Australian National University

Parallelizing a 2D advection solver on both CPU and GPU

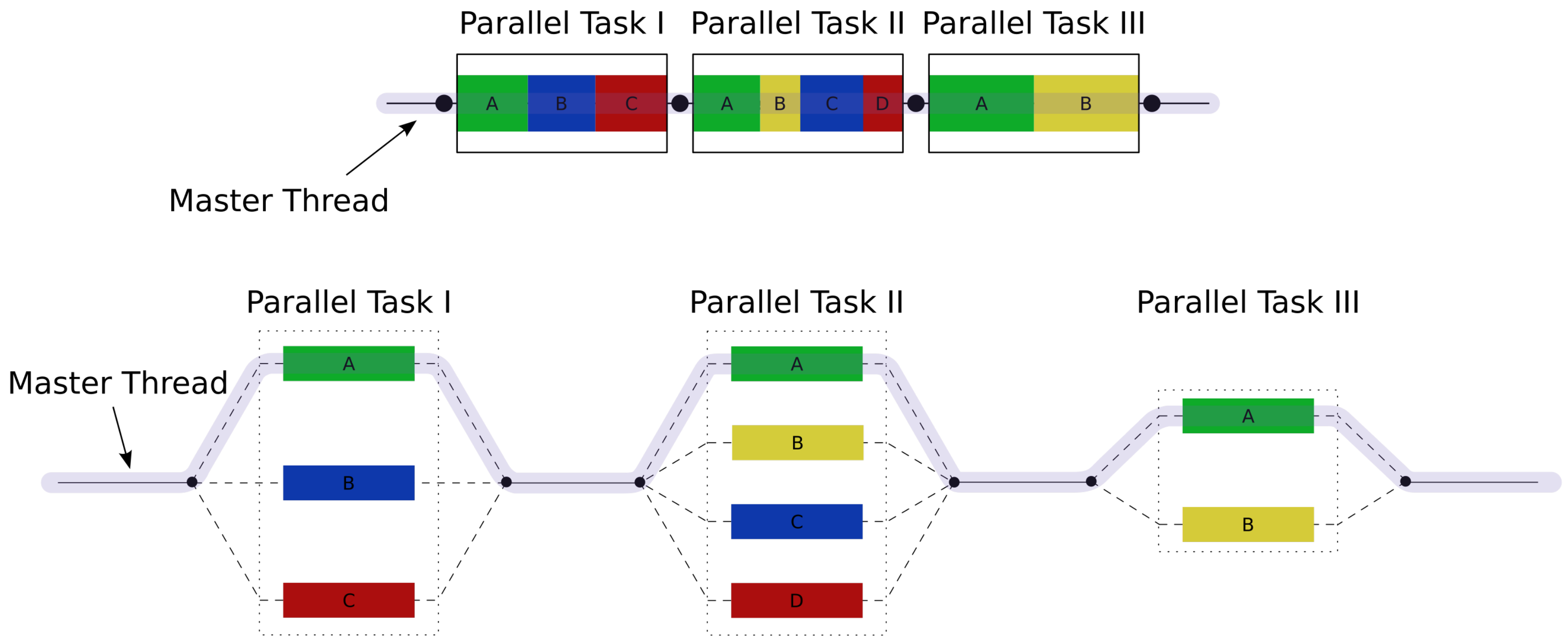
Zhiyuan Huang u6656110 COMP8755

Supervisor: Peter Strazdins

Parallel computing

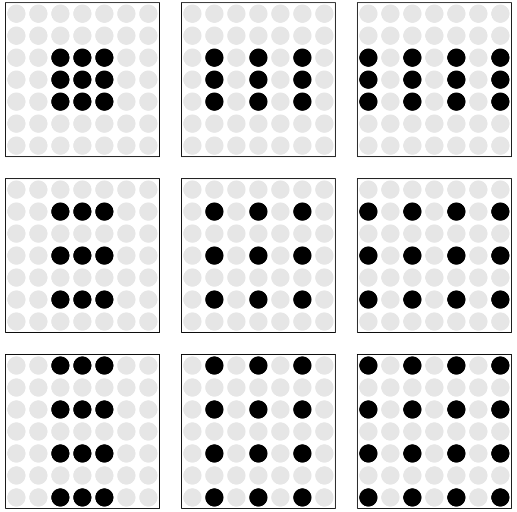
- Large program divided into small tasks, solve tasks independently, combine the results afterwards.
- CPU parallelization (limited number of powerful cores)
- GPU parallelization (tremendous cores with limited power)
- Parallel computing has been widely used in many area including machine learning, high performance computing, etc.

Parallel computing model



Stencil computing

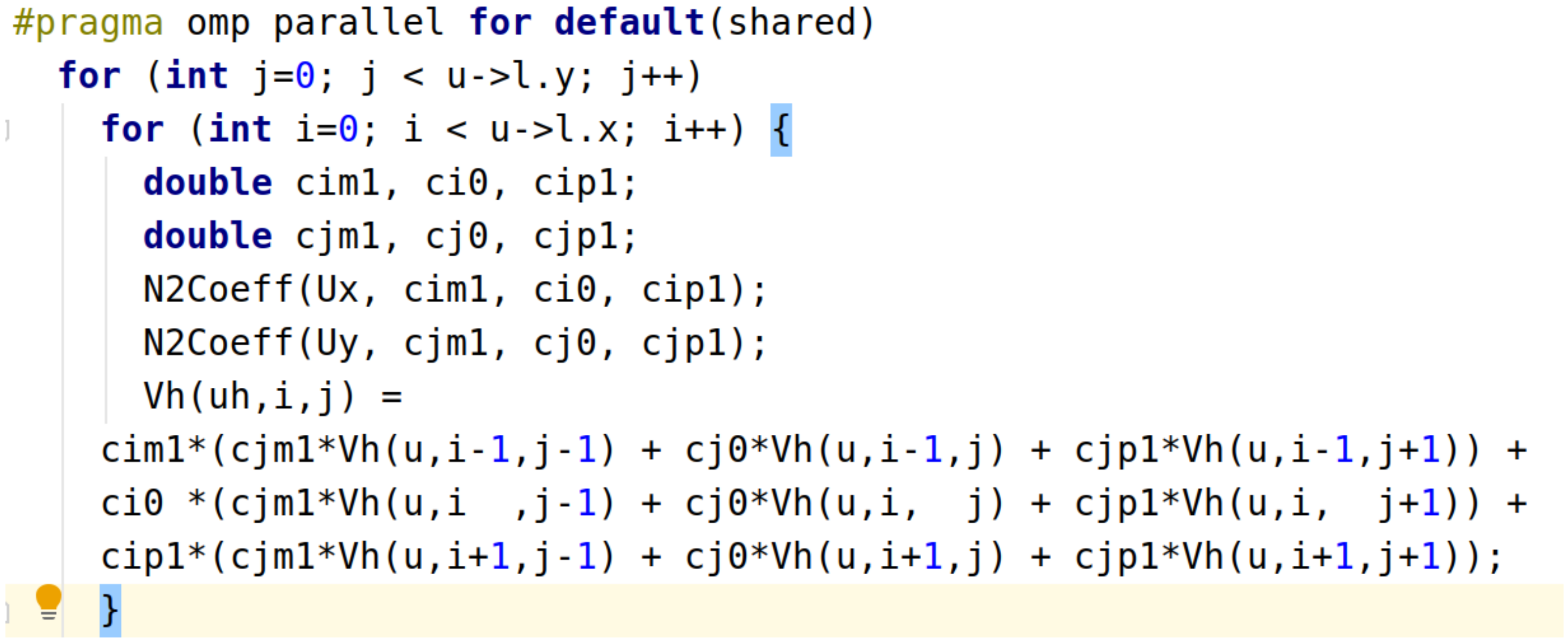
- A program used to solve partial differential equations using stencil computation technique.



A depiction of the nodes used in each stencil. From left to right, top to bottom the stencils depicted are NN, WN, FN, NW, WW, FW, NF, WF and FF.

Peter Strazdins, C. L. J. R. M. J. R., Brendan Harding and Armstrong, R. C.,2016. A robust technique to make a 2d advection solver tolerant to soft faults. Procedia Computer Science, 80, 10 (2016), 1917–1926.

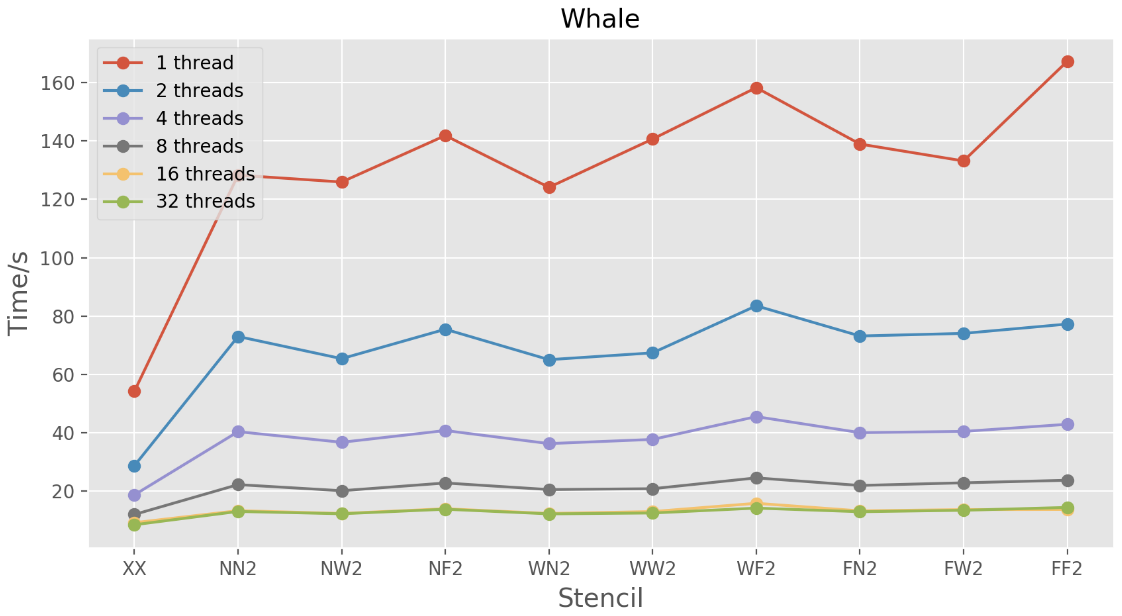
OpenMP implementation



```
#pragma omp parallel for default(shared)
  for (int j=0; j < u->l.y; j++)
    for (int i=0; i < u->l.x; i++) {
      double cim1, ci0, cip1;
      double cjm1, cj0, cjp1;
      N2Coeff(Ux, cim1, ci0, cip1);
      N2Coeff(Uy, cjm1, cj0, cjp1);
      Vh(uh,i,j) =
    cim1*(cjm1*Vh(u,i-1,j-1) + cj0*Vh(u,i-1,j) + cjp1*Vh(u,i-1,j+1)) +
    ci0 *(cjm1*Vh(u,i  ,j-1) + cj0*Vh(u,i,  j) + cjp1*Vh(u,i,  j+1)) +
    cip1*(cjm1*Vh(u,i+1,j-1) + cj0*Vh(u,i+1,j) + cjp1*Vh(u,i+1,j+1));
    }
```

Australian National University

OpenMP parallelization results (CPU)



Conclusions:

(1) When doubling the thread number, the time elapsed reduced nearly 50%, which means we have parallelized almost all the time consuming part.

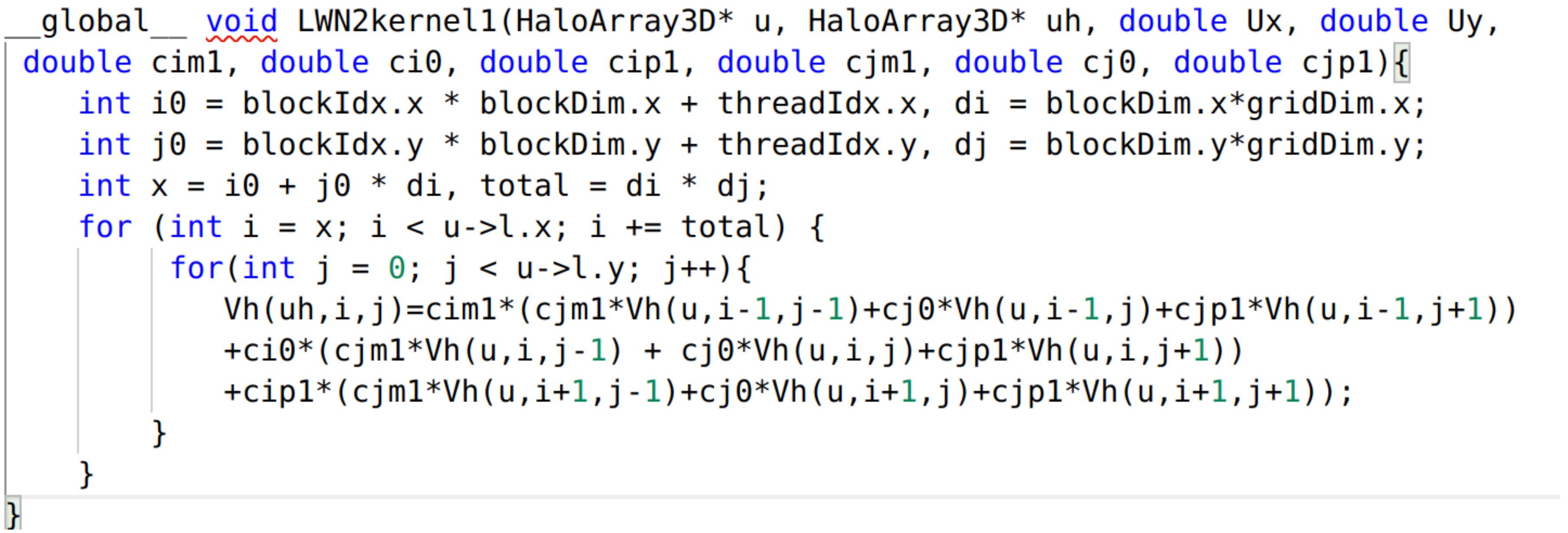
(2) When threads number increases, the time reduced ratio decreases, might result from cache misses.

CUDA implementation

- Row-wise: Every thread calculate elements in a row
- Column wise: Every thread calculate elements in a column
- Block-wise: Every thread calculate elements in a block
- Element-wise: Every thread calculate one element in a block and then calculate element in the same position in the next block

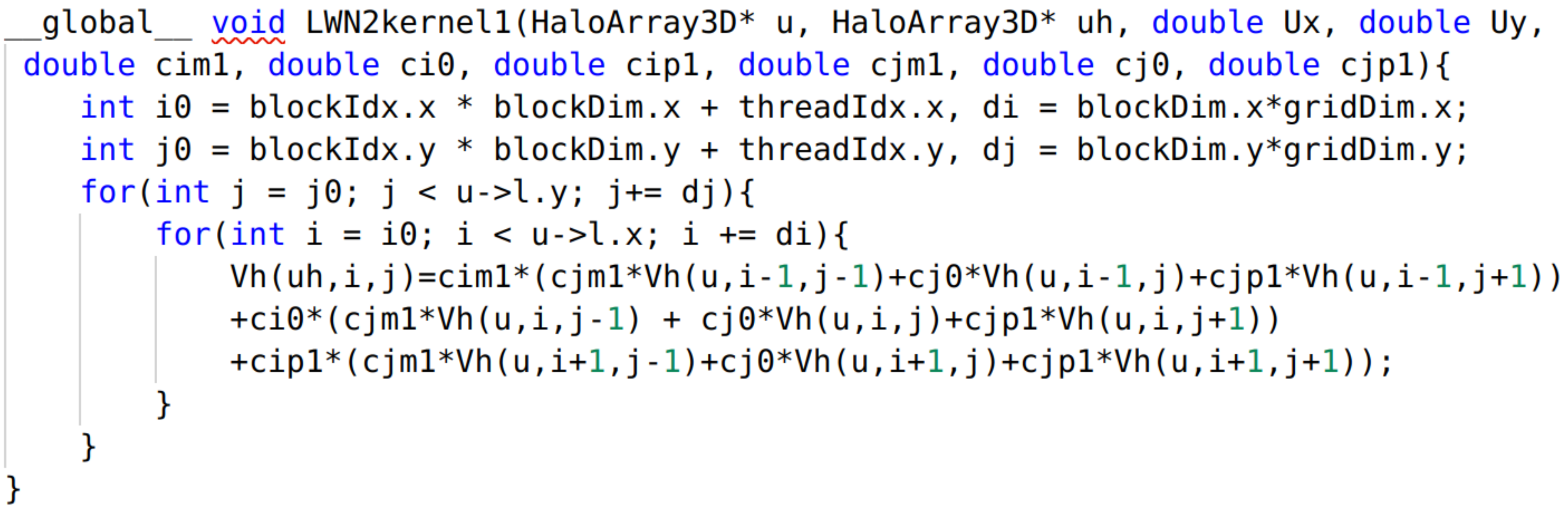
Australian National University

CUDA implementation: Row-wise



```
__global__ void LWN2kernel1(HaloArray3D* u, HaloArray3D* uh, double Ux, double Uy,
 double cim1, double ci0, double cip1, double cjm1, double cj0, double cjp1){
    int i0 = blockIdx.x * blockDim.x + threadIdx.x, di = blockDim.x*gridDim.x;
    int j0 = blockIdx.y * blockDim.y + threadIdx.y, dj = blockDim.y*gridDim.y;
    int x = i0 + j0 * di, total = di * dj;
    for (int i = x; i < u->l.x; i += total) {
        for(int j = 0; j < u->l.y; j++){
           Vh(uh,i,j)=cim1*(cjm1*Vh(u,i-1,j-1)+cj0*Vh(u,i-1,j)+cjp1*Vh(u,i-1,j+1))
           +ci0*(cjm1*Vh(u,i,j-1) + cj0*Vh(u,i,j)+cjp1*Vh(u,i,j+1))
           +cip1*(cjm1*Vh(u,i+1,j-1)+cj0*Vh(u,i+1,j)+cjp1*Vh(u,i+1,j+1));
        }
    }
}
```

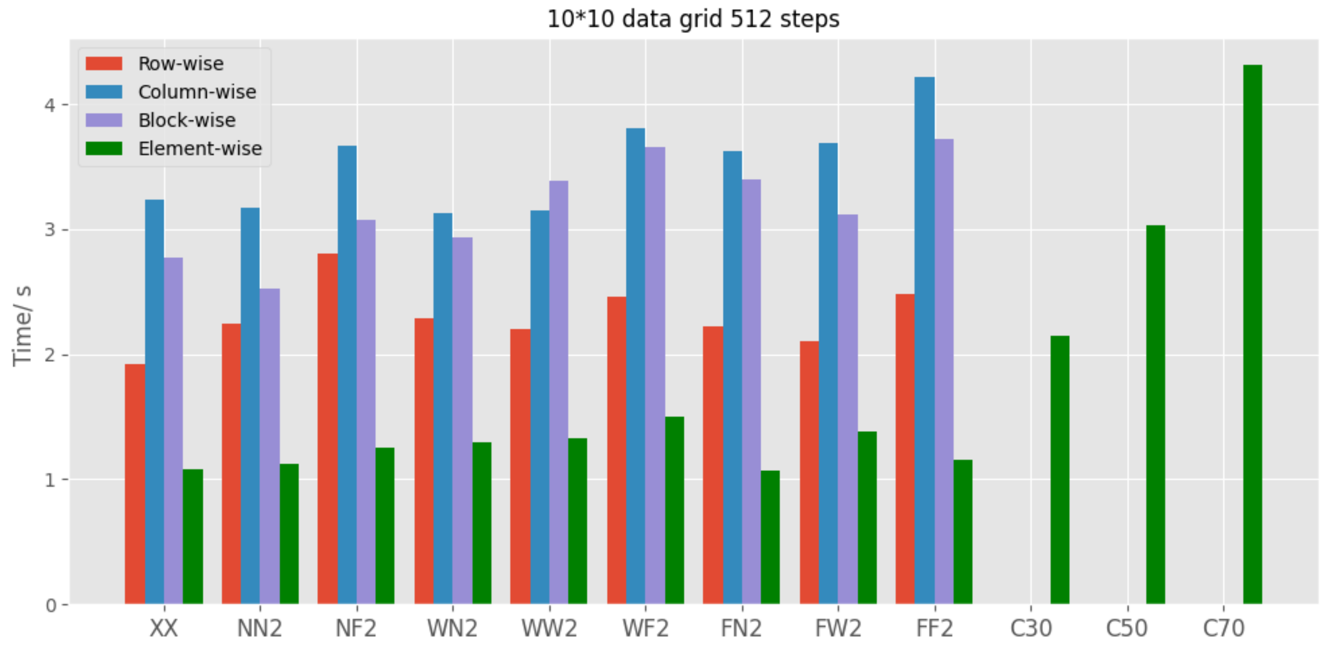
CUDA implementation: Element-wise



```
__global__ void LWN2kernel1(HaloArray3D* u, HaloArray3D* uh, double Ux, double Uy,
 double cim1, double ci0, double cip1, double cjm1, double cj0, double cjp1){
    int i0 = blockIdx.x * blockDim.x + threadIdx.x, di = blockDim.x*gridDim.x;
    int j0 = blockIdx.y * blockDim.y + threadIdx.y, dj = blockDim.y*gridDim.y;
    for(int j = j0; j < u->l.y; j+= dj){
        for(int i = i0; i < u->l.x; i += di){
            Vh(uh,i,j)=cim1*(cjm1*Vh(u,i-1,j-1)+cj0*Vh(u,i-1,j)+cjp1*Vh(u,i-1,j+1))
            +ci0*(cjm1*Vh(u,i,j-1) + cj0*Vh(u,i,j)+cjp1*Vh(u,i,j+1))
            +cip1*(cjm1*Vh(u,i+1,j-1)+cj0*Vh(u,i+1,j)+cjp1*Vh(u,i+1,j+1));
        }
    }
}
```

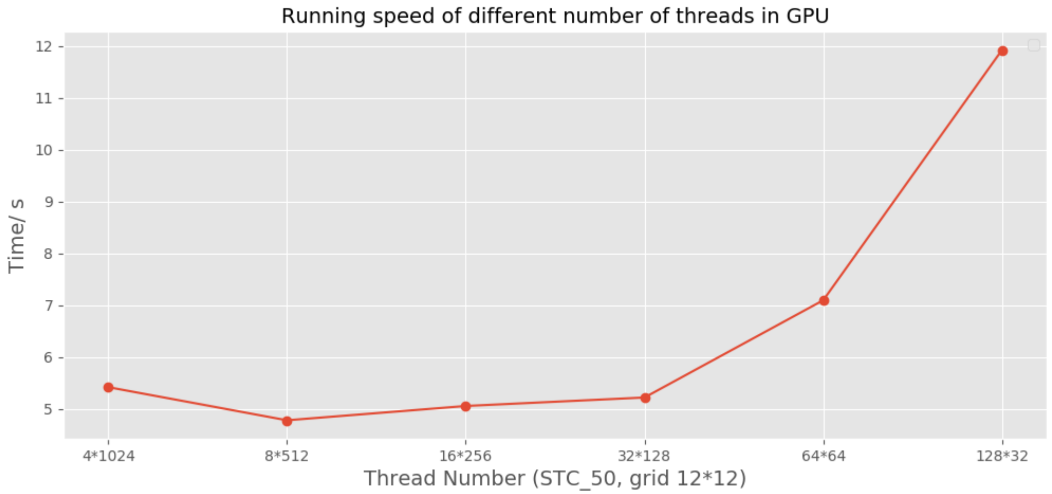
Australian National University

CUDA parallelization results (GPU)



Element-wise implementation shows best performance. Not only faster than other implementation, but also get results for combined stencils (C30, C50, C70). Threads use least resources in element-wise implementation.

CUDA parallelization results (GPU)



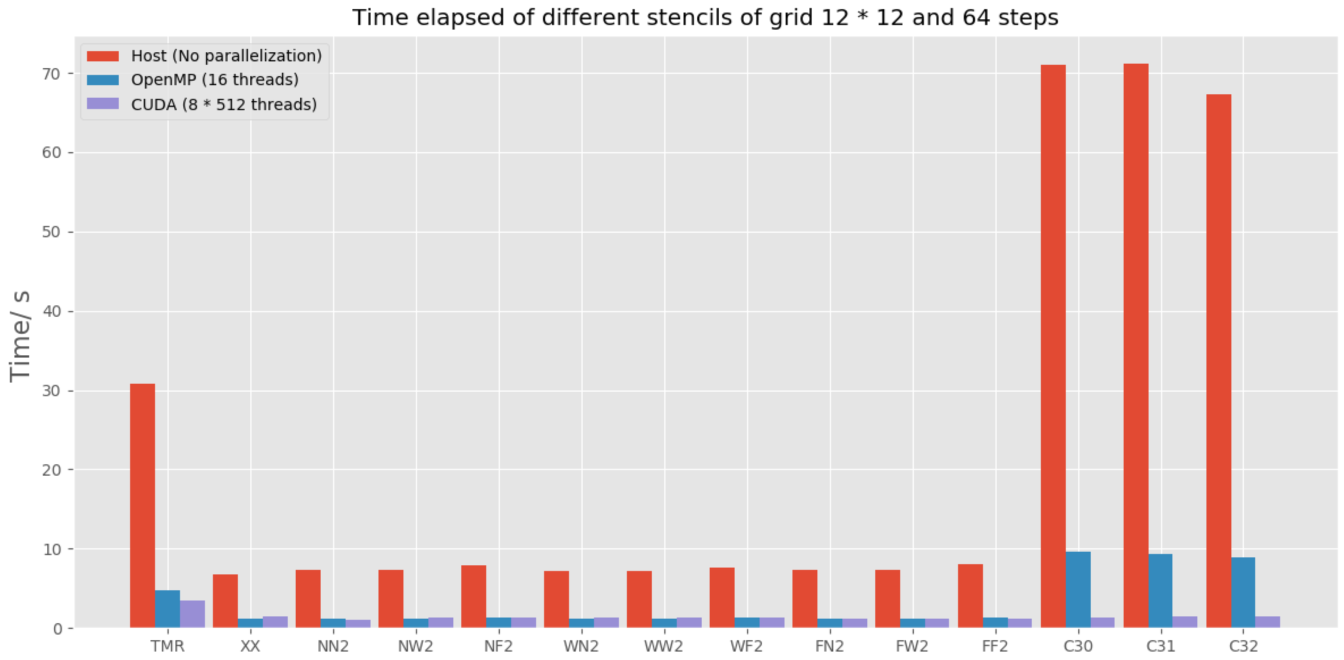
Conclusions:

(1) Thread dimensions have significant impact on program performance

(2) Thread number less than 128 in a thread block will greatly decrease running speed.

Australian National University

CUDA and OpenMP results comparison



Both OpenMP and CUDA show great ability to optimize the program. CUDA is better for parallelizing this program

For more details, you can see my project repo on gitlab:

https://gitlab.cecs.anu.edu.au/u6656110/comp8755-weather-project

My thesis is available at:

https://www.overleaf.com/read/yyywvqwjrnng

Q & A