GPU COMPUTING
LECTURE 11 - STENCIL COMPUTATIONS

Kazem Shekofteh
kazem.shekofteh@ziti.uni-heidelberg.de
Institute of Computer Engineering
Ruprecht-Karls University of Heidelberg
Inspired from lectures by Holger Fröning

STENCIL COMPUTATIONS

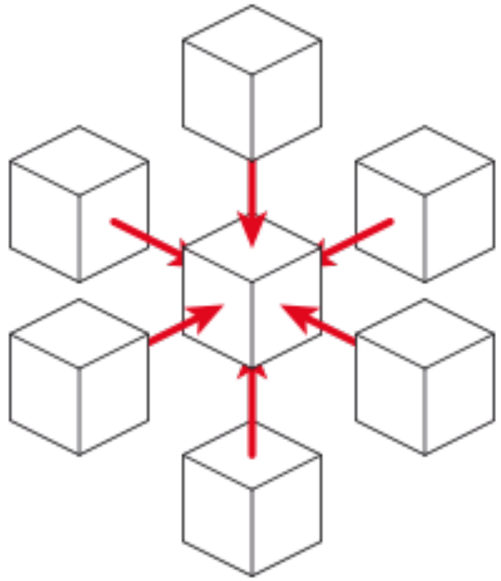
Iterative kernel that updates regular arrays based on a certain pattern

- Pattern is called stencil (6-point stencil in the example)

Applications

- Image processing (blob analysis) (1)
- Solving Partial Differential Equations (PDE) (2)
- Computational Fluid Dynamics (CFD) for science and engineering

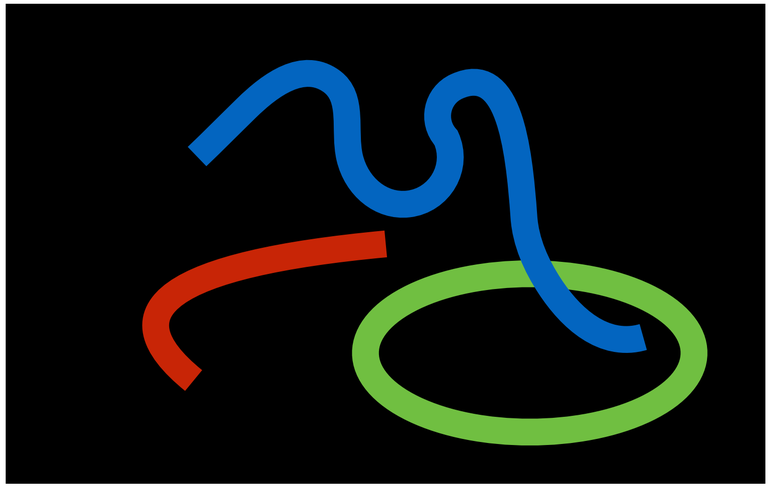
For irregular grids see Finite Element Methods (FEM)



Source:wikipedia.org

IMAGE PROCESSING – CONNECTED COMPONENT LABELING

CONNECTED COMPONENT LABELING

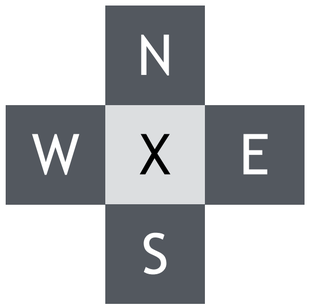


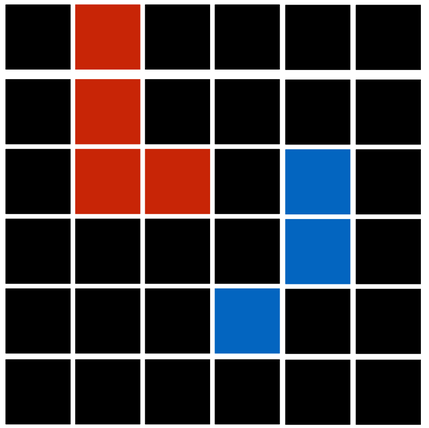
Objective: identify connected areas in this image

Motivation: more meaningful patterns or areas that are easier to process

In the output, each component (segment) is identified by a different color (value)

4-way or 8-way connectivity





CONNECTED COMPONENT LABELING

Threshold: convert color to b/w bitmap

Label: walk sequentially through the pixels

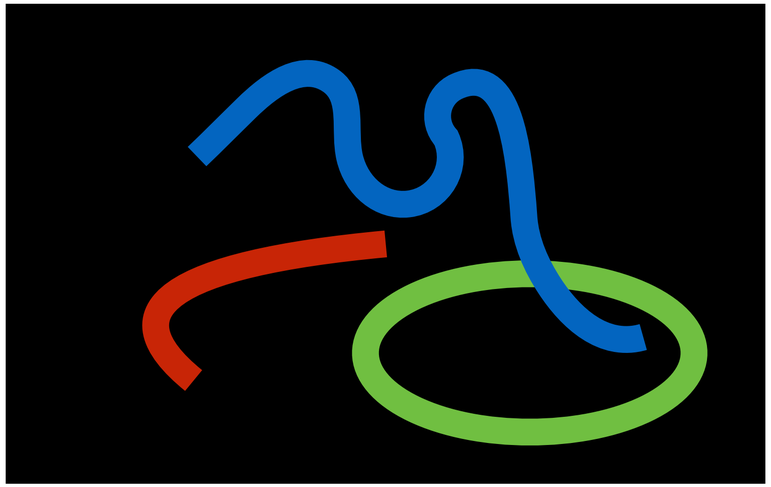
- For each pixel, use a stencil to identify neighbour pixels and their segment label
- If valid segments are present, use lowest label
- If not, assign a new segment label

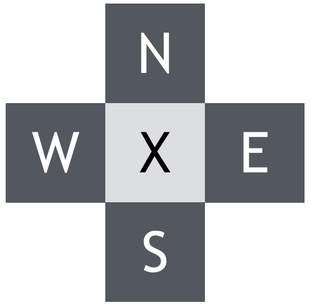
Merge: merge labels to reduce them to a minimal set

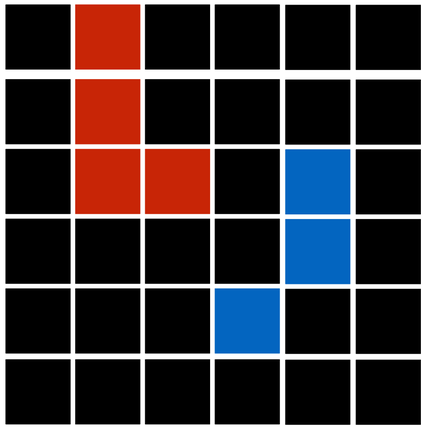
Output: each pixel labeled to the segment it belongs to

Objectives

- As few segments as possible -> less iterations -> **sequential** processing
- As **parallel** as possible -> more iterations but independency







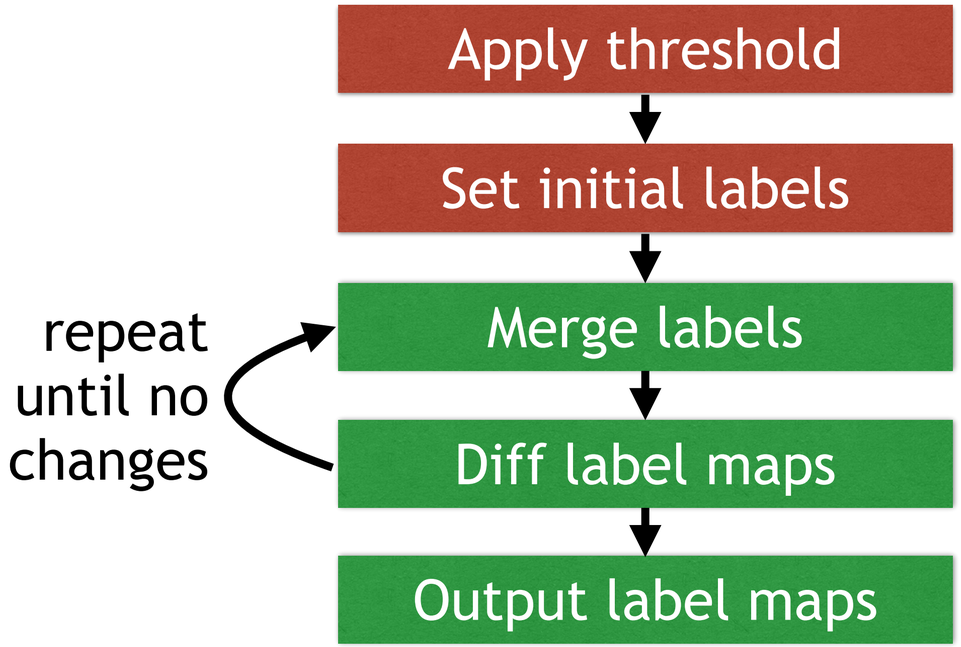
THRESHOLD/INITIAL LABELING

Threshold to b/w image $x = (\frac{r+g+b}{3} > thres)?1:0$

For GPU, set initial labels in parallel

- Exploits parallelism, but creates more labels
- Later reduction necessary

```
if ( x != 0 ) {
  if ( tid[N] ) return tid[N];
  if ( tid[W] ) return tid[W];
  if ( tid[S] ) return tid[S];
  if ( tid[E] ) return tid[E];
  // more for the 8pt stencil
  return getNewTid();
}
return 0;
```



MERGING/DIFF

Merge labels

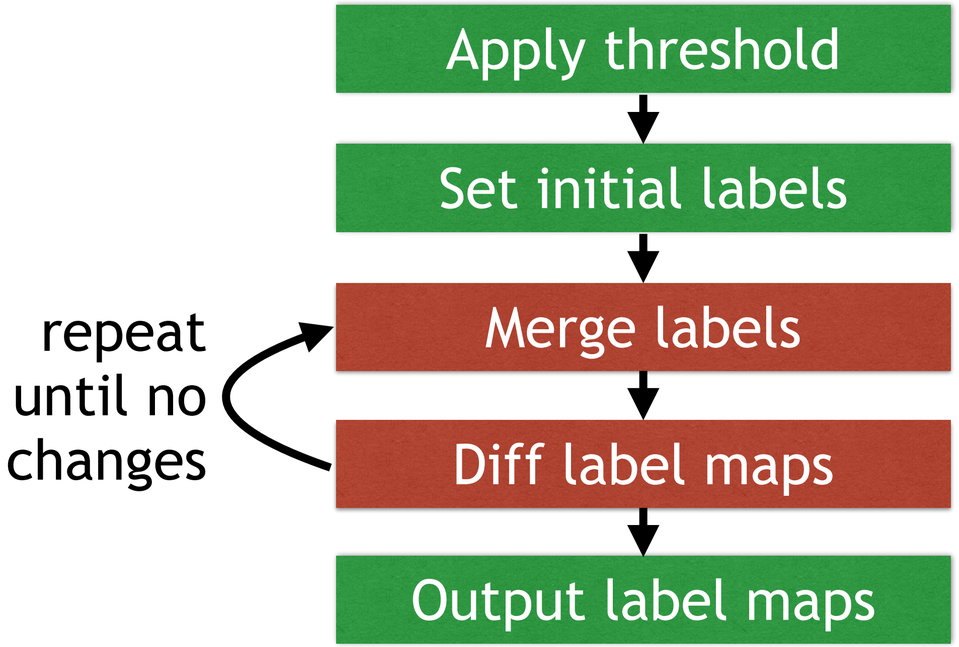
```
if ( tid[N] < x ) return tid[N];
if ( tid[W] < x ) return tid[W];
if ( tid[S] < x ) return tid[S];
if ( tid[E] < x ) return tid[E];
//also NW,NE,SW,SE for the 8pt stencil

return x;
```

Diff labels

Either compare to shadow array

Or set flag upon changes



NAIVE MERGING

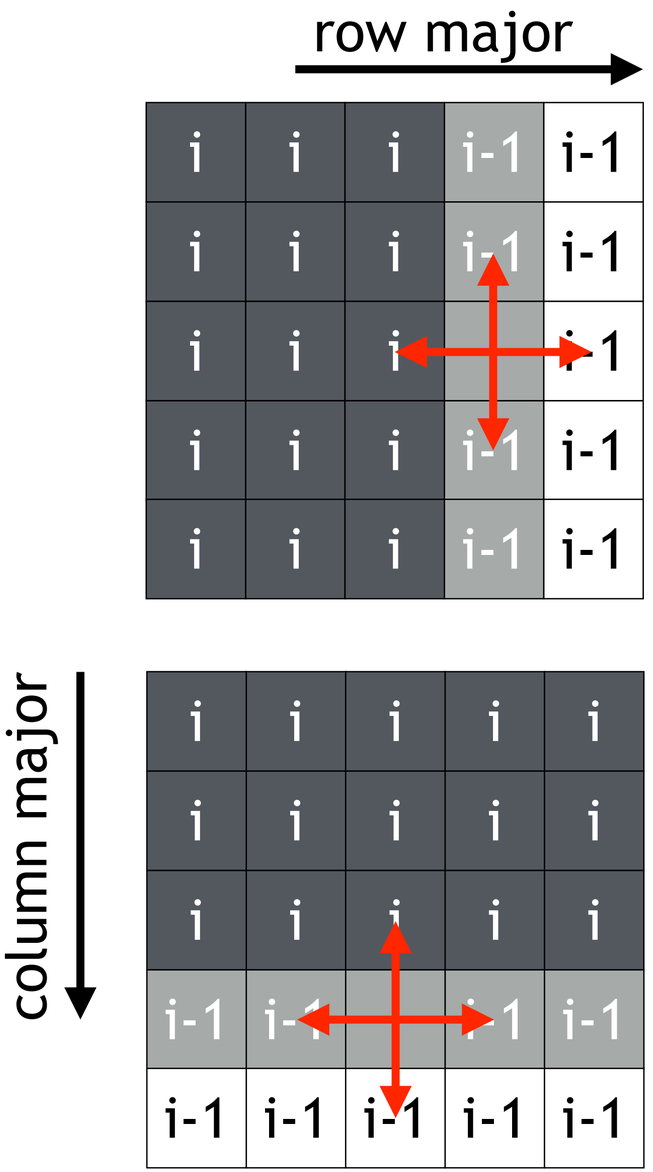
Iteration `i` (already updated), previous iteration `i-1`

Marching order is a trade-off between

- Row major order: more cache hits (intra-thread locality)
- Column major order: better coalescing (inter-thread locality, typically more important)

Marching order observes values from different iterations (`i, i-1`)

- One thread per pixel
- Each thread reads {N,W,S,E}



DIAGONAL MERGING

Using diagonals

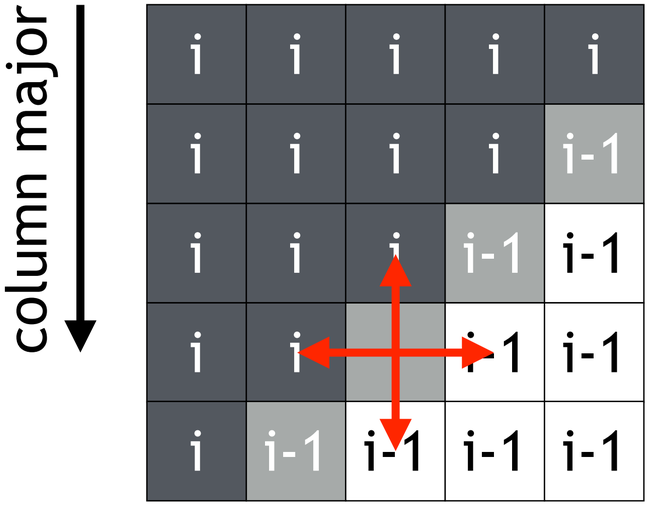
- Diagonal better as more elements are already updated

-> less iterations

- Still wasteful as elements are read multiple times and no re-use is exploited

Similarities to stencil computations

- Gauss-Seidel: optimal for sequential processing (less iterations)
- Jacobi: better for parallel processing (less dependencies)



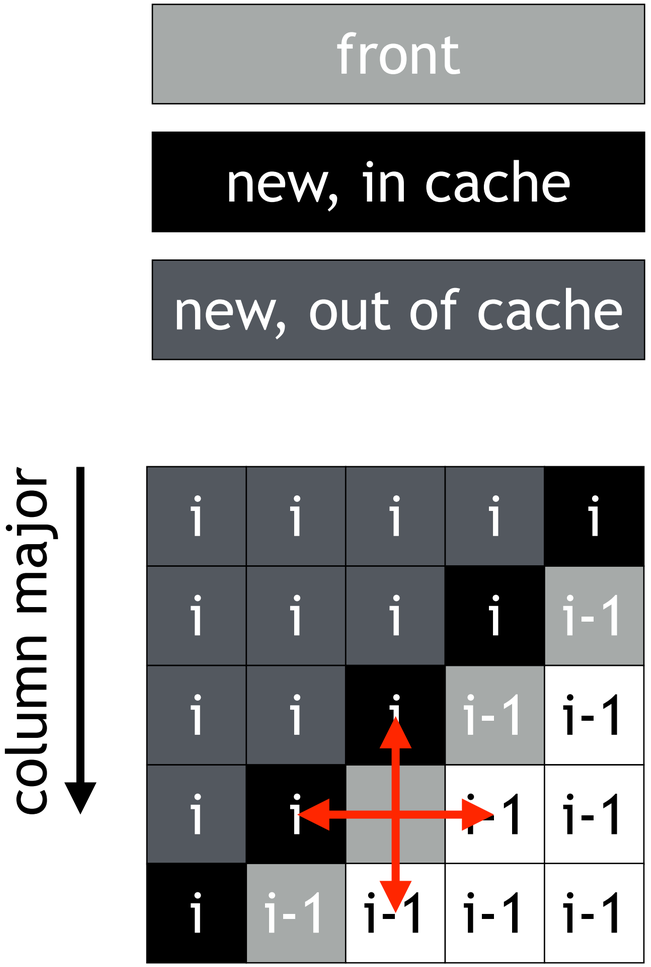
WAVEFRONT MERGING

Wavefronts

- Each thread processes an element
- Storing previous elements in shared memory to reduce memory contention

Here: stencil distance of 1 -> keep one element in shared memory

For many recent GPUs, this optimization is already close to peak



WAVEFRONT MERGING FOR LARGER STENCIL

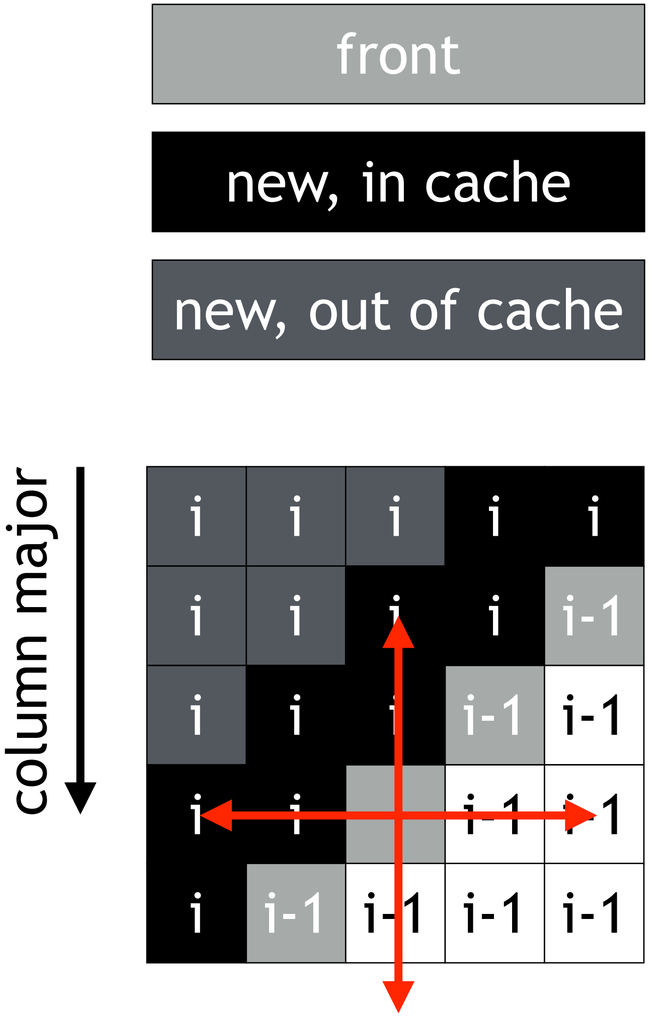
Wavefronts – cache subimages in shared memory

Loop is divided into three phases

- Fill
- Steady state
- Drain

Inefficient in the first and last phase

- Similar to pipeline fill/flush



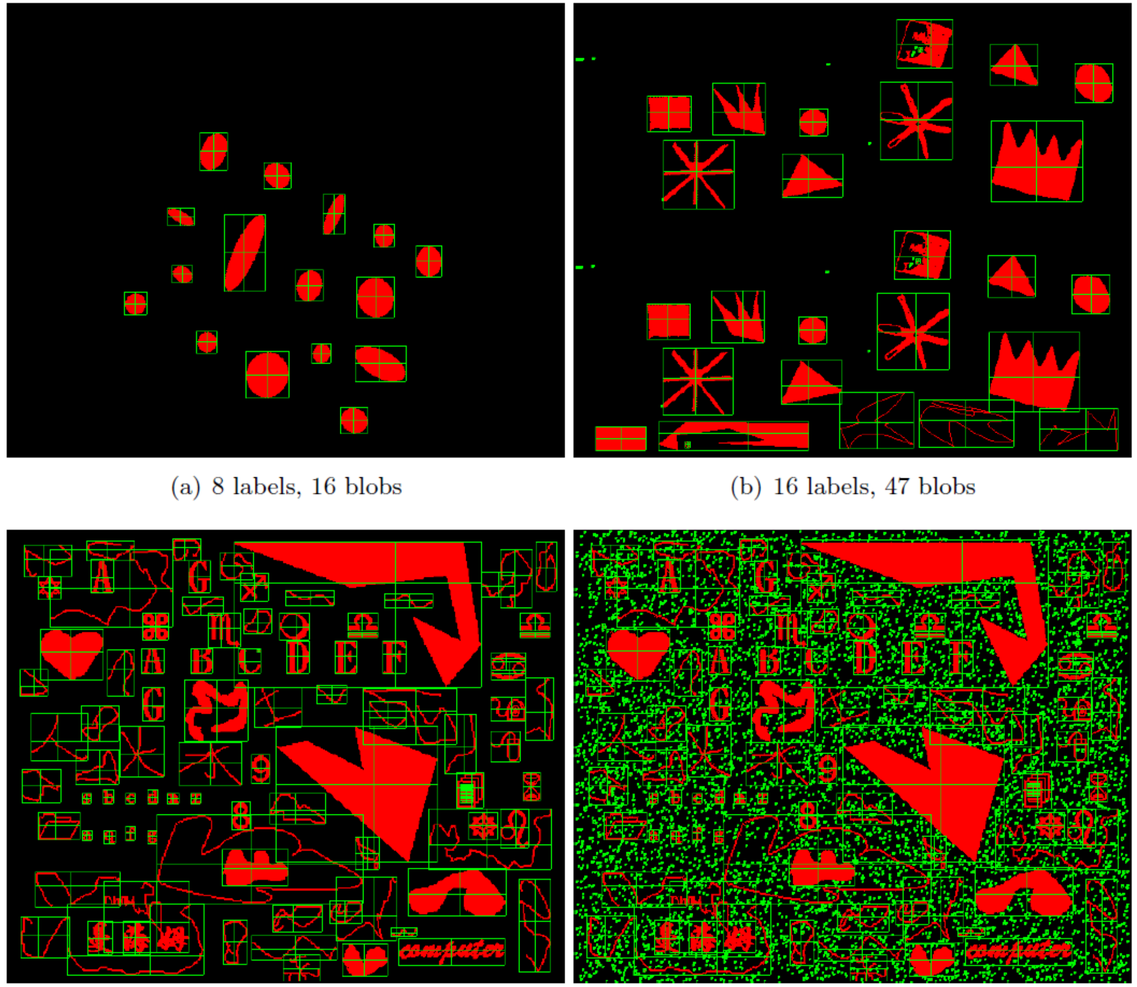
APPLICATION EXAMPLE

Tracking of infrared markers

Find center of gravity

FPGA implementation

Real-time constraints -> optimizations to reduce number of iterations



(a) 8 labels, 16 blobs

(b) 16 labels, 47 blobs

(c) 32 labels, 96 blobs

(d) 64 labels, 5759 blobs

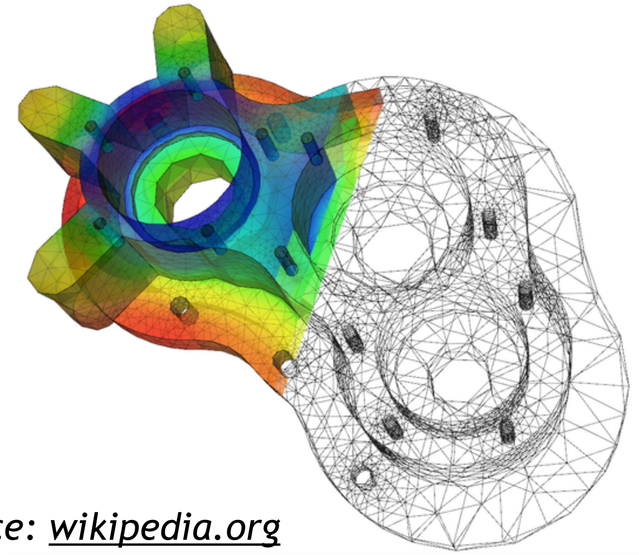
PhD thesis of Yuning Yang, University of Heidelberg, 2012

PARTIAL DIFFERENTIAL EQUATIONS

PARTIAL DIFFERENTIAL EQUATIONS (PDE)

(Partial) differential equations widely used

- Partial: function of multiple independent variables
- Engineering, physics, biology, economics, chemistry
- Heat transfer, Newtonian gravity, seismic wave propagation, electrostatics



Source: wikipedia.org

Finite-difference methods (FDM) approximate a solution using finite difference equations

- Taylor's polynomial
- Approximation of the derivative (spacing k)
- Multi-variable functions

$$\frac{d}{dx}f(x) = f_x = \lim_{k \to 0} \frac{f(x+k) - f(x)}{k}$$

$$f_x \approx \frac{f(x+k) - f(x)}{k}$$

$$\frac{d}{dx}f(x,t) = f_x = \lim_{k \to 0} \frac{f(x+k,t) - f(x,t)}{k}$$

FINITE DIFFERENCES

Forward difference is intuitive

$$f_{x,fwd} \approx \frac{f(x+k) - f(x)}{k}$$

Backwards difference is an alternative

$$f_{x,bwd} \approx \frac{f(x) - f(x-k)}{k}$$

Both have errors of O(k), likely with different signs though

-> Central difference with $O(k^2)$ by averaging both formulas

$$f_x \approx \frac{f_{x,fwd} + f_{x,bwd}}{2} = \frac{f(x+k) - f(x-k)}{2k}$$

FINITE DIFFERENCES

Multi-variable function f

$$\frac{d}{dx}f(x,t) = f_x \approx \frac{f(x+h,t) - f(x,t)}{h}$$

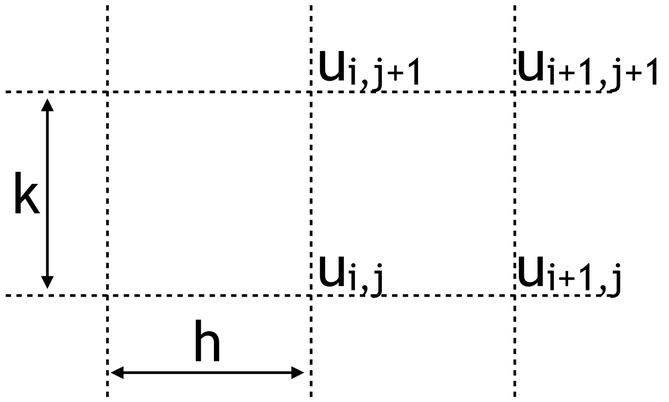
$$\frac{d}{dt}f(x,t) = f_t \approx \frac{f(x,t+k) - f(x,t)}{k}$$

Based on the differences k and h, define grid points

$$x_i = ih; i = 0, 1, 2, \ldots, \frac{L}{h} \quad t_j = jk; j = 0, 1, 2, \ldots, \frac{T}{k} \qquad x \in [0, L]; t \in [0, T]$$

I.e., time advances in steps of k, and (1D) space is discretized at evenly spaced points h

Also valid for a multi-dimensional space



THE HEAT EQUATION

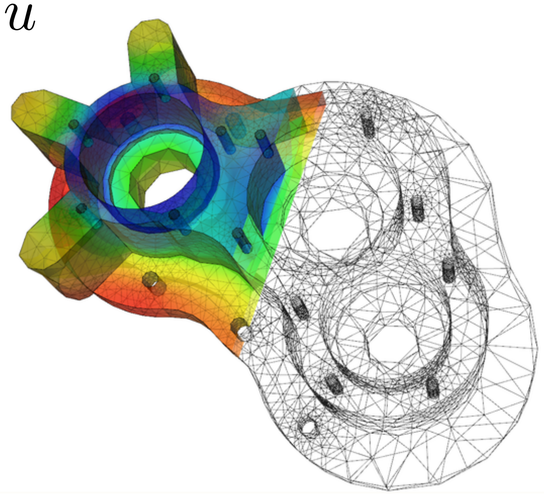
Describes the heat transfer over time

c is thermal conductivity, or how fast heat is transported through material

$$\frac{d}{dt}u - c \cdot \Delta u = 0 \qquad \Delta u = \nabla^2 u = \frac{d^2}{dx^2}u + \frac{d^2}{dy^2}u + \frac{d^2}{dz^2}u$$

For a single dimension: $u_t = c \cdot u_{xx}$

If there was a heat or chemical source: $u_t = c \cdot u_{xx} + g(x,t)$



Source: wikipedia.org

Many equations that involve 1 time derivative and 2 spatial derivatives are parabolic

The methods introduced here will work for most of them

A COMMON APPROXIMATION OF THE 2ND ORDER PARTIAL DIFFERENCE

Taylor series of f(x) at a number x_0:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \ldots = \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!}(x - x_0)^n$$

$$\Delta x = x - x_0 : f(x_0 + \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{f''(x_0)}{2!}\Delta x^2 + \frac{f'''(x_0)}{3!}\Delta x^3 + \ldots$$

Expand and approximate for both directions:

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{f''(x)}{2!}\Delta x^2$$

$$f(x - \Delta x) \approx f(x) - f'(x)\Delta x + \frac{f''(x)}{2!}\Delta x^2$$

$$f'(x)\Delta x \approx f(x) + \frac{f''(x)}{2!}\Delta x^2 - f(x - \Delta x)$$

$$f(x + \Delta x) \approx f(x) + f(x) + \frac{f''(x)}{2!}\Delta x^2 - f(x - \Delta x) + \frac{f''(x)}{2!}\Delta x^2$$

$$f''(x) \approx \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2}$$

EXPLICIT METHOD

Replace the heat equation with the difference equation $u_t = c \cdot u_{xx}$

Forward difference for t, central difference for x

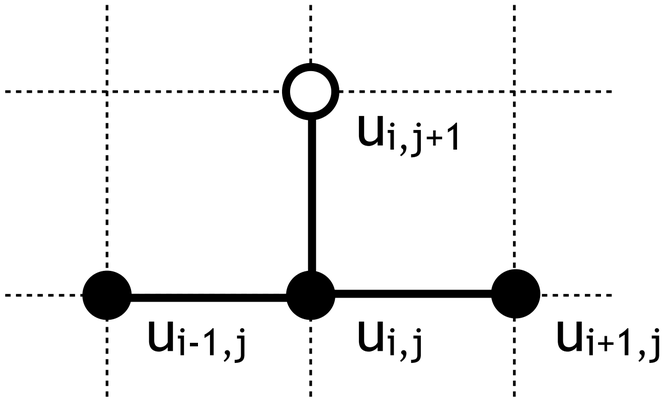
$$u_t = \frac{u(x, t+k) - u(x,t)}{k}; u_{xx} = \frac{u(x+h,t) - 2u(x,t) + u(x-h,t)}{h^2}$$

Discretize based on $u_{i,j} = u(x_i, t_j)$

Solve this for the next time step j+1

$$\frac{u_{i,j+1} - u_{i,j}}{k} = c\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

$$u_{i,j+1} = ru_{i-1,j} + (1-2r)u_{i,j} + ru_{i+1,j}; r = \frac{ck}{h^2}$$



Explicit methods produce a formula for time step j+1 based on step j

LIMITATIONS OF THE EXPLICIT METHOD

$$u_{i,j+1} = ru_{i-1,j} + (1-2r)u_{i,j} + ru_{i+1,j}; r = \frac{ck}{h^2}$$

Increasing time or spatial resolution

-> Amount of grid points increases -> more computations

Increasing time or spatial resolution introduces numerical errors

-> solution can be unbounded unless time steps are small

Original formula written using a matrix equation (spatial vector)

$$u_{j+1} = Au_j$$

Suppose U_j is the vector of correct values at time step t_j, and E_j is the error of the approximation. Then:

$$u_{j+1} = AU_j + AE_j$$

Eigenvalue of A^k < 1 -> error diminishes over time, > 1 -> error increases

$$u_{j+k} = A^kU_j + A^kE_j$$

It can be shown that the eigenvalue is smaller than 1 for:

$$r = \frac{ck}{h^2} < 0.5 \text{ or } k < \frac{h^2}{2c}$$

h^2 is the problem if higher resolutions are required, as k has to be very small then

-> Otherwise stability problems

FROM EXPLICIT TO IMPLICIT METHOD

Explicit methods compute a single value for each data point at t=t+k

- Require impractically small time steps to keep the error in the result bounded

Implicit methods instead compute all points at t=t+k by solving a system of equations

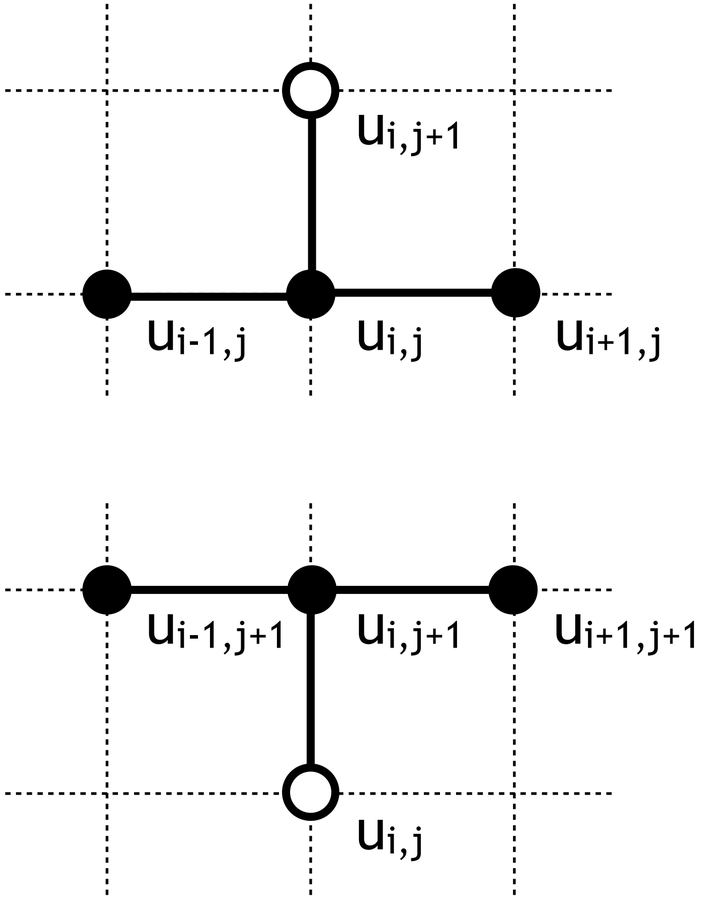
- Each data point at time t provides an equation for time t+k
- Assume m space points, there are m-2 equations and m-2 unknown values; therefore it can be solved with standard methods

-> Much more computationally intensive

- But also much more numerically stable

Computationally intensive -> GPUs

$$k < \frac{h^2}{2c}$$



IMPLICIT METHOD

$$u_{i,j+1} = ru_{i-1,j} + (1-2r)u_{i,j} + ru_{i+1,j}; r = \frac{ck}{h^2}$$

Explicit

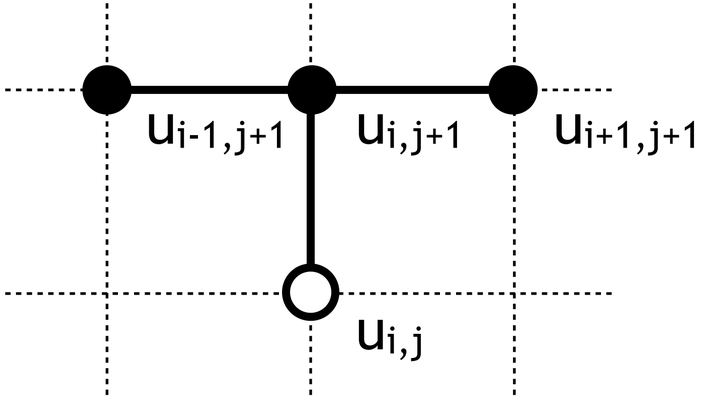
If we approximate u_{xx} and u_t rather at t_{j+1} instead of t_j, and using a backwards difference for u_t, we obtain:

$$u_{i,j} = -ru_{i-1,j+1} + (1+2r)u_{i,j+1} - ru_{i+1,j+1}; r = \frac{ck}{h^2}$$

Implicit

This stencil defines three output points for one input point

- Thus, for each output point we need 3 equations
- Relationship is linear
- Solving a linear equation system
- Matrix multiply important here



IMPLICIT METHOD

$$u_{i,j} = -ru_{i-1,j+1} + (1+2r)u_{i,j+1} - ru_{i+1,j+1}; r = \frac{ck}{h^2}$$

Solving the linear equation system

Boundary condition b

$$u_j = B \cdot u_{j+1} - r \cdot b_{j+1}$$

$$B \cdot u_{j+1} = u_j + r \cdot b_{j+1}$$

$$B = \begin{bmatrix} 1+2r & -r & & & & \\ -r & 1+2r & -r & & & \\ & \ddots & \ddots & \ddots & & \\ & & -r & 1+2r & -r & \\ & & & -r & 1+2r & \end{bmatrix}$$

Eigenvalue of B always lower than 1 -> no stability problems

Not shown here

THE HEAT EQUATION – ERRORS

Both explicit and implicit method make errors

- $O(h^2)$ error in approximating u_{xx}, $O(k)$ error in approximating u_t
- $O(h^2+k)$ total
- Stability of implicit method allows arbitrarily large k, but to maintain accuracy we need $k \sim h^2$

Crank-Nicholson Method

- Combines both methods by weighted averages of u_{xx} at j and j+1

$$\frac{u_{i,j+1}-u_{i,j}}{k} = \frac{\beta c}{h^2}(u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}) + \frac{(\beta-1)c}{h^2}(u_{i+1,j}-2u_{i,j}+u_{i-1,j})$$

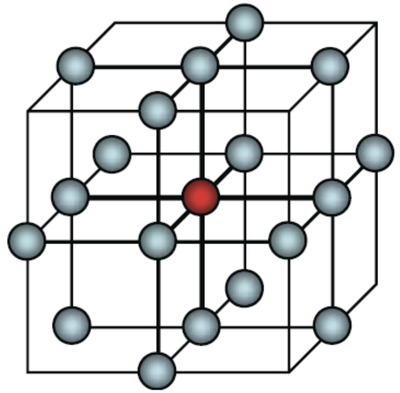
- $k \sim h$ now possible, and optimal choice leads to $O(h^6)$ error

For an accuracy of 4 digits, 1M points for the implicit method, while only about 560 points for CNM

APPLICATION EXAMPLE: POISSON'S EQUATION

Poisson's equation is a PDE of elliptic type, widely used in mechanical engineering and theoretical physics

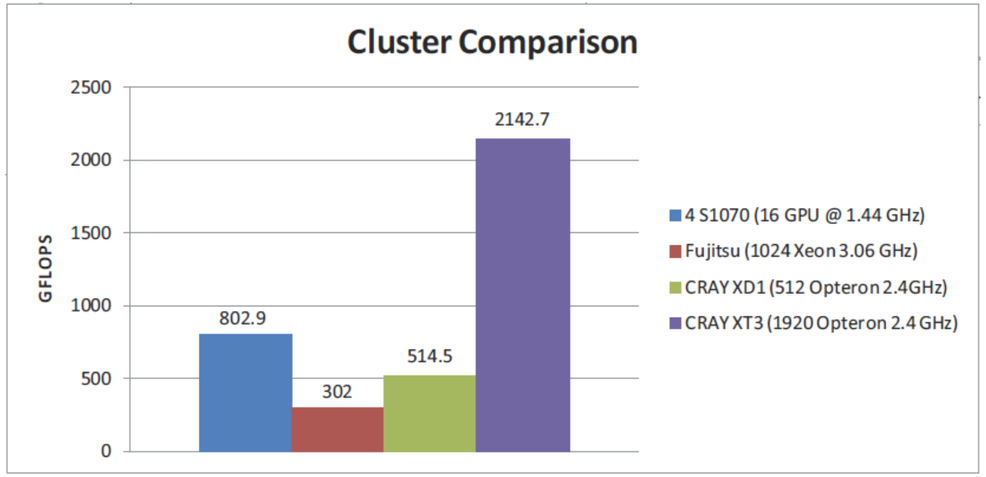
$$\Delta \cdot \Phi = f$$



E. Phillips, M. Fatica, Implementing the Himeno benchmark with CUDA on GPU clusters, in 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)

- To describe the potential field caused by a given charge or mass density distribution
- With the potential field known, one can then calculate the associated gravitational or electrostatic field
- Newtonian gravity, electrostatics, …

The Himeno benchmark focusses on the solution using a 19-point stencil



LIMITS

Many equations solvable as PDE, unless

- High dimensionality
- Highly nonlinear structures

We often find n-Body methods for such cases

> ”… gravitational Vlasov-Poisson equation, a six-dimensional PDE for the Liouville flow of the phase space probability distribution function, with gravitational potential arises self-consistently from the Poisson equation.”
>
> Salman Habib, et al. 2013. HACC: extreme scaling and performance across diverse architectures. International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13). DOI: https://doi.org/10.1145/2503210.2504566

PERFORMANCE OPTIMIZATIONS

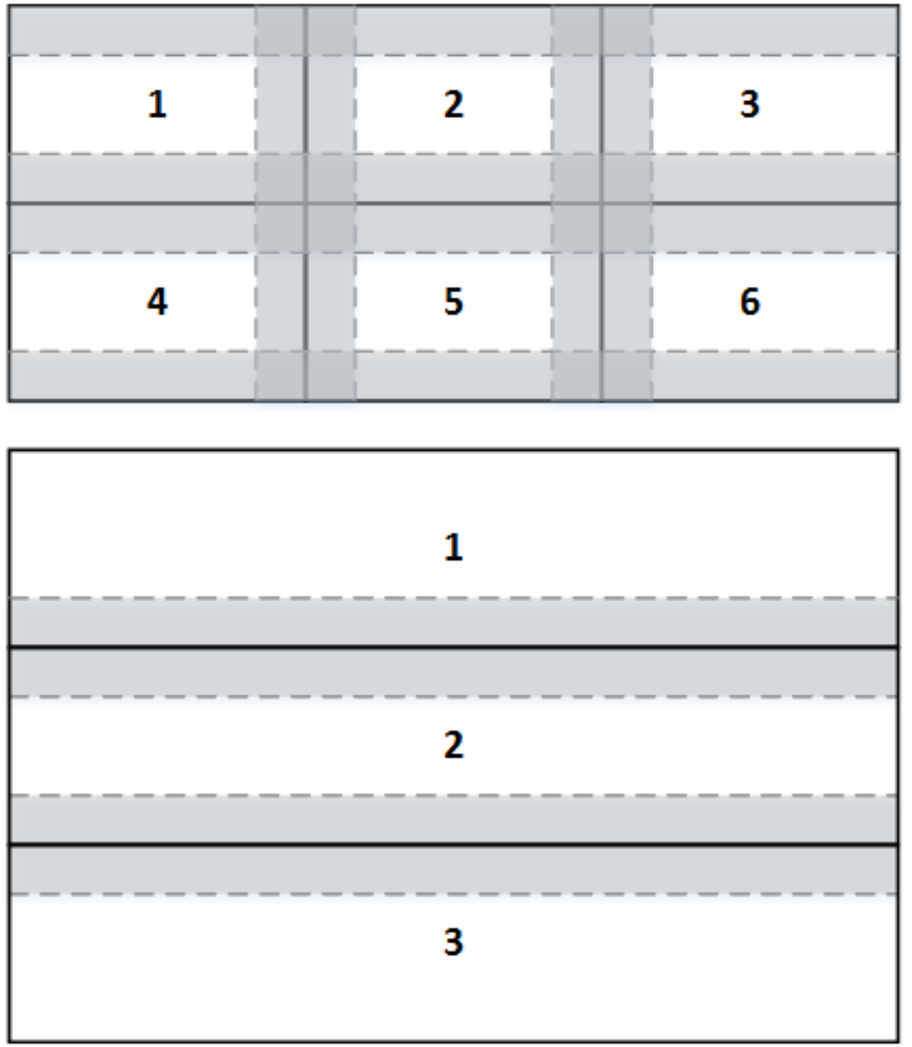
PARALLELIZATION OF STENCIL CODES

$$u_{i,j+1} = ru_{i-1,j} + (1-2r)u_{i,j} + ru_{i+1,j}; r = \frac{ck}{h^2}$$

- Stencil codes are memory-bound
  - Computational intensity
  - E.g. explicit method: 5 flops, 3 elements each 4B -> 5/12
- Two main questions
  - How to partition the data among the threads
    - Obviously domain decomposition
    - 2D/3D thread blocks?
  - How to leverage shared memory as a scarce resource in the optimal way?
- These two questions are closely related

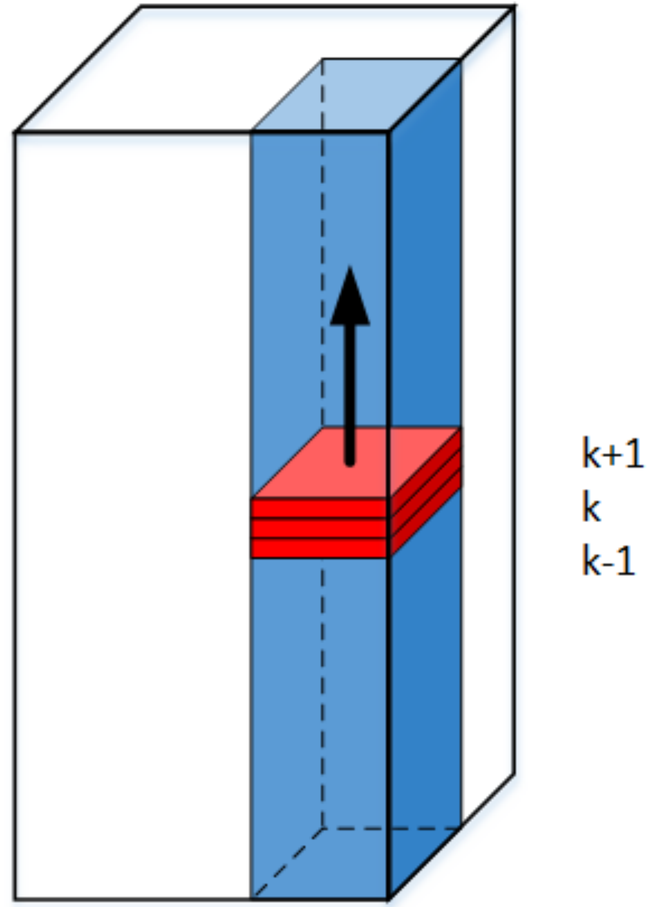
PARALLELIZATION OF STENCIL CODES

- Partitioning of a 2D data structure
  - Overlap area is called halo
- 2D partitioning
  - Vertical halos are poorly aligned in memory
- 1D partitioning
  - Only horizontal
- Surface-to-volume effects
- Communication cost depending on data layout?
  - Shared memory: likely no
  - Distributed memory: yes



PARALLELIZATION OF STENCIL CODES

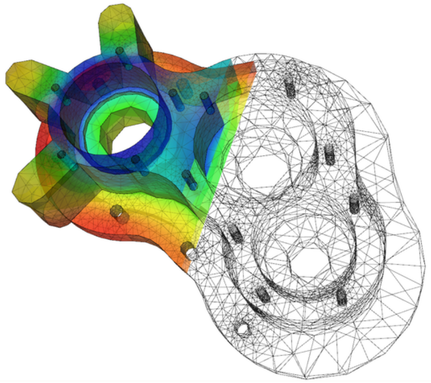
- Shared memory use
  - Minimal usage -> multiple thread blocks per SM
- Marching planes
  - Only keep 3 planes in shared memory
  - Cycling buffer as we march along a direction
- Then: Z-direction of the data block virtually unlimited
- Further optimizations: texture memory
  - Reduced instruction overhead



Marching through a 3-dimensional data array

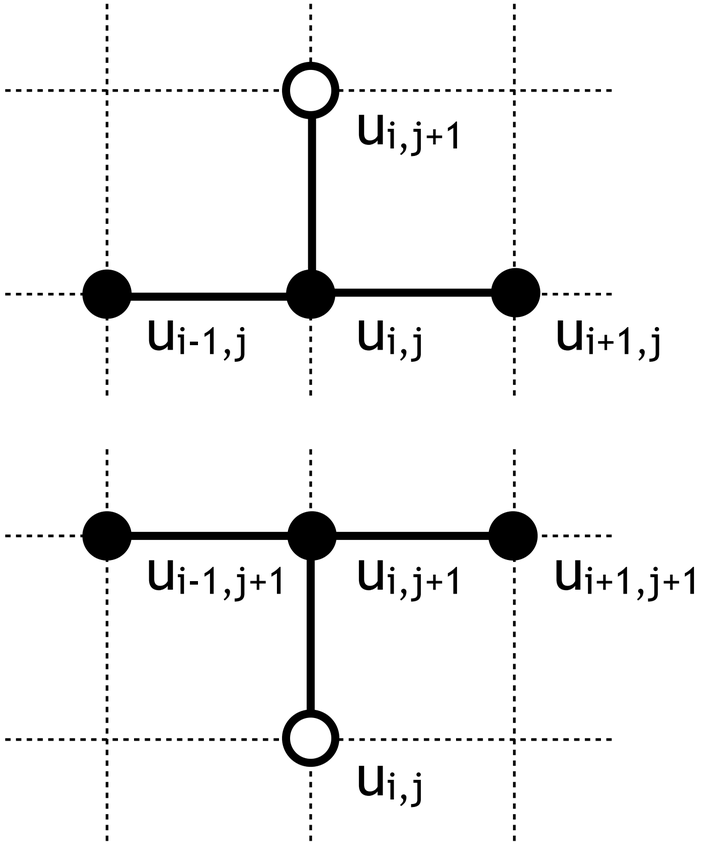
WRAPPING UP

SUMMARY



Source: wikipedia.org

- Stencil codes as prime example for local communication
  - Nearest-neighbor references only (local communication depending on stencil size)
  - Naive/slightly optimized implementations easy
- Explicit and implicit methods
  - Compute complexity vs. numerical stability
- Highly regular computation, little amount of data reuse
  - Memory-bound and little data reuse => mind the memory access performance

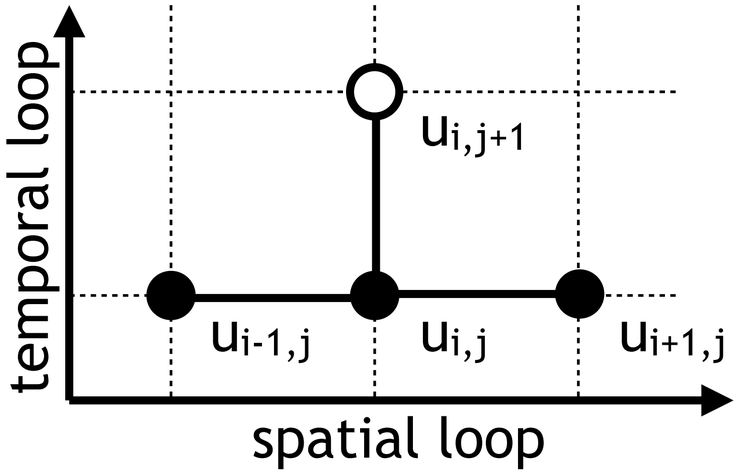


APPENDIX: TEMPORAL BLOCKING

TEMPORAL BLOCKING (TB)

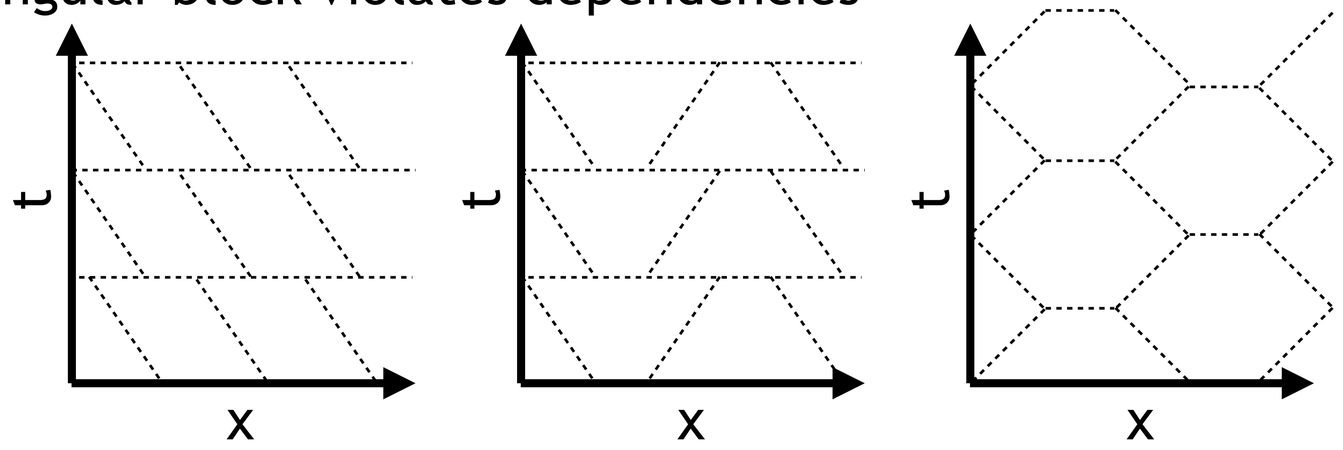
Improve locality by temporal blocking

- For a subdomain, perform multiple updates (bt) at once, then proceed to next subdomain
- bt: temporal block size



Skewed block shape is required due to dependencies

- Simple rectangular block violates dependencies



wavefront trapezoid diamond

TEMPORAL BLOCKING USING TRAPEZOID SHAPE

```
// Temporal blocking using trapezoid shape
for (  t1 =  ceild ( -N-29, 32 );
       t1 <= floord ( T-2, 32 );
       t1 ++ )

   for (  t2 =  max ( t1, -t1-1 );
          t2 <= min3 ( floord ( -16*t1+T-1, 16),
                       floord ( 16*t1+N+13,16 ),
                       floord ( T+N-3, 32 ) );
          t2 ++ )

      for (  t3 = max4  ( 0, 16*t1+16*t2, 32*t1+1, 32*t2-N+2 );
             t3 <= min4 ( T-1, 32*t2+30, 16*t1+16*t2+31, 32*t1+N+29 );
             t3 ++)

         lbv = max3 ( 32*t2, t3+1, -32*t1+2*t3-31 );
         ubv = min3 ( -32*t1+2*t3, 32*t2+31, t3+N-2 );
         for ( t4 = lbv; t4 <= ubv; t4 ++)
            A[t3+1][(-t3+t4)] = ( A[t3][(-t3+t4)-1] +
                                  A[t3][(-t3+t4)] +
                                  A[t3][(-t3+t4)+1] )
                                / 3;
```

```
// Simple stencil in 1D
for ( t = 0; t < T; t++ )
   for ( x = 1; x < N-1; x++ )
      A[t+1][x] = ( A[t][x-1] +
                    A[t][x] +
                    A[t][x+1])
                    * c;
```