ME 766 Course Project

Parallelization of Advection Scheme using MPI

Saran S - 133106001
Rahul Joshi - 133106002

Indian Institute of Technology, Bombay

April 29, 2014

Problem Description

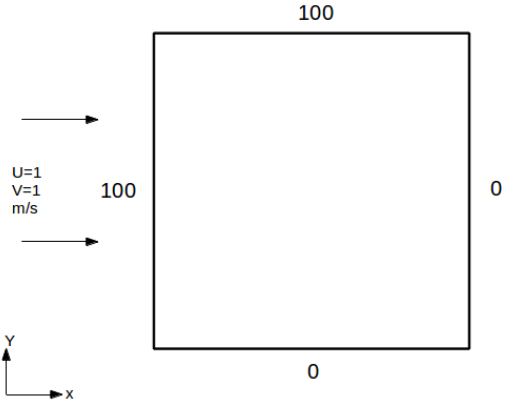


Figure : Schematic of advection problem

Implementation methods

- Serial vectorised Python code using Numpy module
- Using Python MPI library mpi4py

Vectorised code

- Non-vectorised code : Single pair of operands at a time
- Vectorised code : Multiple pair of operands at a time

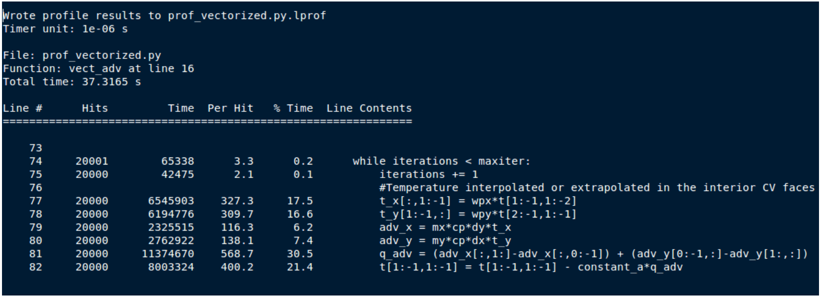
Profiling

- Program analysis to measure the memory and time complexity involved
- Performed to optimize code

Profiling

- Program analysis to measure the memory and time complexity involved
- Performed to optimize code

In our program line by line profiling has been done using Kernprof



```
Wrote profile results to prof_vectorized.py.lprof
Timer unit: 1e-06 s

File: prof_vectorized.py
Function: vect_adv at line 16
Total time: 37.3165 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================

    73
    74     20001        65338      3.3      0.2      while iterations < maxiter:
    75     20000        42475      2.1      0.1          iterations += 1
    76                                                   #Temperature interpolated or extrapolated in the interior CV faces
    77     20000      6545903    327.3     17.5          t_x[:,1:-1] = wpx*t[1:-1,1:-2]
    78     20000      6194776    309.7     16.6          t_y[1:-1,:] = wpy*t[2:-1,1:-1]
    79     20000      2325515    116.3      6.2          adv_x = mx*cp*dy*t_x
    80     20000      2762922    138.1      7.4          adv_y = my*cp*dx*t_y
    81     20000     11374670    568.7     30.5          q_adv = (adv_x[:,1:]-adv_x[:,0:-1]) + (adv_y[0:-1,:]-adv_y[1:,:])
    82     20000      8003324    400.2     21.4          t[1:-1,1:-1] = t[1:-1,1:-1] - constant_a*q_adv
```

MPI Implementation

- To utilise multiple processing elements available based on distributed shared memory concept
- Initiates same script on all PE's
- PE works on a part of whole domain data
- Data finally gathered back to root PE

we implemented MPI library for python - **mpi4py**

Computer Specification

- Hardware : Intel Core i5-2430M CPU @ 2.40GHz 4, Memory 3.8 GB, L1 Cache 32K, L2 Cache 256K, L3 Cache 3072K
- Software : Ubuntu 12.04, 64 bit

Results

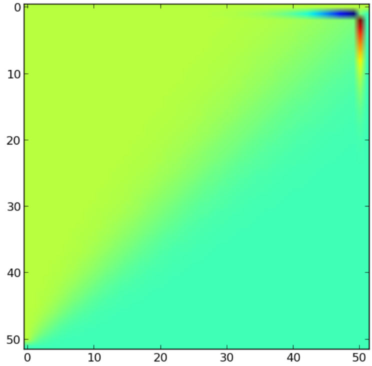
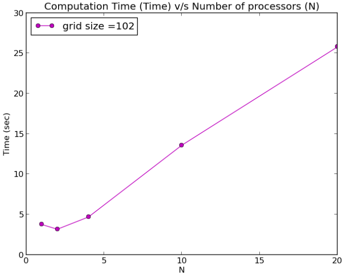
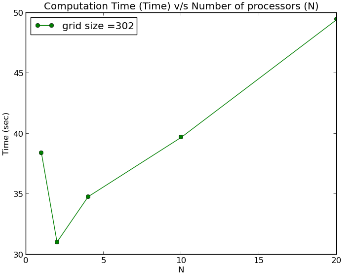


Figure : Solution of the advection problem

Results

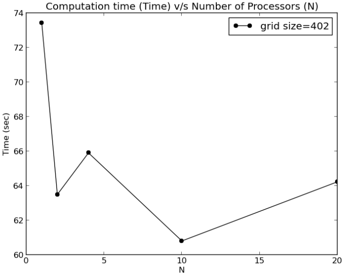


(a) Grid size 102

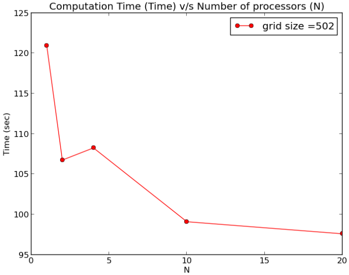


(b) Grid size 302

Results

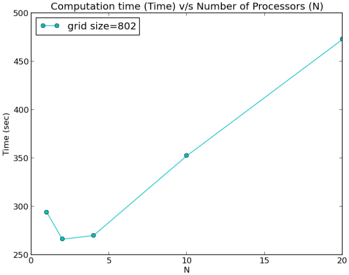


(a) Grid size 402

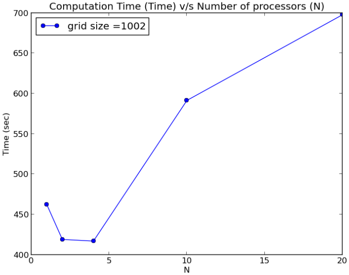


(b) Grid size 502

Results



(a) Grid size 802



(b) Grid size 1002

Conclusions

- Profiling helped us to identify the portion of the code which took maximum time for computation
- The performance for 2 and 4 PEs was as expected with increase in grid size
- For 10 and 20 PEs, the computation time taken was more
- For grid size 402 and 502 there was a change in trend of the computation time