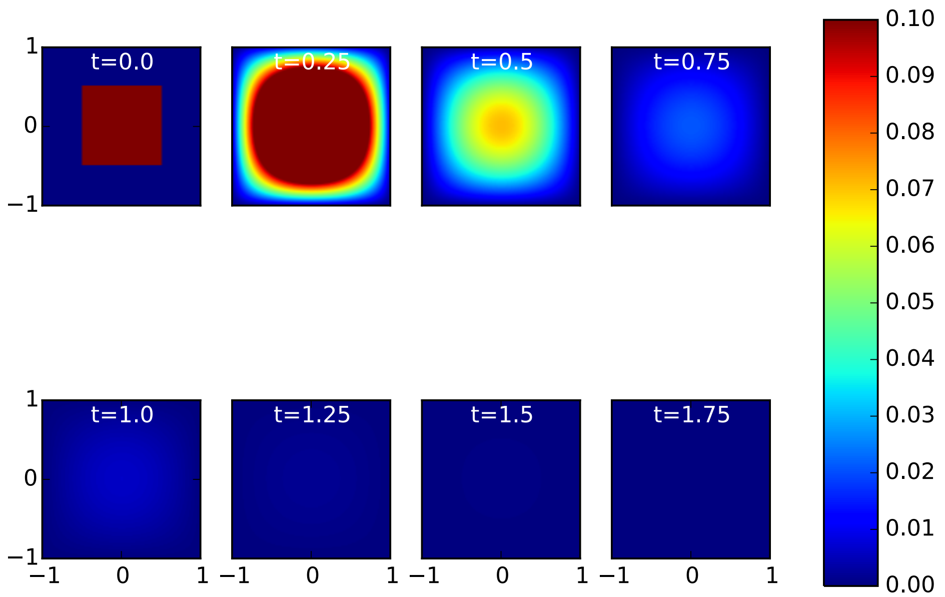

Figure 1: Time evolution of the system for a grid of 128x128 points.

c) Compute an approximation to the integral of ρ over the entire domain in `compute_diagnostics`. Compare your result after 1000 iterations using the result of the provided OpenMP code that solves equation (1). To run the code use the parameters in Table 1.

The parallel implementation is based on the MPI reduction operation.

d) Suggest other ways to divide the real-space domain between processes with the aim of minimizing communication overhead. Prove your argument by computing the message communication size for the tiling domain decomposition and for your suggestion.

Assume a square domain of size NxN is split in P rectangular tiles (stripes) along the $y-$ (or $x-$) directions, with one tile per process. Then the maximum number of elements that need to be communicated between neighboring processes is $2N$, since two boundary rows of grid-points need to be communicated to two neighboring processes. This applies for all tiles that are not on the domain boundaries of the domain (since in our exercise, the boundary conditions are not periodic so data do not need to be communicated at the domain boundaries). For the boundary tiles the message communication size is N since only one row of data needs to be communicated.

Another approach is to split the grid into square tiles with one tile per process. The total amount of communication per process if the domain is divided into square tiles is

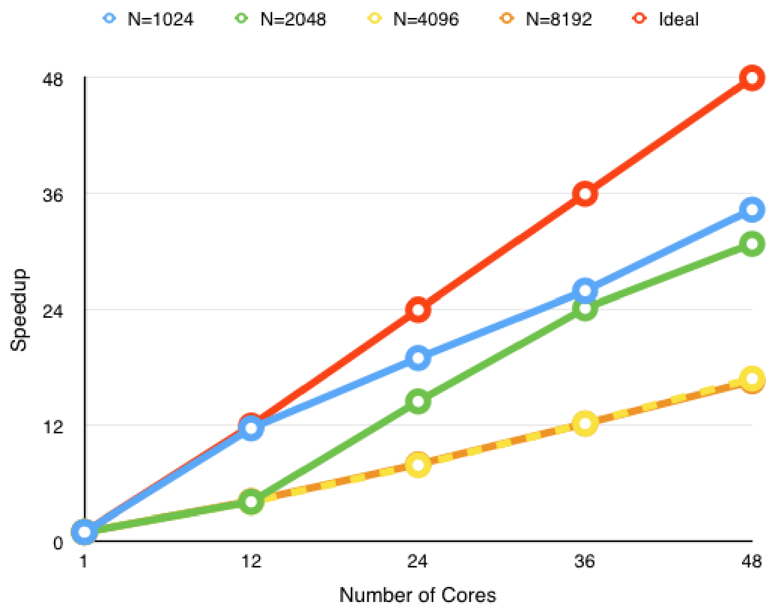

Figure 2: Weak scaling for 100 time-steps. N^2 is the number of local grid points

$2[N_x/P_x + N_y/P_y]$ where $N_{x,y}$ and $P_{x,y}$ is the number of grid points and processes in each direction. As we are doing sparse matrix-vector multiplication the computation scales as N_xN_y/P_xP_y. If we keep the total grid size N fixed and increase the total number of processes P the computational load per process will decrease as $1/P$. The communication will stay constant for rectangular tiles, but will decrease as $1/\sqrt{P}$ for square tiles. Hence finite difference discretization should scale much better if we use square tiles rather than rectangular.

For the boundaries, instead of sending and receiving on each edge of the sub-domain we can send twice the data on one side per dimension and shift the local grid. The next time-step we send from the opposite side and shift the grid back. This will reduce communication overhead and can have a noticeable effect for a small local domain.

e) (Optional) Make a strong and weak scaling plot.

Looking at the strong scaling (Fig. 2), for a total grid size $N = 2^{10}$ the sub-matrix fits entirely into the cache minimizing memory access and resulting in almost perfect scaling. We observe similar behavior for $N = 2^{11}$, as soon as the application utilizes the second NUMA node. As we increase the system size we exceed the cache size and additional memory accesses reduce performance. Despite the inter-node communication between the two nodes, the code continues to scale on up to 48 processes (cores).

In the weak scaling (Fig. 3) the total computational domain per process is kept constant. More specifically, the global grid dimension N_g is equal to $N\sqrt(P)$, where P is the number of processes and N is the grid dimension when running on a single core ($P = 1$). The drop in the scaling for 12 cores is attributed to the fact that the problem size fits entirely into the cache for $P = 1$. We also observe that the efficiency remains constant as the number of NUMA nodes increases.

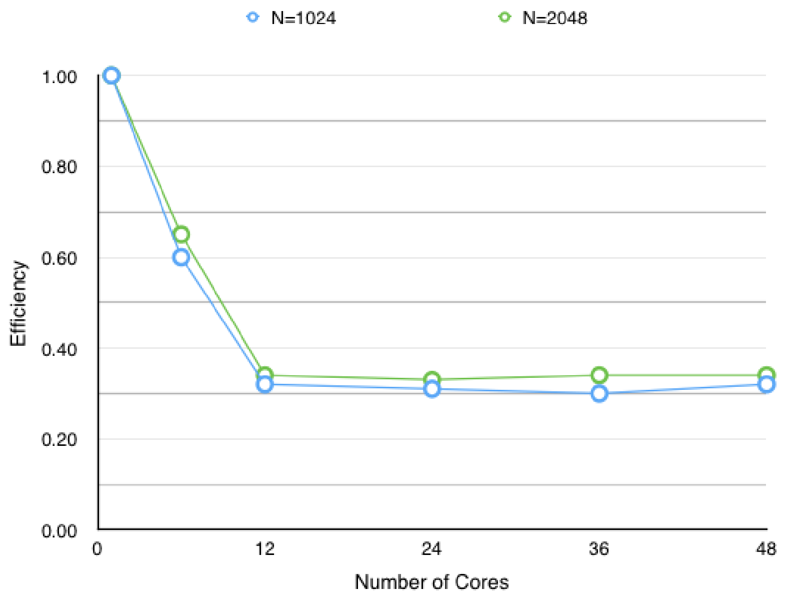

Figure 3: Weak scaling for 100 time-steps. N^2 is the number of local grid points

Table 1: Example parameters.

		$\Omega : [-L, L]$	$N \times N$	timesteps	
	D	L	N	T	Δt
Set 1	1	1	128	1000	0.00001
Set 2	1	1	256	1000	0.000001
Set 3	1	1	1024	1000	0.00000001