Acceleration of Stencil Code Using OpenMP

A Hands-on Approach

Learning Objectives

- **APPLY** basic OpenMP pragmas to solve a common scientific parallel pattern
- **IDENTIFY** which loops in a program are parallelizable
- **COMPARE** sequential to parallel execution times
- **USE** Profiler tools to evaluate bottle necks in code

Expected Time of the Activity:

Module Approximate Timing

- Problem Introduction : 5 min
- Sequential Code: 5 min
- Profiling and Identification of main bottleneck in stencil loop: 2 min
- Explain how to accelerate this loop using OpenMP pragmas: 10min
- Compare Sequential vs Parallel OpenMP implementation: 3 min

Total time for the module: 25 minutes

Problem Introduction : Stencil Code

Stencil (also know as convolution or filtering) 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)
- ❑ Image Processing
  (ex. Image filters)
- ❑ Machine Learning
  (ex. Convolutional Neural Networks)

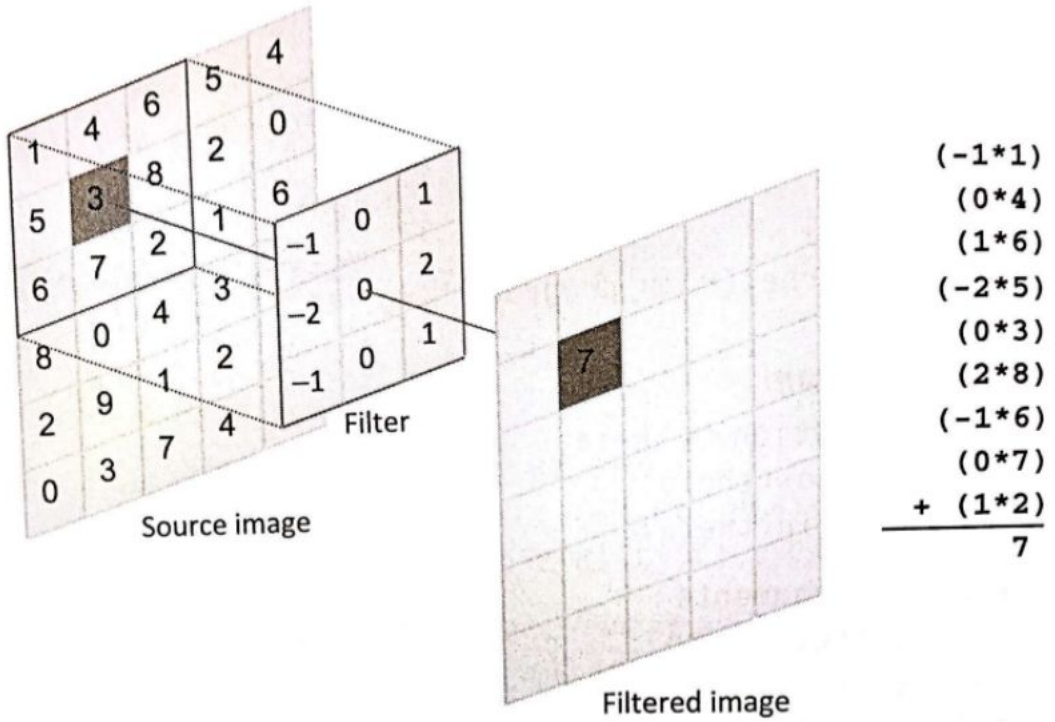
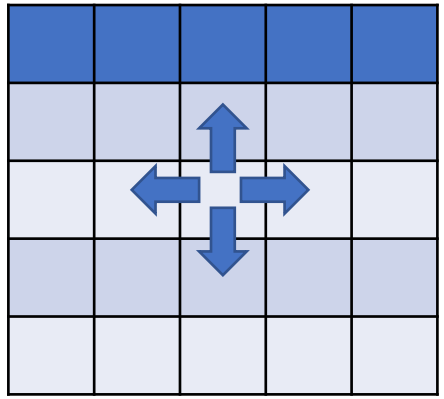


Figure 1 Stencil code on a 2 D array, using 9 neighbors to calculate the output

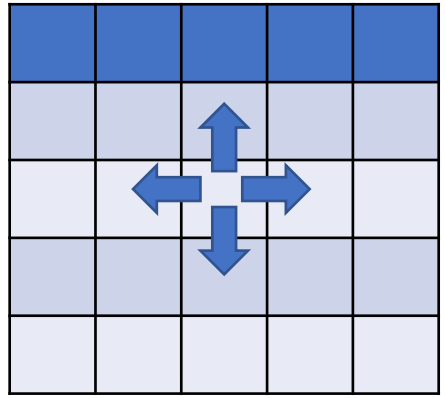
Example Sequential 4 neighbors Code in C No Boundary Consideration

```
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;
}
```



Sequential Code Profiling

Full code can be found in file: stencil.c

Compile : gcc –g –Wall –o3 –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 2^12 by 2^12, 6553 iterations(convolutions) and speed 10^3)

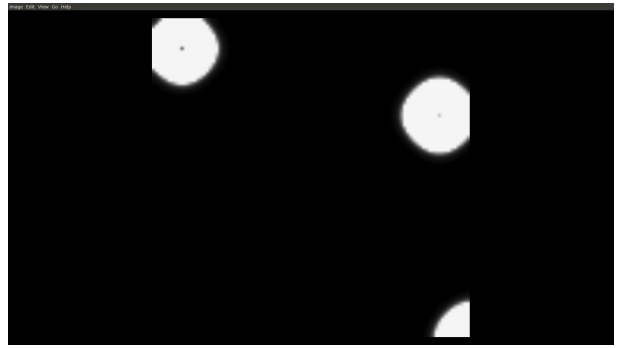
use gprof or perf to analyze timings and bottleneck on code:

Function	Execution %
Init() function	0.1%
heatTransfer().	99.9%
Print_result()	0%

in our case is very clear the function that needs acceleration is the heatTransfer(), which is the function that executes the code on the previous slide

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 is also a 2D array with the temperatures after a x amount of time, showing how the heat transfer from original cells to their neighbors
- The 2D arrays output can as png images



OpenMP Acceleration of main loop

```
1. for (int i = 0; i < iter_count; ++i) {
    2. for (int y = 0; y < height; y++) {
        3. 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;    ast= (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;
}
```

1. This loop controls the number of times we calculate the stencil code.
To calculate stencil at t2 we need the results at t1 so there is a clear loop dependency here.
So Do NOT add #pragma omp parallel for for this loop

2. A pragma for here will assign one iteration of the for loop for each thread.
Therefore each thread will execute the entire inner loop
3. and each writing the result on a different loc
out[threadId][x]
So there is no loop dependency and the #pragma omp can be placed

3. If we place a pragma for here there is no loop dependency, but the work done per thread is very small and most probably the code will be SLOWER

OpenMP Acceleration of main loop

```
1. for (int i = 0; i < iter_count; ++i) {
    //stencil code
   #pragma omp parallel for num_threds(8) collapse(2) schedule(static,4)
   2. for (int y = 1; y < height-1; y++) {
       #pragma omp  simd
       3. for (int x = 1; x < width-1; x++) {
        //f1,f2,f3,f4 are the coefficients
        //of the stencil or filter
        out[y][x]=f1*in[y-1][x] +
                    f2*in[y+1][x] +
              f3*in[y][x-1] +
              f4*in[y][x+1] +
       }
   }
   //swap input and output array
   tmp= out; out=in; in=tmp;
  }
```

#pragma simd
to indicate that the loop can be transformed into a SIMD loop
NOTE: it probably is a good idea to also make the array in and out restrict to indicate to the compiler that there is no aliasing between those two pointers

OpenMP Acceleration Profiling

Compile : gcc -g –Wall –fopenmp –o3 –o stencil stencilOpenMP.c

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

gcc –version: 9.3.0

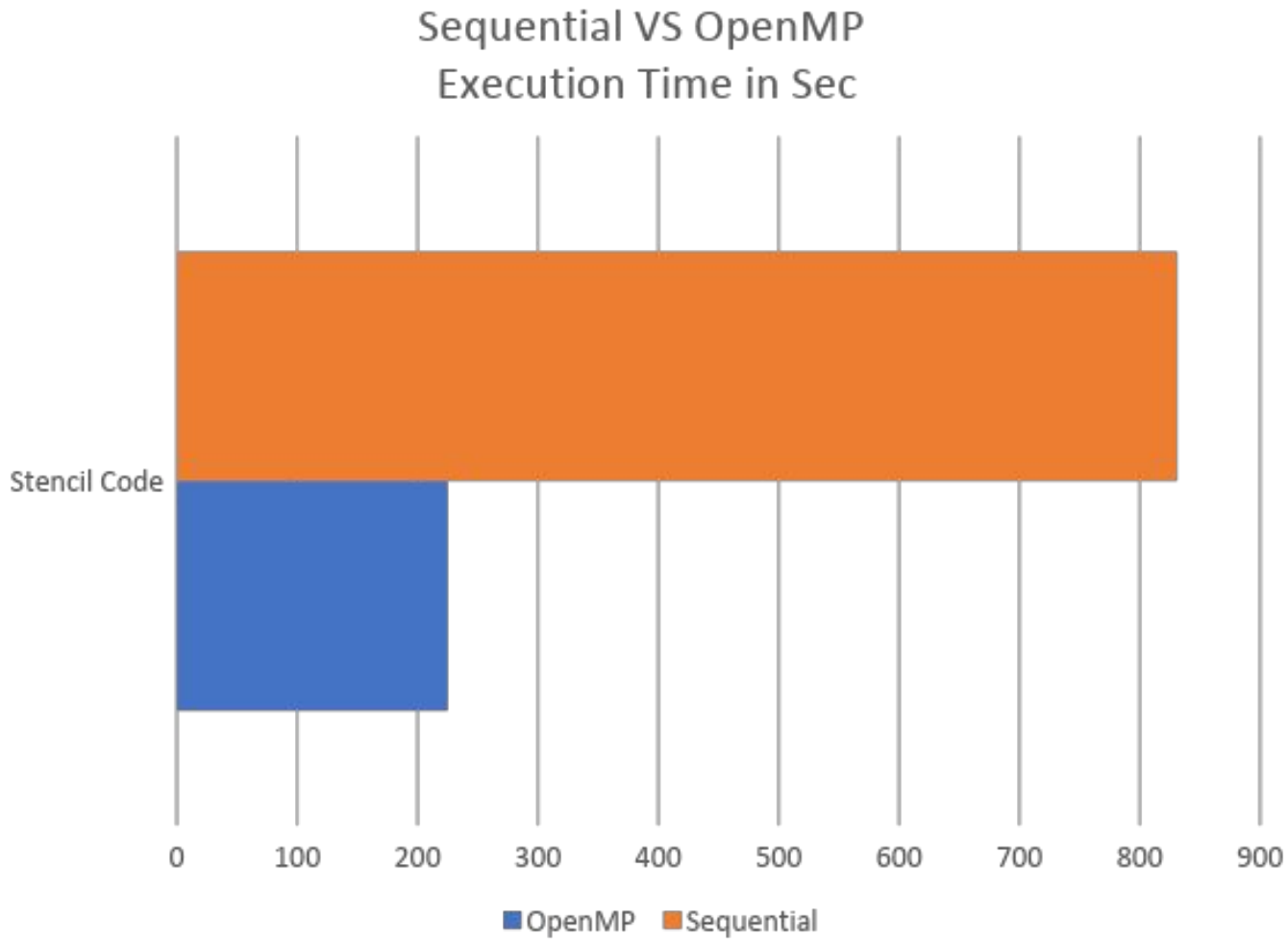
OpenMP version: 201307 ->4.0

command output:

Elapsed time 225.10 sec (for an array of size 2^12 by 2^12, 6553 iterations(convolutions) and speed 10^3)

A ~4x Speedup

Comparation Sequential vs OpenMP



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

Module Evaluation

- Possible Questions for Students:
  - (code) The stencil code :

    Rewrite code for stencil so instead of hardcoding in the loop the neighbor cells, make the code more flexible and take this neighborhood and its coefficients as a filter passed in one input parameter
  - (code) Change it to 3D stencil code
  - if stencil or convolutional filter is separable it can be applied first to rows and than two the columns. Which one maybe faster for acceleration ?
  - On the pragma omp for schedule(static,4) what does it mean? Useful in this example?
  - Will a static, 8 work better? Why?
- Ask students to run code and modify:
  - Enhance memory allocation using: mmap
  - Change stencil to 9 points instead of just 4
  - Explore Collapse() clause
  - Explore the simd Aligned() clause