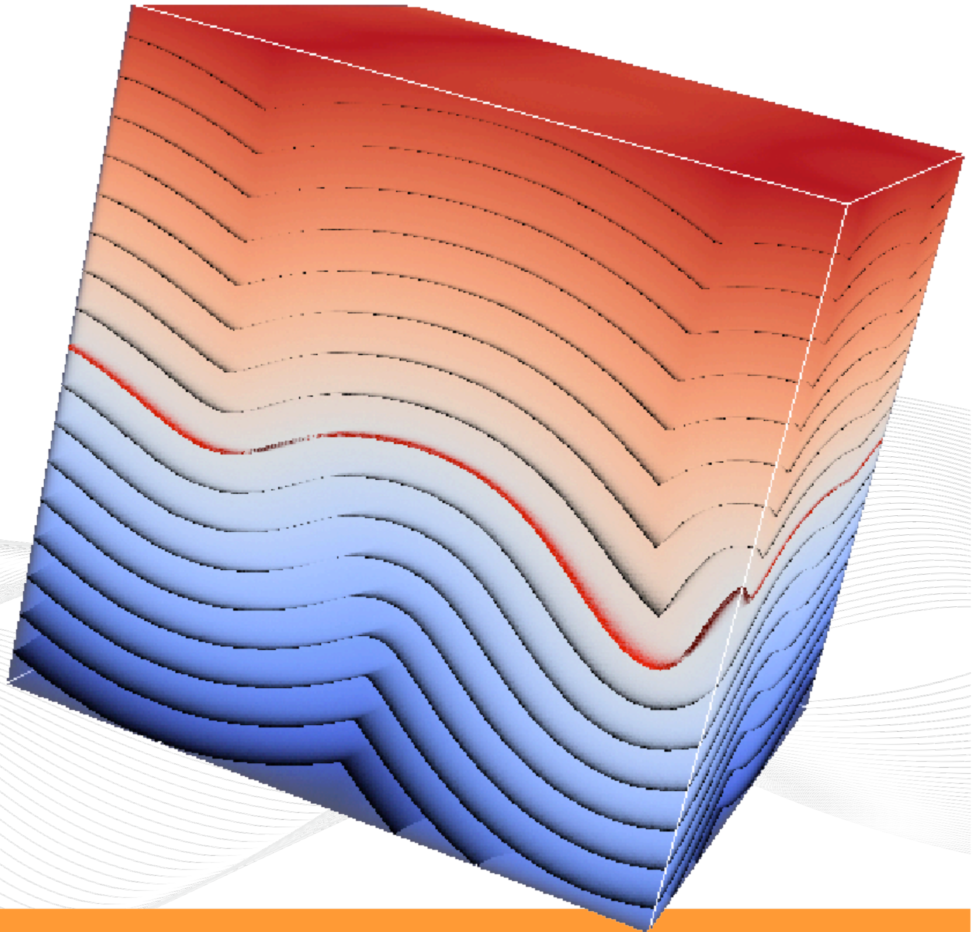
Massively Parallel Front Propagation

For Simulations of Geological Folds

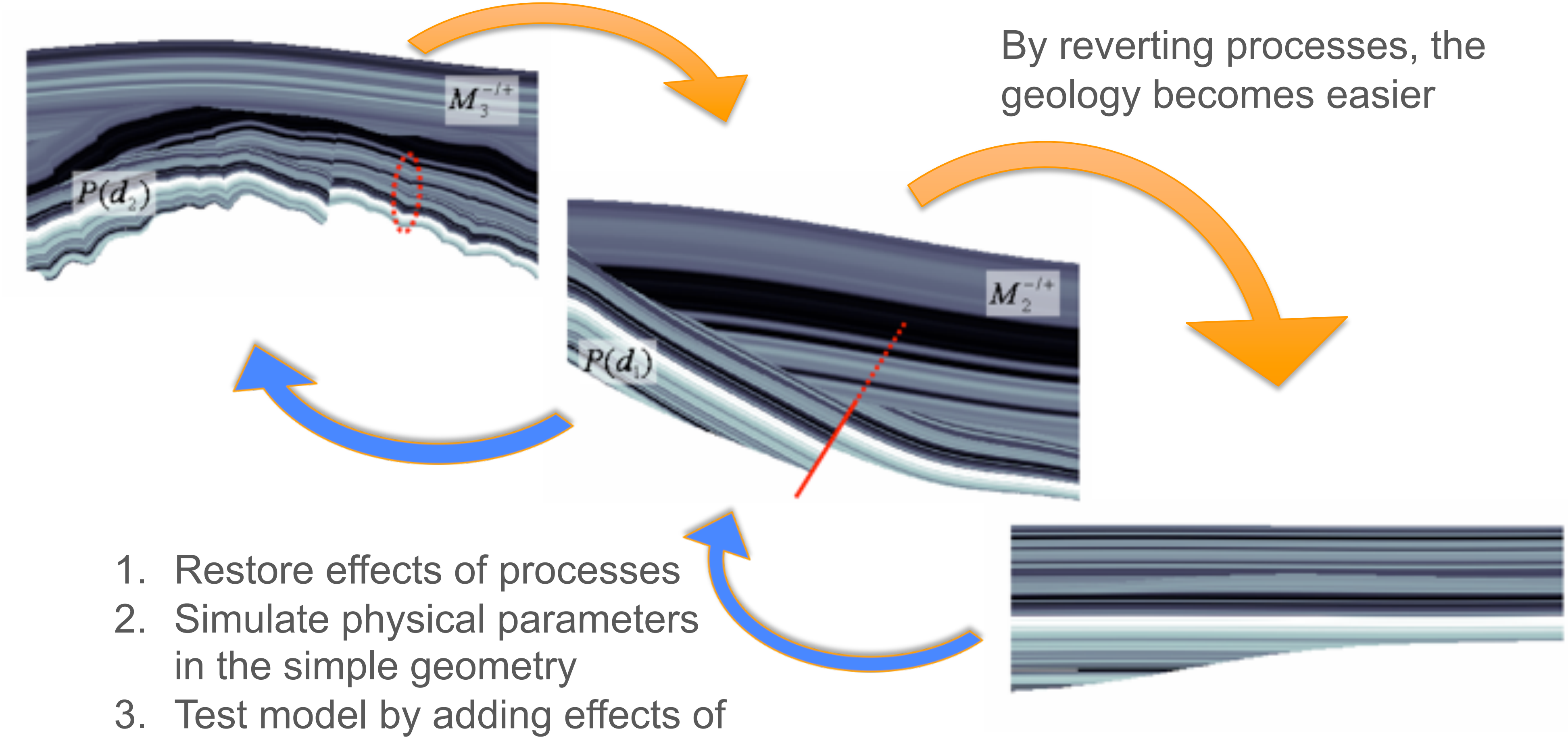
Tor Gillberg

Simula Research Laboratory AS
and Kalkulo AS

19 January 2011



Restoration made practical by Statoil and Kalkulo



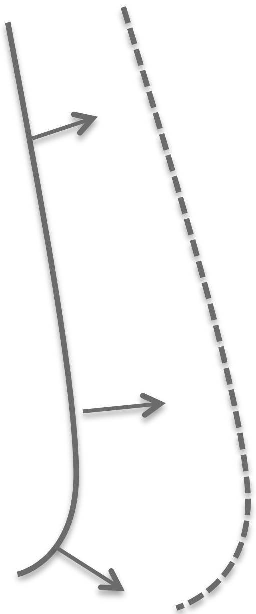
By reverting processes, the geology becomes easier

1. Restore effects of processes
2. Simulate physical parameters in the simple geometry
3. Test model by adding effects of processes

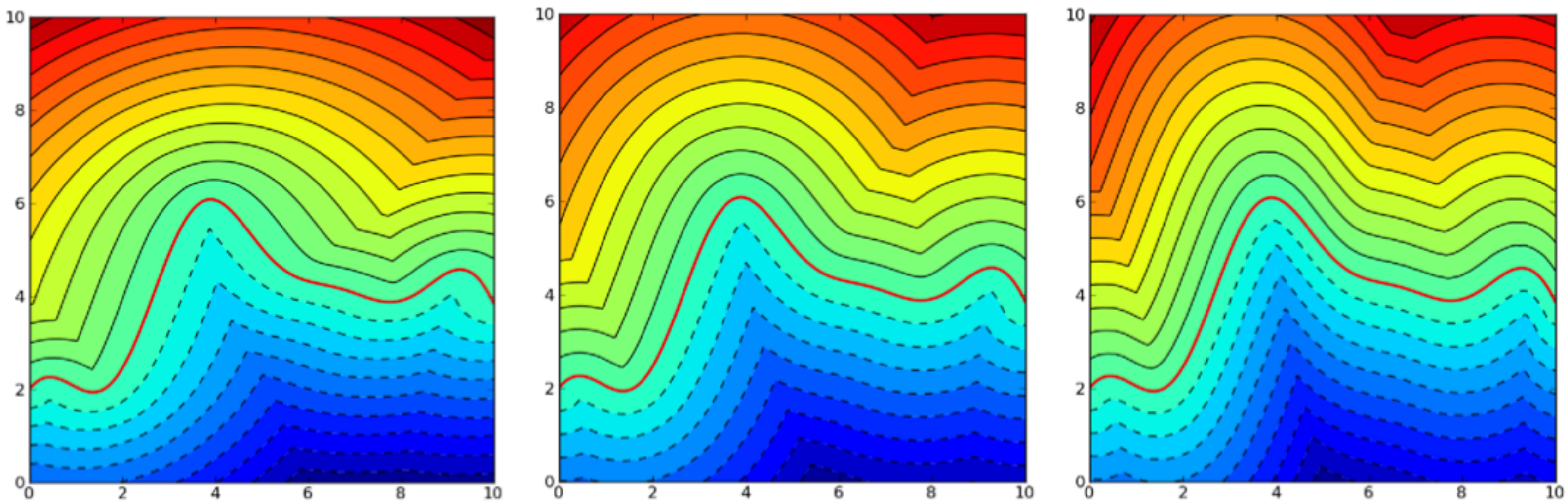
Petersen, S.A. and Hjelle Ø. [2008] Earth Recursion, an Important Component in Shared Earth Model Builders, 70th EAGE Annual Meeting, Rome, 2008.

Restoration along generalized distances, or simulated folds

Layers/Distances are the position of a propagating front at different times ***T(x)***



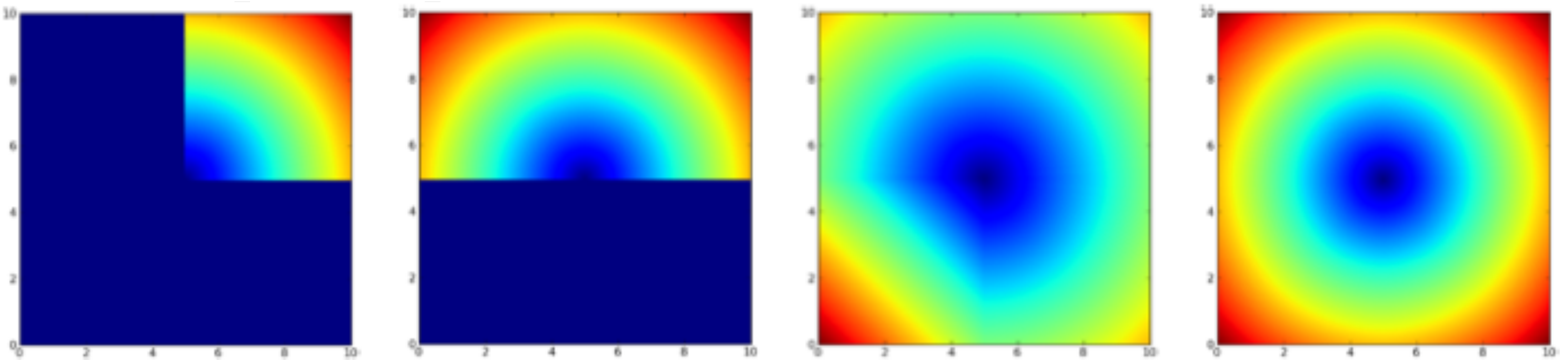
$$F(x)\|\nabla T\| + \Phi(x)\,(\mathbf{a}\cdot\nabla T) = 1, \;\; (anisotropic)$$



(a) Fold of class 1A (b) Fold of class 1B (parallel) (c) Fold of class 1C

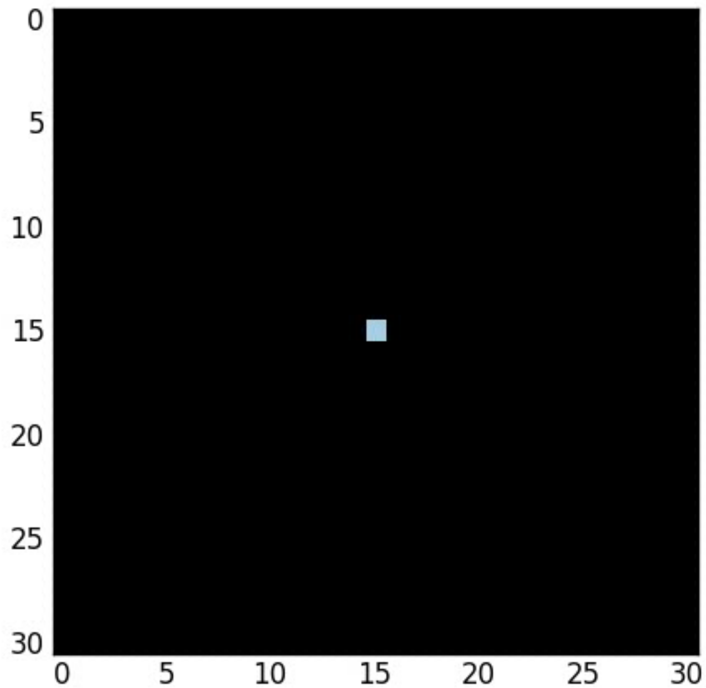
Ø. Hjelle, S. A. Petersen, A Hamilton-Jacobi framework for modeling folds in structural geology, Mathematical Geosciences, 2011

Solution method must be fast for the application to be interactive



- **Sweeping methods**: Compute distances in different directions (iterative)

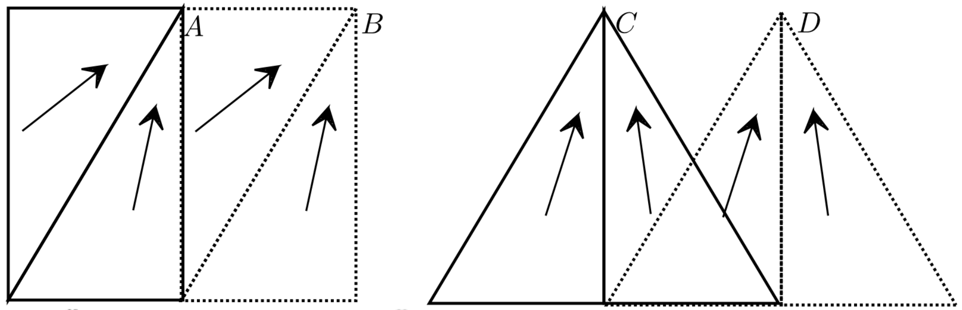
- **Front tracking methods;** Wave simulation. Smart, but sequential

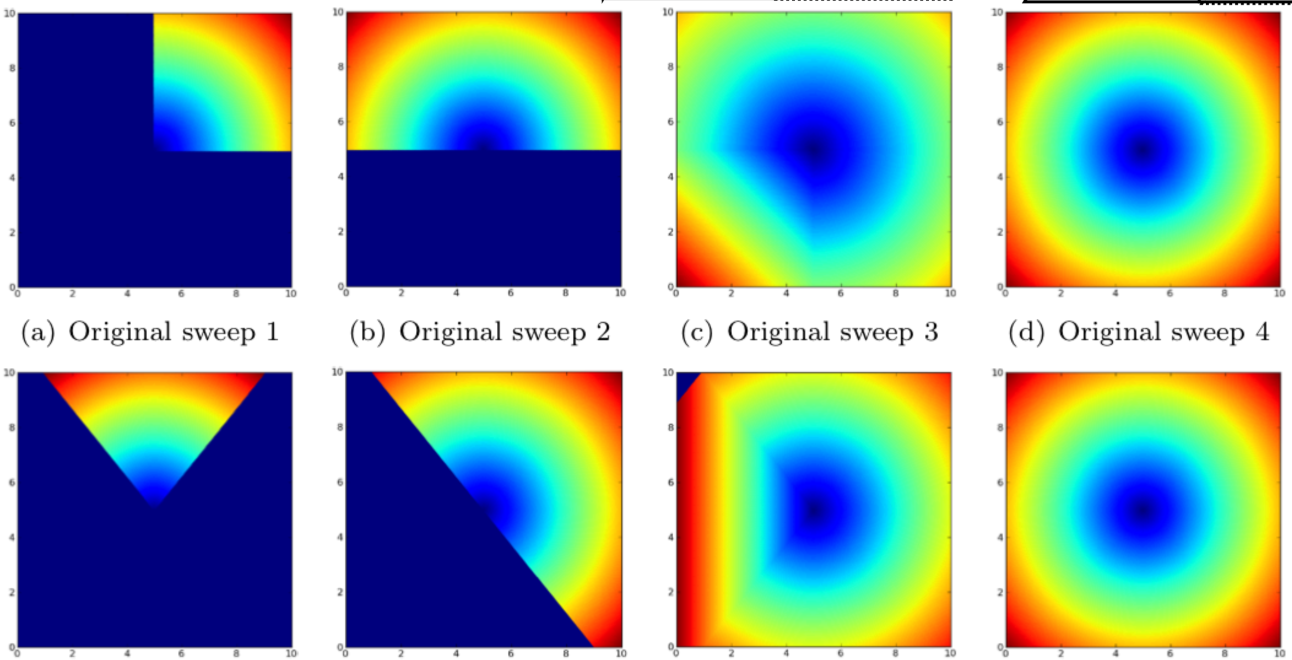


Smart methods are slow on large 3D grids

Sweeping methods for parallel implementations

Modifications of stencil shape, and iteration order give parallel possibilities



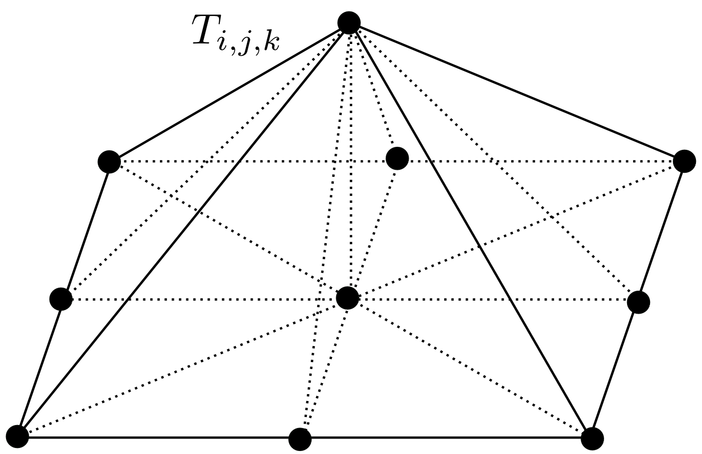


(a) Original sweep 1 (b) Original sweep 2 (c) Original sweep 3 (d) Original sweep 4

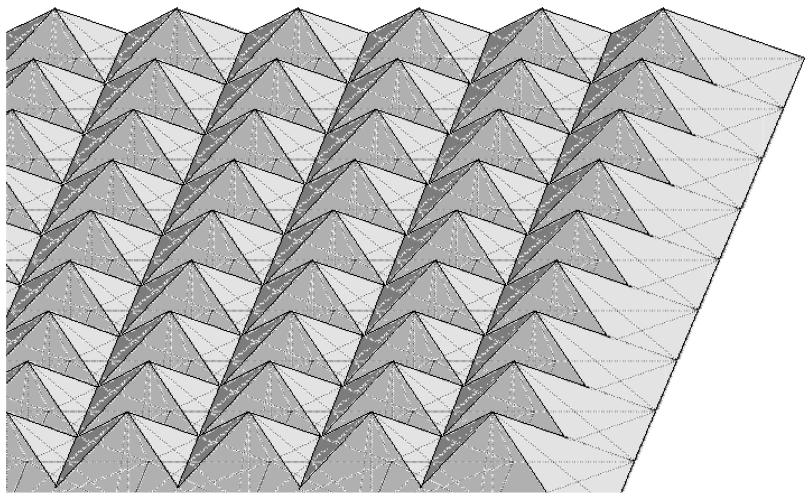
(e) New sweep 1 (f) New sweep 2 (g) New sweep 3 (h) New sweep 4

O.Weber, et al. Parallel algorithms for approximation of distance maps on parametric surfaces, ACM Transactions on Graphics, 2008

A new algorithm;
The 3D Parallel Marching Method



A surface can be updated in parallel

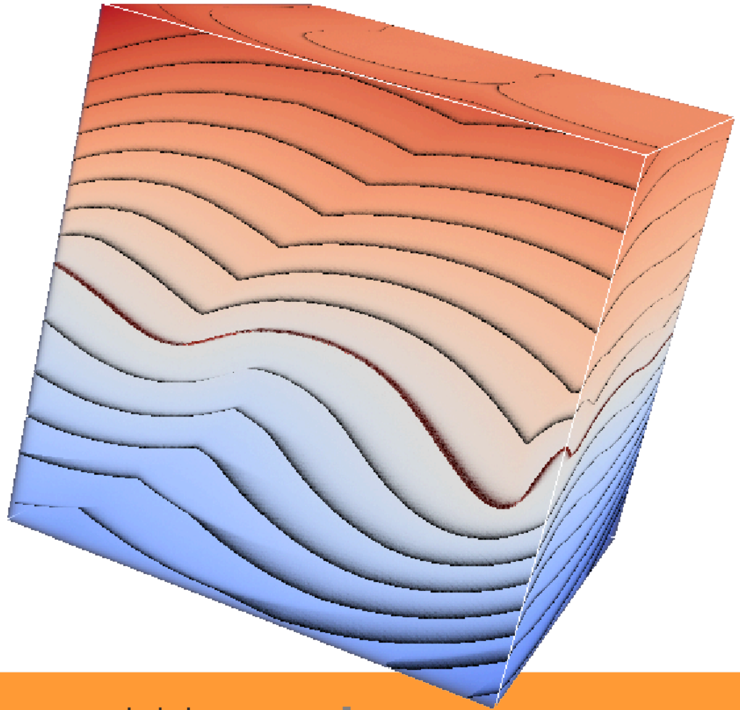


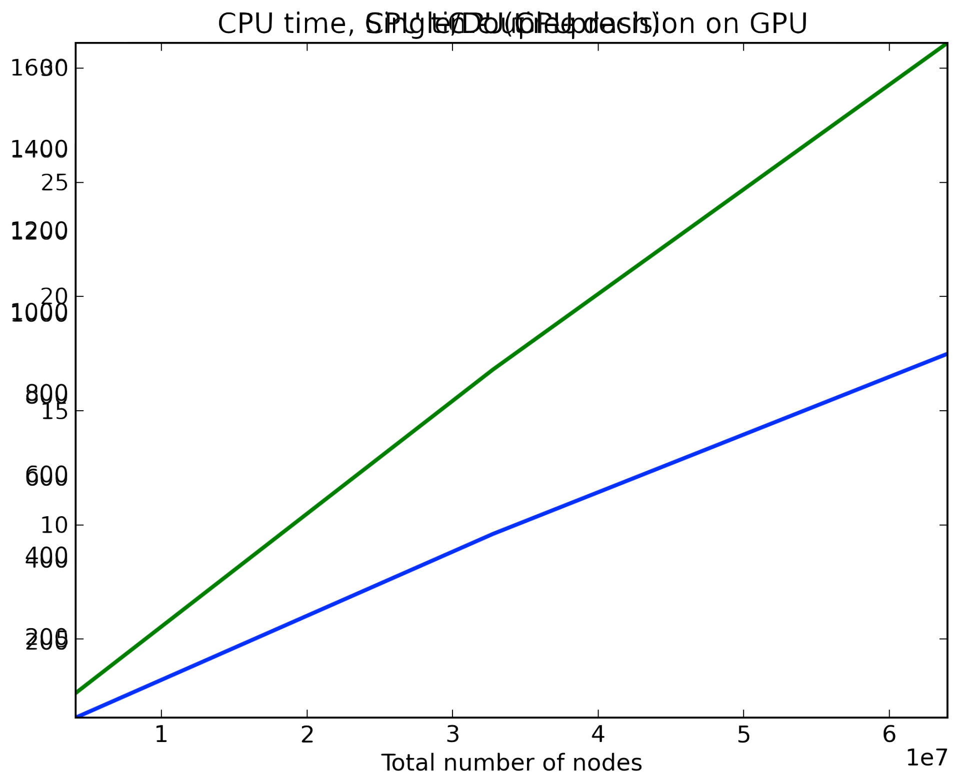
for $i = 2 \rightarrow n_x$ **do**
  for all $j \in [1, n_y]$ and $k \in [1, n_z]$ **do**
    Update $T_{i,j,k}$ using values $T_{i-1,j\pm a,k\pm b}, a \in \{0,1\}, b \in \{0,1\}$
  end for
end for

Parallelizing sequential C code

#pragma mint copy(T,toDevice)
#pragma mint parallel
for $i = 2 \rightarrow n_x$ **do**
  for all $j \in [1, n_y]$ and $k \in [1, n_z]$ **do**
    Update $T_{i,j,k}$ using values $T_{i-1,j\pm a,k\pm b}, a \in \{0,1\}, b \in \{0,1\}$
  end for
end for
end for
#pragma mint copy(T,fromDevice)

1. Using OpenMP
2. Using **Mint** to generate Cuda code for GPUs





160^3 **320^3** **400^3**

Some conclusions and remarks

Although not a 'smart' algorithm, the 3D Parallel Marching Method is simple to parallelize and implement

Thank you!

Mint is a powerful and free tool for parallelization

https://sites.google.com/site/mintmodel/

The code is still in the development stage, and there are many optimizations to explore

N	t_1	t_2	s_2	t_4	s_4	t_6	s_6	t_8	s_8	$t_{S\,PGPU}$	$s_{S\,PGPU}$	t_{DPGPU}	s_{DPGPU}
160^3	104.4	54.1	1.9	28.3	3.7	19.3	5.4	14.7	7.1	1.6	67.1	2.6	39.7
320^3	864.6	449.9	1.9	234.6	3.7	158.9	5.4	121.6	7.1	9.6	89.9	16.8	51.4
400^3	1661.0	866.5	1.9	416.0	3.7	305.9	5.4	233.8	7.1	17.5	95.0	31.1	53.4

D. Unat, X. Cai, S. Baden, Mint: Realizing CUDA performance in 3D stencil methods with annotated C, in: Proceedings of the 25th International Conference on Supercomputing (ICS'11), ACM Press, 2011, pp. 214–224