Acceleration of Stencil Code Using MPI

A Hands-on Approach

Learning Objectives

- **APPLY** basic MPI functions to accelerate a common scientific pattern
- **USE** Profiler tools to evaluate bottle necks in code
- **IDENTIFY** which loops/task can be distributed (loop dependency)
- **COMPARE** sequential to distributed execution times

Expected Time of the Activity:

Module Approximate Timing

- Problem Introduction : 5 min
- Sequential Code: 5 min
- Profiling and Identification of main bottleneck, stencil loop: 2 min
- Explain how to accelerate this loop using MPI : 12min
- Compare Sequential vs Parallel OpenMP implementation: 1 min

Total time for the module: 25 minutes

Problem Introduction : Stencil Code

Stencil code: are a class of iterative functions which update an array according to some pattern (stencil)

They are most found in:

- ❑ Scientific Simulations:
  (ex. Heat Transfer)
- ❑ Computational Fluid Dynamics
  (ex. PDE solvers, Poisson Equation)
- ❑ Image Processing
  (ex. Image filters)
- ❑ Machine Learning
  (ex. Convolutional Neural Networks)

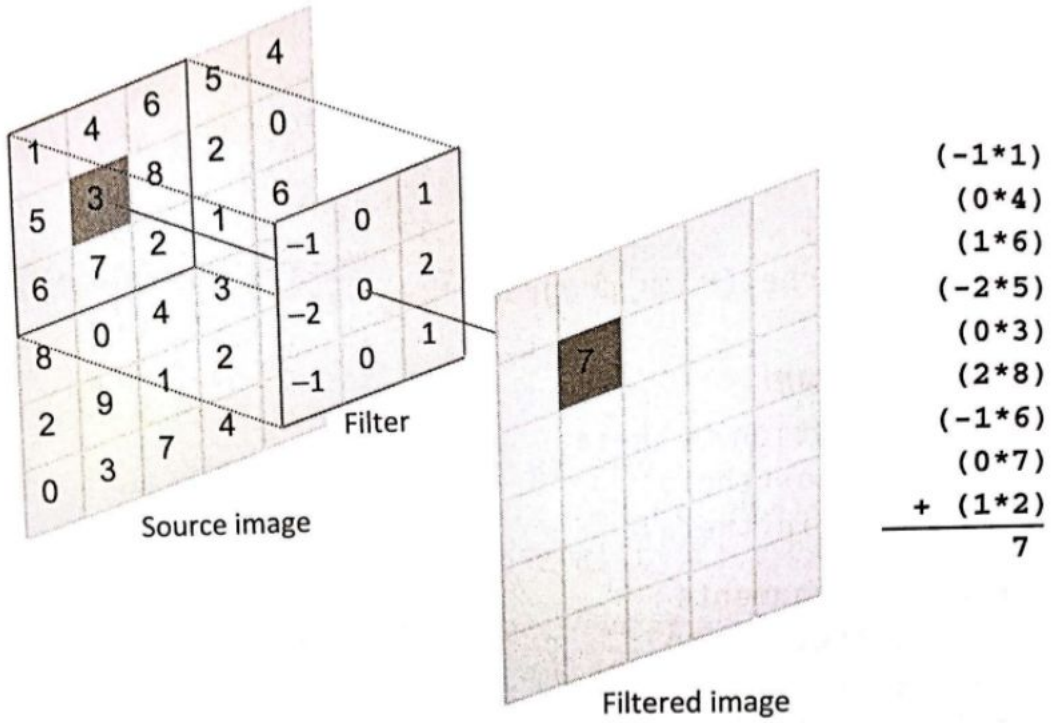
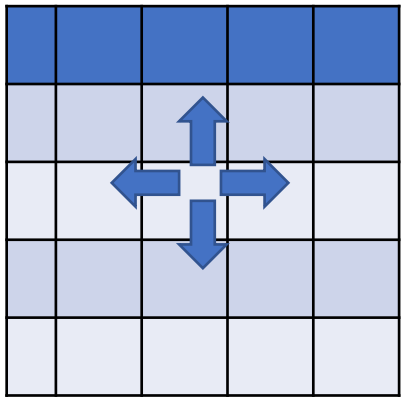


Figure 1 Stencil code on a 2 D array, using 9 neighbors to calculate the output

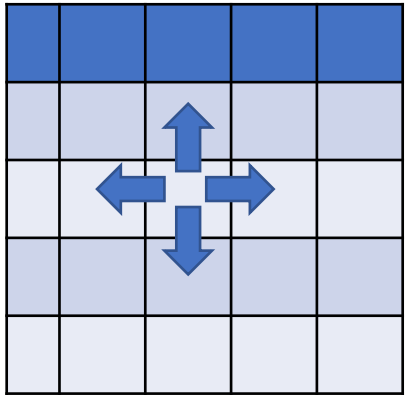
Example Sequential 4 neighbors Code in C No Boundary Consideration

```
for (int i = 0; i < iter_count; ++i) {
    for (int y = 0; y < N; y++) {
        for (int x = 0; x <N; x++) {
         //ctop,cbottom,ceast,cright are the coefficients
         //of the stencil or filter
         out[y][x]=in[y][x]+(ctop*in[y-1][x] +
                        cbottom*in[y+1][x] +
                  cwest*in[y][x-1] +
                  ceast*in[y][x+1] )/SPEED;
        }
    }
    //swap input and output array
    tmp= out; out=in; in=tmp;
  }
```



Example Sequential 4 neighbors Code in C With Boundary

```
for (int i = 0; i < iter_count; ++i) {
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
         //o[y][x] is same as o[y*N+x]
              center=y*N+x;
              west= (x==0) ?center:center-1;  east= (x==N-1) ?center:center+1;
        top= (y==0) ?center:center-N;   bottom= (y==N-1) ?center:center+N;
        out[y*N+x]=in[center]+ (ctop*in[top] +
                        cbottom*in[bottom] +
                  cwest*in[west] +
                  ceast*in[east] )/SPEED;
        }
    }
    //swap input and output array
    tmp= out; out=in; in=tmp;
}
```



NOTE :Explaining Boundary condition notation
If (row x is equal to 0) then west=center
else west=center-1
Can be written in C as one compound statement
west= (x==0) ?center:center-1

Sequential Code Profiling

Full code can be found in file: stencil.c

Compile : gcc –o stencil stencil.c

CPU: Intel i7-7820HQ 8 cores @2.90GHz

gcc –version: 9.3.0

Elapsed time 831.090 sec. (for an array of size 2^11 by 2^11, 6553 iterations(convolutions) and speed 10^8)

Use

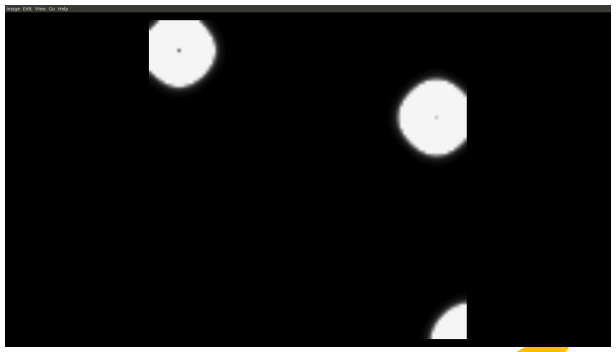
Use gprof or perf (profilers) to analyze timings and bottleneck on code:

Function	Execution %
Init function	0.1%
Stencil	99.9%
PrintResult	0%

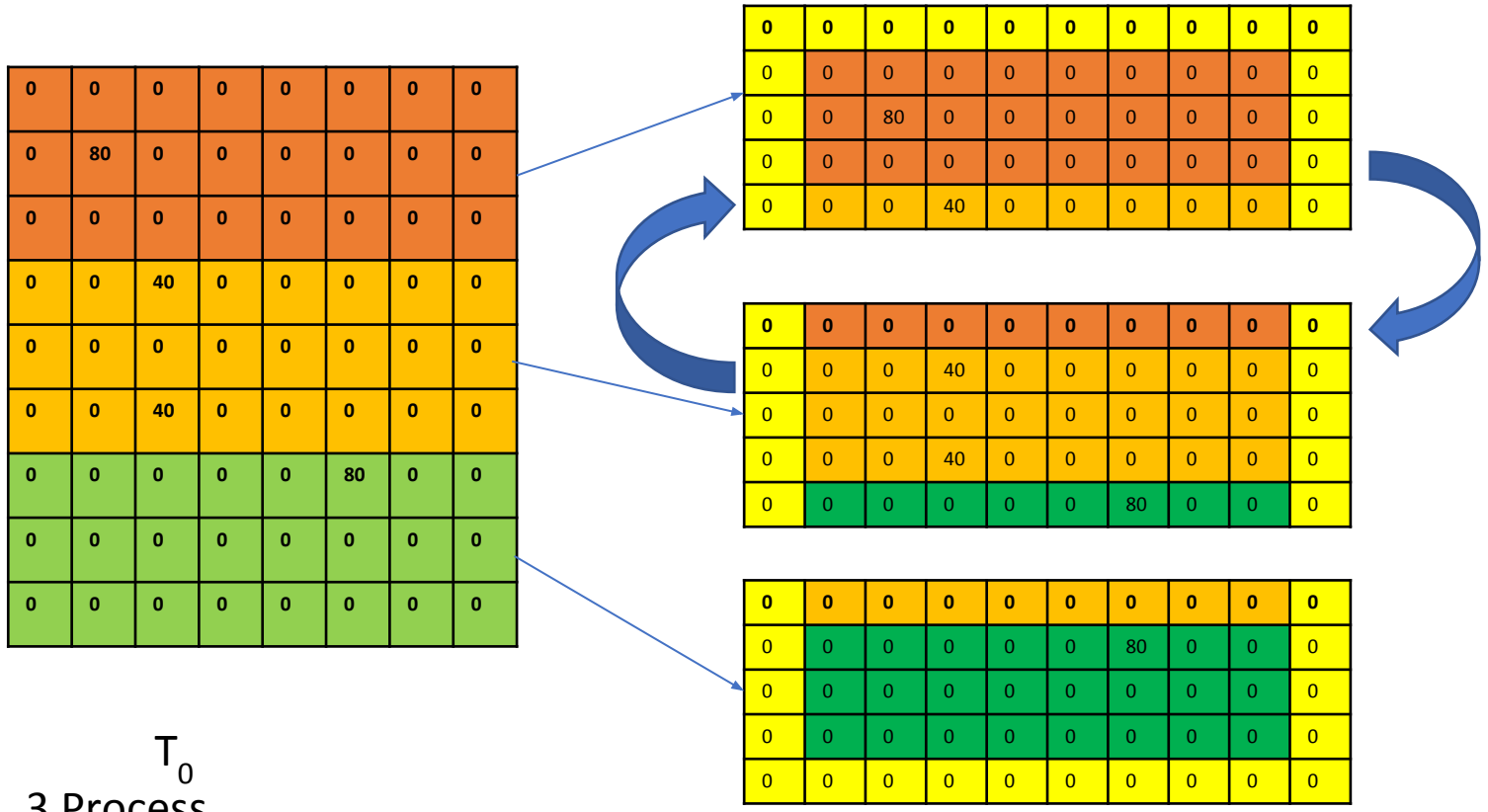
in our case is very clear the function that needs acceleration is the Stencil for loop shown on previous slide (6)

Output Visualization

- The Input to the program is a 2D array , each cell in the array contains the initial temperature
- The Output of the program also a 2D array , contains the temperatures after an X amount of time
- The 2D arrays output can be displayed as png images



MPI Acceleration of main loop. Divide Array in rows



T_0

3 Process

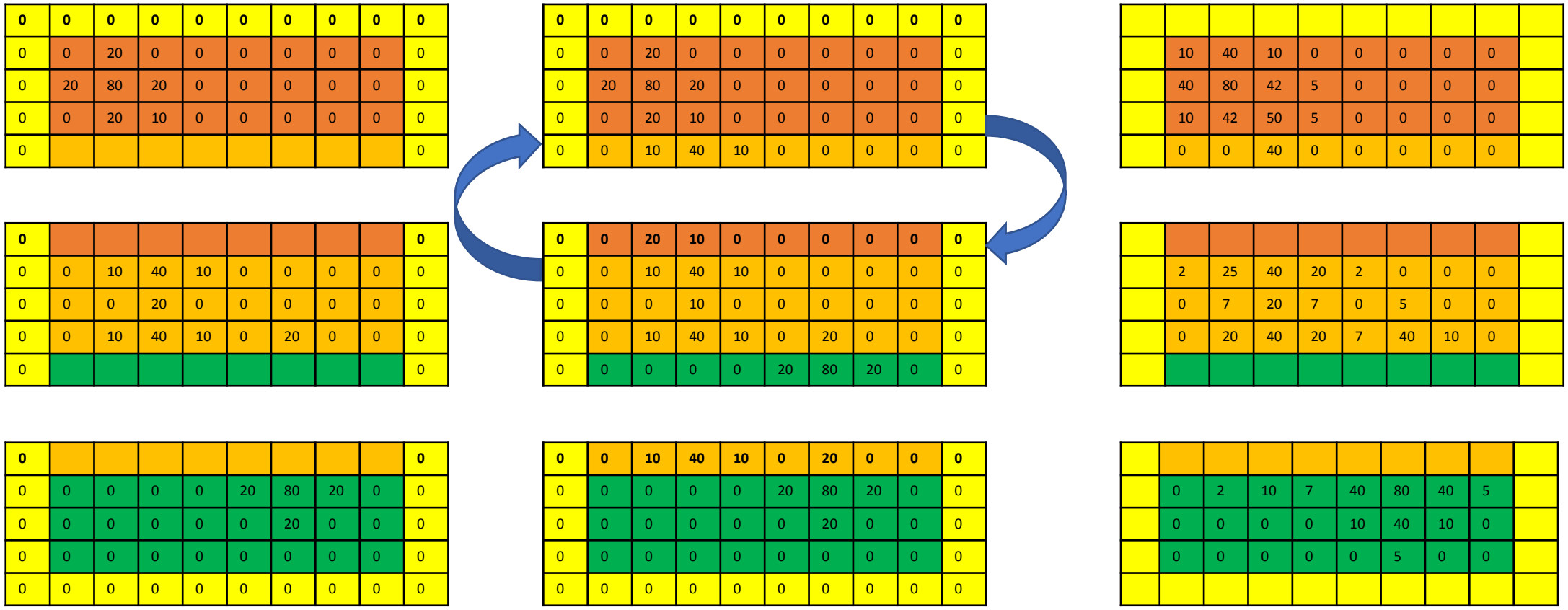
Each calculates 3 rows

To Calculate the Heat at T_1 each process needs its 3 rows plus the top/button boundery

T_1

T1=T0+(Ncell+Scell+Ecell+Wcell)/4

MPI Acceleration of main loop. Divide Array in rows



T_1 exchange T_2

MPI Stencil

```
//Distribute the Data Matrix among the processes
int myRows=ROWS/numProcess;
float *myData=new float[myRows*cols];
float *buff= new float[myRows*cols] ;
MPI_Scatter(data, myRows*cols, MPI_FLOAT, myData, myRows*cols,MPI_FLOAT);
memcpy(buff,myData, myRows*cols*sizeof(float));
//exchange the boundary rows
for (int iter=0; iter<numIter;iter++){
        If(myID>0){
            MPI_Isend(myData,cols,MPI_Float,myID-1,0);  //send first row of myData to previous process
            MPI_Irecv(prevRow,cols,MPI_Float,myID-1,0); //receives last row from previous process
         }
        If(myID<numProcess-1){
            MPI_Isend(&myData[(myRows-1)*cols],cols,MPI_Float,myID+1,0); //send last row of myData to next process
            MPI_Irecv(nextRow,cols,MPI_Float,myID+1,0); //receives first row from the next process
         }
```

MPI Stencil

```
//calculate main Loop
 for(int i=1; i<myRows-1;i++){
      for(int j=1; j<cols-1;j++){
           buff[i*cols+j] = myData[i*cols+j]+( cbotton* myData[(i+1)*cols+j]+
                                                cwest*cmyData[i*cols+j-1] +
                                              ceast*myData[i*cols+j+1] +
                                ctop*myData[(i-1)*cols+j])/SPEED;
```

(full code on stencilMPI.c)

MPI Stencil

```
//calculate First row
If(myId>0){
    request[1].wait(status);
    for(int j=1; j<cols-1;j++){
        buff[j] = myData[j]+( cbotton* myData[cols+j]+
                                cwest*myData[j-1] +
                            ceast*myData[j+1] +
                        ctop*prevRow[j])/SPEED;
}}
```

MPI Stencil

```
//calculate last row
If(myId>numP-1){
    request[3].wait(status);
    for(int j=1; j<cols-1;j++){
            buff[(myRows-1)*cols+j]=
        cbotton*nextRow[j]+
        cwest*myData[(myRows-1)*cols+j-1]+
        ceast*myData[(myRows-1)*cols+j+1]+
        ctop*myData[(myRows-2)*cols+j];
}}
//At the end of stencil swap buffers
Swap(buff with myData and so the for loop again)
```

OpenMP Acceleration Profiling

CPU: Intel i7-7820HQ 8 cores @2.90GHz

gcc –version: 9.3.0

MPI version: OpenMPI 1.2

Compile : mpic++ -g –Wall –o3 –o stencil stencilMPI.c

Run: mpiexec –n 4

Use

a. MPI_Wtime() to time areas in your code

b. Use gprof or perf (profilers) to analyze timings and bottleneck on code

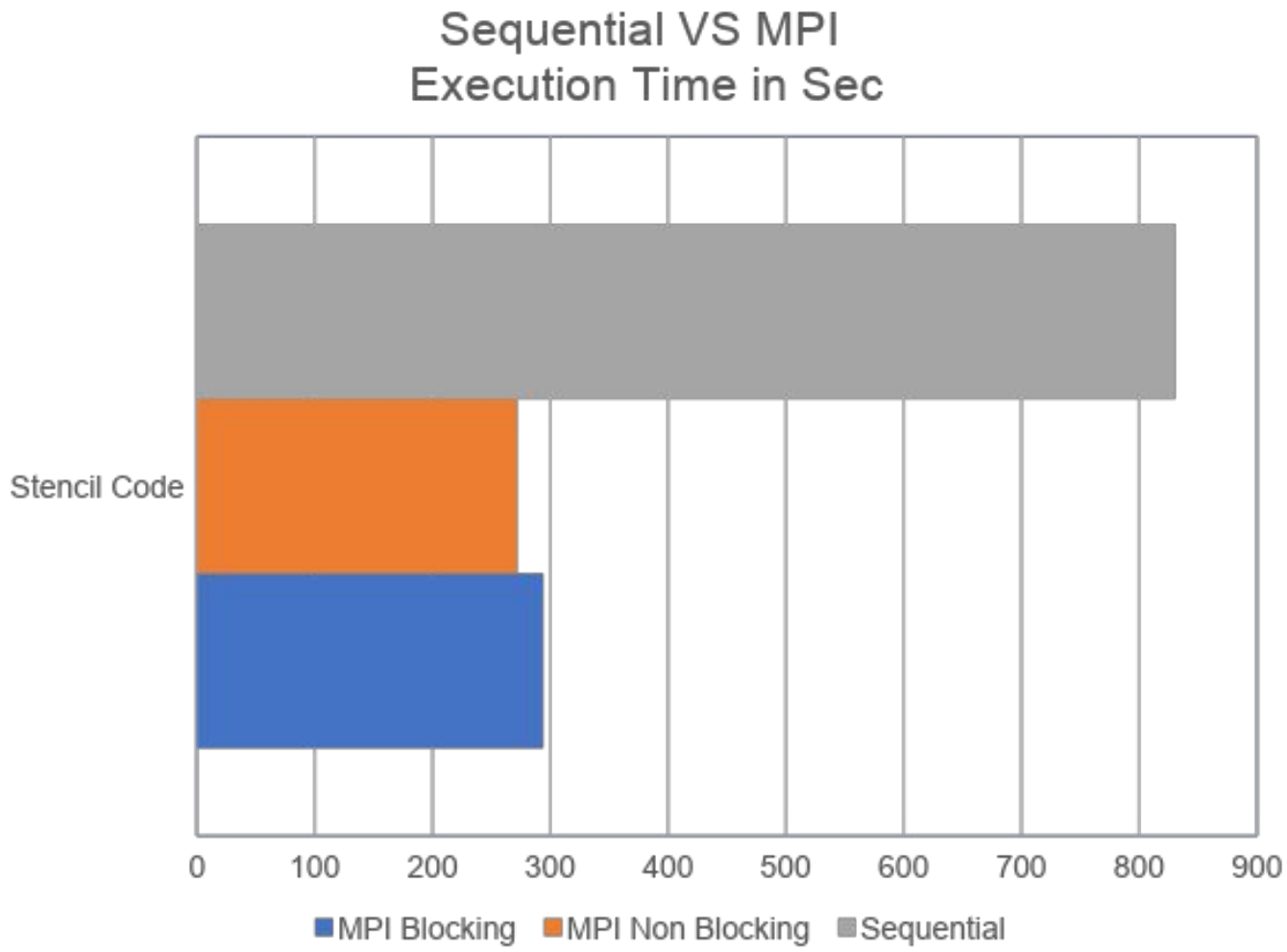
Elapsed time:

Non-Blocking: 294 sec

(for an array of size 2^11 by 2^11, 6553 iterations(convolutions) and speed 10^8)

A ~4 Speedup

Comparation Sequential vs MPI



Conclusion

- Stencil Code is ubiquitous is Science:
  - Differential Equations Solver
  - System of Equations Solver
  - Heat Transfer
  - Artificial intelligence
  - Etc
- With small changes to original sequential code we achieved speeds up of ~4x on an ”oldish and cheap” laptop, clearly only running on the multicores of the machines not on a real distributed system

Module Evaluation

- Possible Advance Questions:
  - (code) The stencil code :

  Rewrite code for stencil so instead of hardcoding in the loop the neighbor cells. Make the code more flexible and input the neighborhood and its coefficients as a filter. Pass the filter as an input to the stencil function
  - (code) Change it to 3D stencil code
  - (code) Chunk not just by rows do it also by columns