University of Chicago

Winter 2023

Advection problem using Lax method, 1st order and 2nd order upwind scheme

Assignment 1 : High Performance Computing

Divya Pattisapu

01-15-2023

1 Milestone 1: Serial Advection

1.1 Introduction

The discretized version of the advection equation, which is a hyperbolic partial-differential equation governing the conserved movement of a specified distribution of scalar quantity with a given velocity, was implemented using the Lax method.

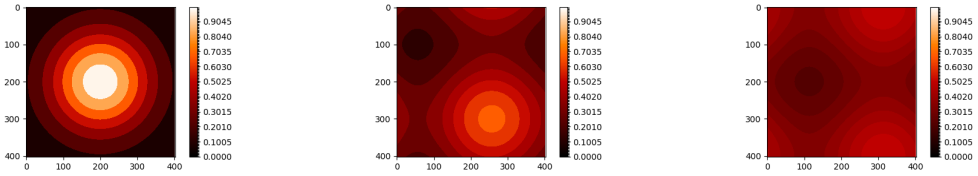
$$C_{i,j}^{n+1} = \frac{1}{4}(C_{i-1,j}^{n} + C_{i+1,j}^{n} + C_{i,j-1}^{n} + C_{i,j+1}^{n}) - \frac{\Delta t}{2\Delta x}[u(C_{i+1,j}^{n} - C_{i-1,j}^{n}) + v(C_{i,j+1}^{n} + C_{i,j-1}^{n})]$$

1. N represents the grid size
2. Work done is proportional to N^2 since we are solving a 2D grid problem
3. NT : Number of time steps
4. T : Total time of simulation
5. L : Length of the square grid
6. C^{n+1} represents the scalar quantity in the next time step
7. C^{n} represents the scalar quantity in the current time step
8. u : velocity of movement in x axis
9. v : velocity of movement in y axis
10. Courant value is the maximum time step that can be taken at which the system remains stable numerically

1.2 Simulation parameters

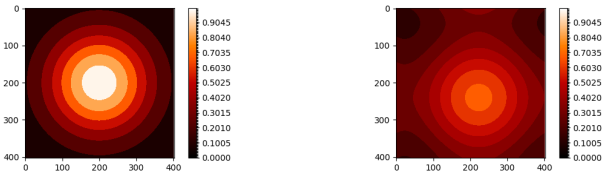
The following parameters were chosen for the serial implementation of the advection equation using Lax method

- Referring to Figure 1, dt = 50, NT = 20000, T = 1000000. From the Lax method equation, we see that the scalar field, is distributed spatially with time and decreases in overall value due to the negative term on the right.
- Referring to Figure 2, dt = 20, NT = 50000, T = 1000000. We see that although the total time is same, there's a difference between the quality of prediction of the scalar field. It should have been the same for 1c and 2d but there is a change, probably due to the granularity of dt
- Referring to Figure 3, dt = 500, NT = 2000, T = 1000000. We see that the script arrives quickly at the answer when dt is large
- Referring to Figure 4: dt = 2857, NT = 350, T = 1000000, this dt is close to the courant value. dt greater than this will render the solution unstable.

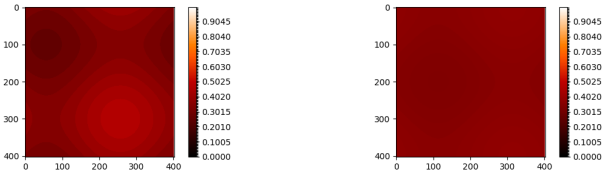


(a) NT = 0 (b) NT = 10000 (c) NT = 20000

Figure 1: dt = 50, NT = 20000, T = 1000000

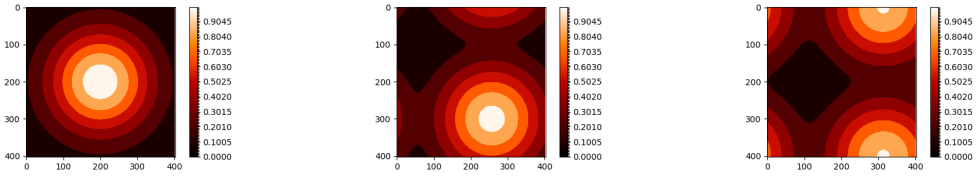


(a) NT = 0 (b) NT = 10000



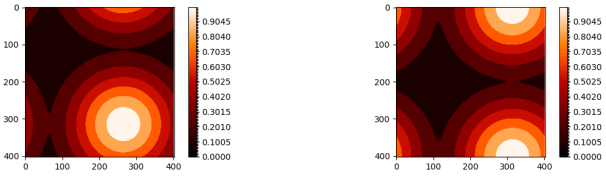
(c) NT = 25000 (d) NT = 50000

Figure 2: dt = 20, NT = 50000, T = 1000000



(a) NT = 0 (b) NT = 1000 (c) NT = 2000

Figure 3: dt = 500, NT = 2000, T = 1000000



(a) NT = 200 (b) NT = 350

Figure 4: dt = 2857, NT = 350, T = 1000000

2 Milestone 2: Multicore scaling studies

2.1 Parallelization Techniques Used

The following three approaches were taken to parallelize the calculations for the problem.

- Explicit loop bound calculations in parallel regions: Parallelizing the outer for loop.
- Nested omp for : Parallelizing both for loops using **omp parallel for**.
- Dividing up the iteration space in slabs : Semi manual allocation of column slabs to each core manually using ids.

2.2 Demonstration of parallel and serial data replication

2.2.1 Lax Method:

These plots are made for the following parameter settings:
laxscheme –N 400 –NT 20000 –L 1.0 –T 1000000 –u 5e-7 –v 2.85e-7 –numthreads 1
Ideally, we cannot have an exact bitwise reproducibility as getting identical floating point results from multiple runs of the same program is not guaranteed. Also, due to dynamic scheduling of parallel computing resources and floating point non associativity, we cannot have an exact bitwise reproducilibility for floating point calculations.

Machine used: Linux server with 20 logical cores with an Intel(R) Core(TM) i7-12700H @ 2.10GHz processor
Compiler used: GNU C Compiler 4:11.2.0-1ubuntu1

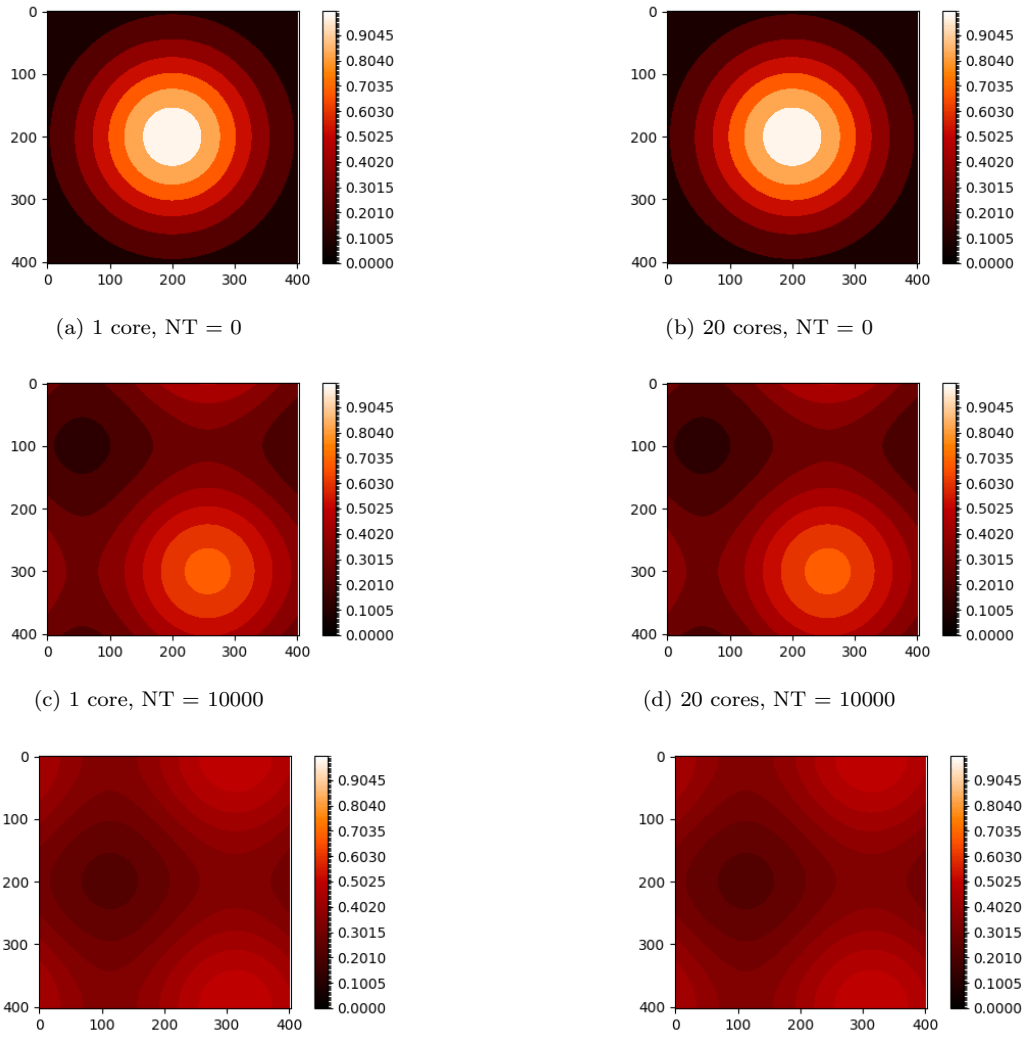
- At NT = 0, Initial condition (compare Figures 5a and 5b : same plots)
- At NT = 10000, Initial condition (compare Figures 5c and 5d : same plots)
- At NT = 20000, Initial condition (compare Figures 5e and 5f : same plots)

2.2.2 Upwind1 Method:

- At NT = 0, Initial condition (compare Figures 6a and 6b : same plots)
- At NT = 10000, Initial condition (compare Figures 6c and 6d : same plots)
- At NT = 20000, Initial condition (compare Figures 6e and 6f : same plots)

2.2.3 Upwind2 Method:

- At NT = 0, Initial condition (compare Figures 7a and 7b : same plots)
- At NT = 10000, Initial condition (compare Figures 7c and 7d : same plots)
- At NT = 20000, Initial condition (compare Figures 7e and 7f : same plots)



(a) 1 core, NT = 0

(b) 20 cores, NT = 0

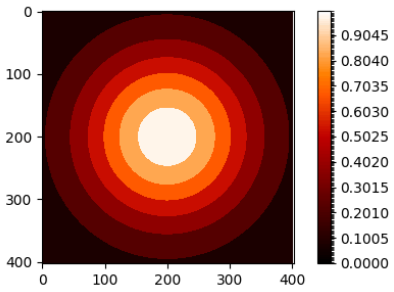
(c) 1 core, NT = 10000

(d) 20 cores, NT = 10000

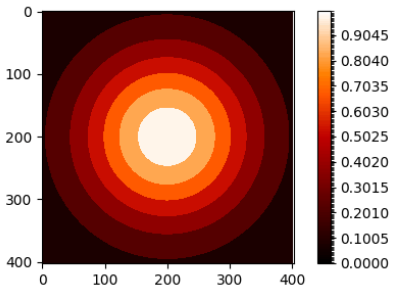
(e) 1 core, NT = 20000

(f) 20 cores, NT = 20000

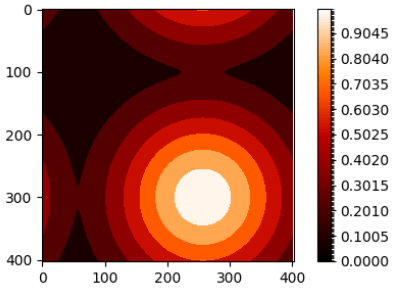
Figure 5: Lax Method: Same answers with serial and parallel versions



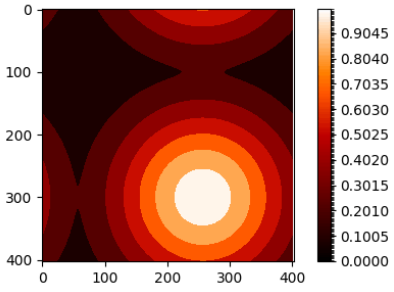
(a) 1 core, NT = 0



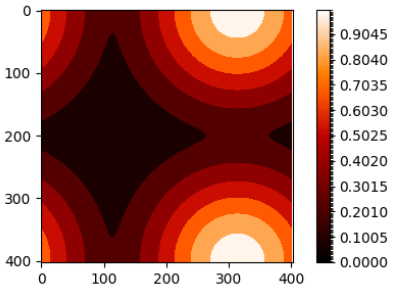
(b) 20 cores, NT = 0



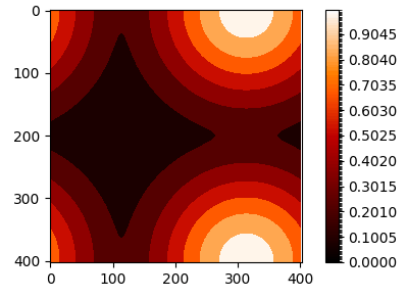
(c) 1 core, NT = 10000



(d) 20 cores, NT = 10000

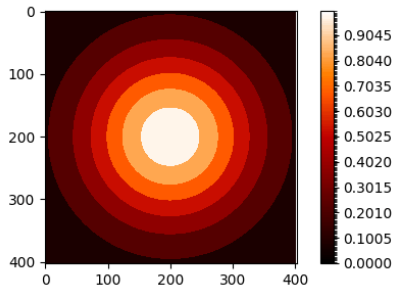


(e) 1 core, NT = 20000

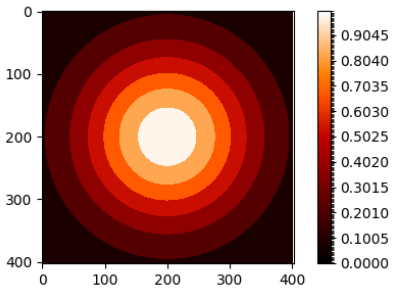


(f) 20 cores, NT = 20000

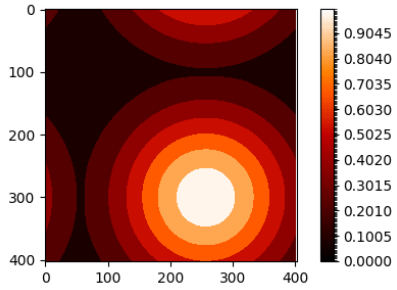
Figure 6: Upwind1 Method: Same answers with serial and parallel versions



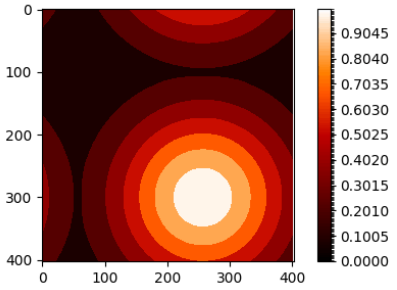
(a) 1 core, NT = 0



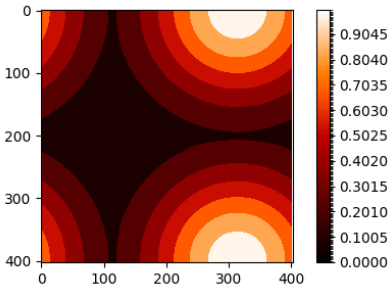
(b) 20 cores, NT = 0



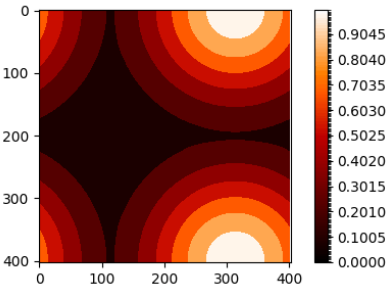
(c) 1 core, NT = 10000



(d) 20 cores, NT = 10000



(e) 1 core, NT = 20000



(f) 20 cores, NT = 20000

Figure 7: Upwind2 Method: Same answers with serial and parallel versions

2.3 Performance metrics used

- Execution time T(N, P) where N is the problem size and P is the number of processors.
- Speedup = T(N,1)/T(N,P) is the multiplier by which performance time decreases.
- Grind rate = N*N*NT/T(N,P) (grid cells/s)
- Grind rate = NT/T(N,P) (time steps/s)
- Parallel efficiency E(N,P) = S(N,P)/P is ideally 1 if the speedup increases linearly with number of processors. However, due to parallelization overhead caused by communication between nodes and other factors, this metric is less than 1.
- Strong Scaling: N remains same, we increase P to see how much we can improve the speedup.

2.4 Best grind rate

Among the 3 parallelization techniques used, the semi manual slab method had the best time. This comparison is made based on the parameters N = 10000, NT = 20000, L=1.0, T = 1e6, u = 5e-7, v= 2.85e-7, P = 64.

- Method 1: Explicit loop bound calculations in parallel regions: Parallelizing the outer for loop
  - T(10000, 64) = 1663 seconds
  - Grind rate = 12.02 (timesteps/s)
  - Grind rate = 1203126927.24 (grid cells/s)
  - Code is present in lax.c
- Method 2: Nested Parallelization
  - T(10000, 64) = 1591 seconds
  - Grind rate = 12.57 (timesteps/s)
  - Grind rate = 1257573903.2 (grid cells/s)
  - Code is present in lax2.c
- Method 3: Semi manual slab wise parallelization
  - T(10000, 64) = 1591 seconds
  - Grind rate = 13 (timesteps/s)
  - Grind rate = 1300065029.24 (grid cells/s)
  - Code is present in main.c

The best grind rate among the three methods is 13 timesteps/s.

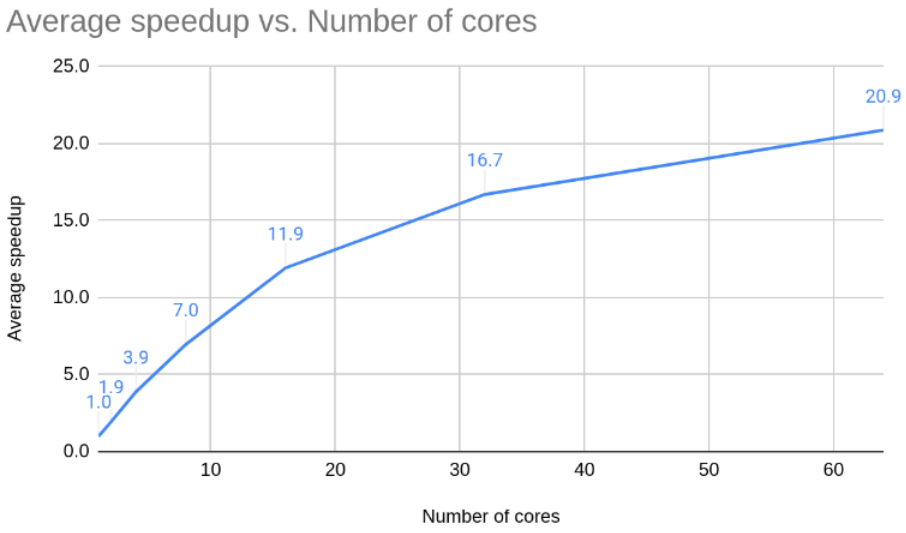


Figure 8: Strong Scaling Plot

2.5 Strong scaling analysis

Strong scaling tries to improve execution time by using more processors without changing the problem size.

- Ideally, the speedup should increase linearly and proportionally (y=x) with the number of processors used.
- For the parameters N = 3200, NT = 400, L = 1.0, T = 1.0e3, u = 5.0e-7, v = 2.85e-7, we get the plot as shown in Figure 8.
- The maximum ideal speedup is 64. However, due to a serial component in the problem, the curve starts to plateau. In this case, it is yet to plateau but I had access to 64 cores.
- Machine used: Linux server with 64 logical cores, 2 threads per core, 16 core(s) per socket and 2 sockets with an Intel Xeon(R) Silver 4216 CPU @ 2.10GHz processor. Compiler used: GNU C Compiler 4:9.3.0-1ubuntu2
- For the second strong scaling study with N=200, the simulation time was too short (0 seconds) due to which the grind rate was calculated as infinity.

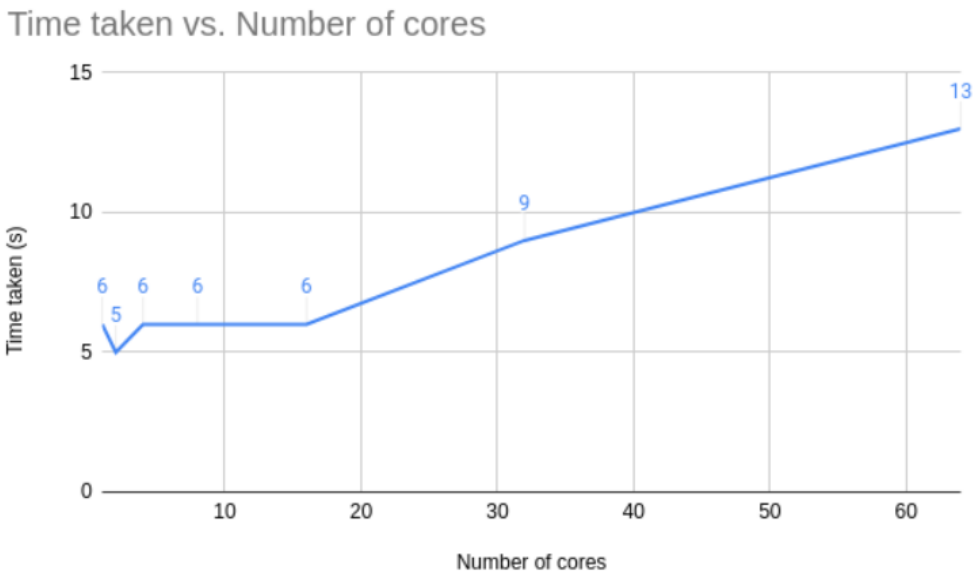


Figure 9: Weak Scaling Plot

2.6 Weak scaling analysis

Here, we scale our problem size with the number of CPUs. The limitation on speedup due to Amdahl's law can be reduced by weak scaling.

- We plot Time taken (in seconds) against number of cores in Figure 9.
- We started with N=800 and increased the value of N such that the time per core remains ideally the same. N = 800*sqrt(P).
- Ideally the weak analysis plot should have been a constant, however, due to the communication overhead, the time taken increases with increase in number of cores.