Cache–aware Multigrid for 2– and 3–dimensional Elliptic Equations

Markus Kowarschik

Ulrich Rüde

Christian Weiß

recently:
Institut für Mathematik
Universität Augsburg

from now on:
Institut für Informatik
Universität Erlangen

http://wwwbode.in.tum.de/Par/arch/cache

GAMM Workshop on Multigrid Methods
Bonn, October 1998

Outline

1. Why do we need cache–aware programming?
2. How to manage the 2D case
   (including a closer look at cache memories)
3. Problems in 3D
4. Conclusion and future work

What is the problem?

$\boxed{\text{Theoretically ...}}$

modern workstations based on superscalar RISC processors can do $>$ 1000 MFlops/sec!

$\boxed{\text{In practice ...}}$

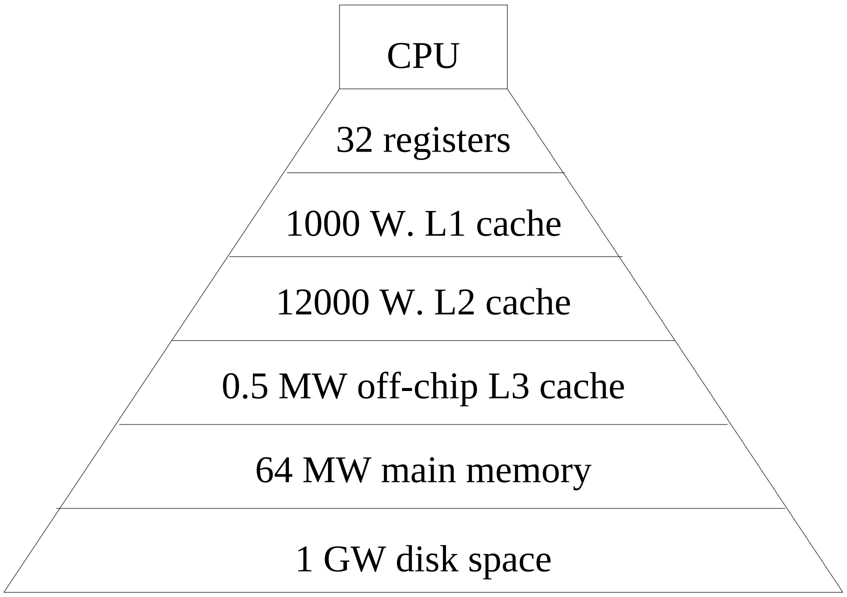
we often obtain disappointing results.

Example: Performance of 3D red/black Gauß–Seidel smoother on a Digital Alpha PWS 500au:

grid size	# unknowns	MFlops/sec
16	4096	415
32	32768	194
64	262144	76
128	$\approx 2.1 \cdot 10^6$	73

Remedy: data locality optimizations via cache–aware programming

Memory hierarchy of the DEC Alpha PWS 500au:



Level	Capacity	Throughput	Latency
Register	512 Byte	34 GByte/s	2 ns
L1 Cache	8 KByte	9.6 GByte/s	2 ns
L2 Cache	96 KByte	4.8 GByte/s	6 ns
L3 Cache	4 MByte	1 GByte/s	25 ns
MM	14 GByte	300 MByte/s	105 ns

We need algorithms that take care of the memory system of the underlying architecture.

The 2D model problem

$$\Delta u = f \text{ in } \Omega = (0,1)^2,$$

$$u = g \text{ on } \partial\Omega$$

Analysis of a MG V–cycle implementation:

- uniform grids, 5–point stencil
- size of the finest grid: 1025^2 points
- restriction: half injection
- linear interpolation
- smoothing: 4 iterations of r/b GS

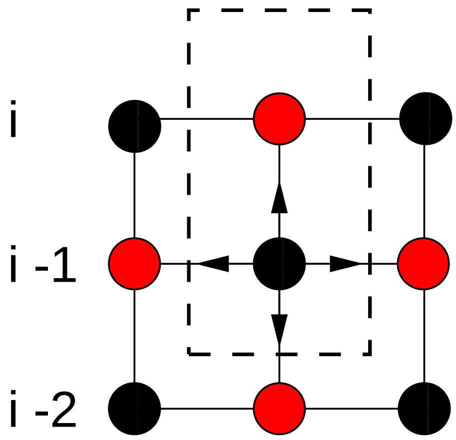
procedure	time consumption
relaxation	84.0%
restriction	9.4%
interpolation	6.3%

$\Longrightarrow$ Tune the smoothing algorithm first!

Basic optimization techniques

Intention:
optimization of $\boxed{\text{temporal and spatial locality}}$

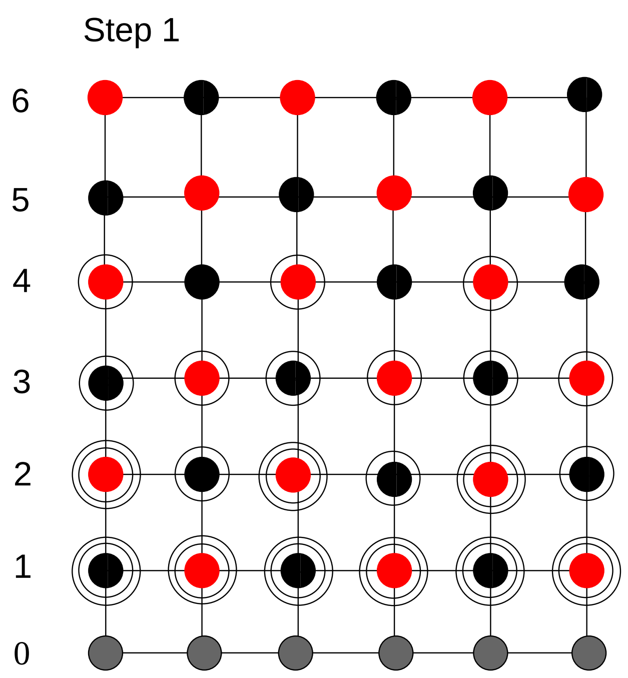
$\boxed{\text{I. Fusing red and black point relaxation}}$



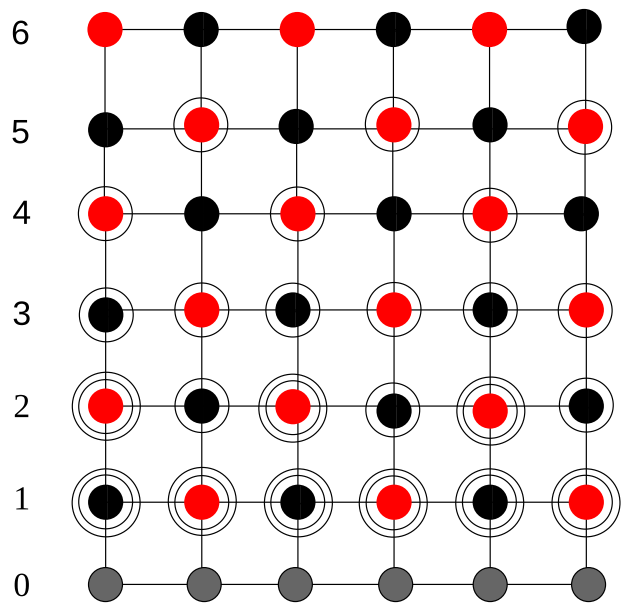
1. relax red points in line i
2. relax black points in line i-1
3. i $\longleftarrow$ i+1
4. goto 1

Basic optimization techniques

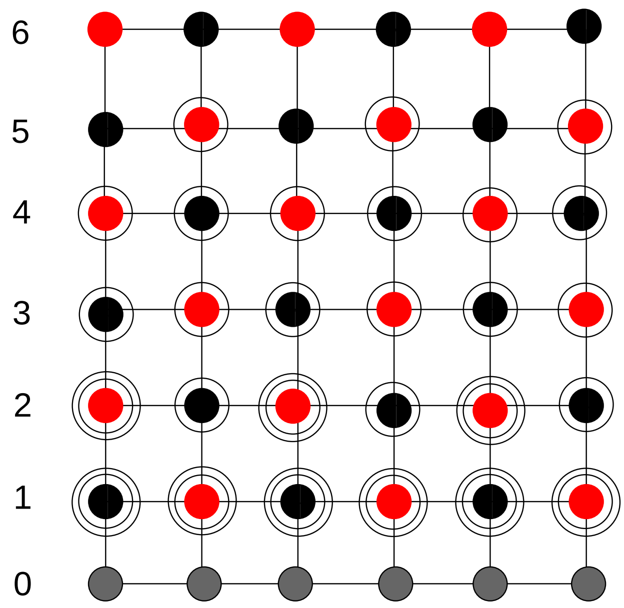
II. Blocking of several r/b GS sweeps



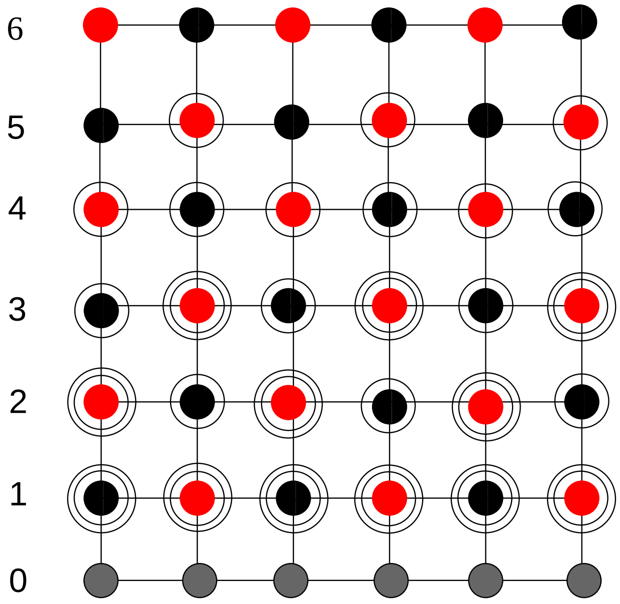
Step 2



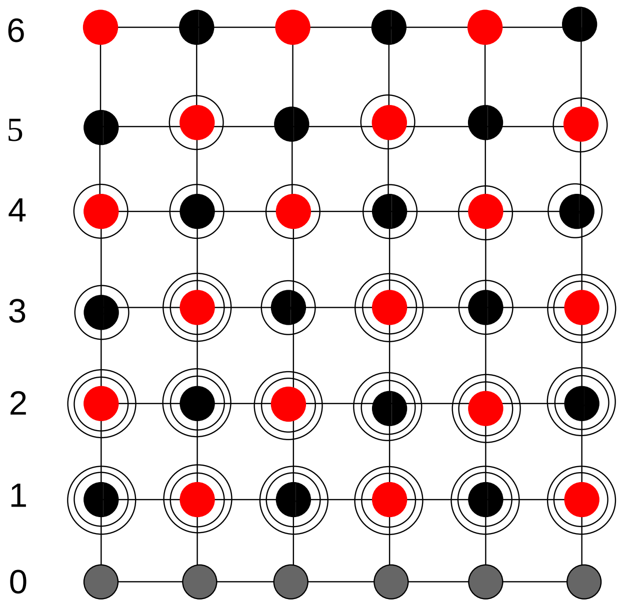
Step 3



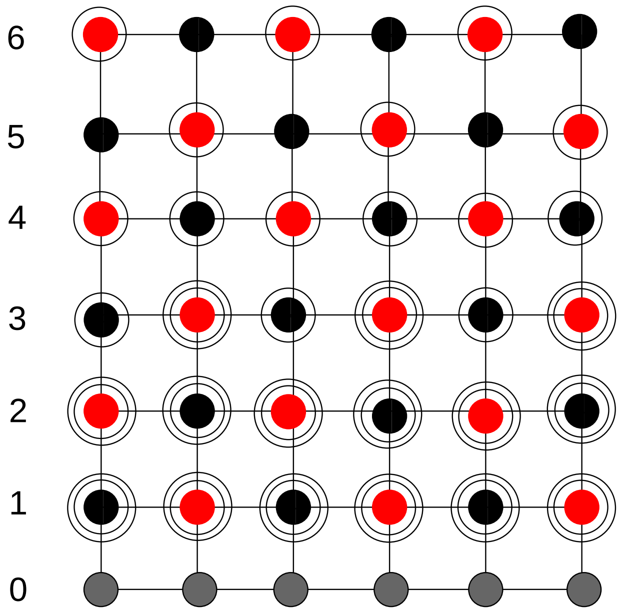
Step 4



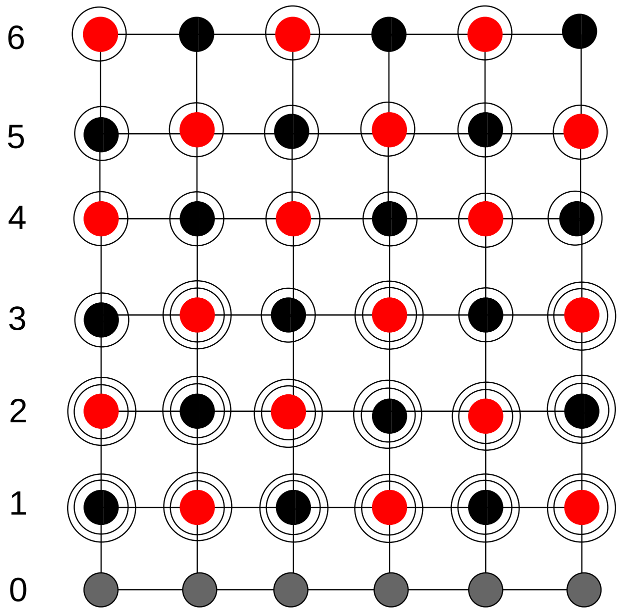
Step 5



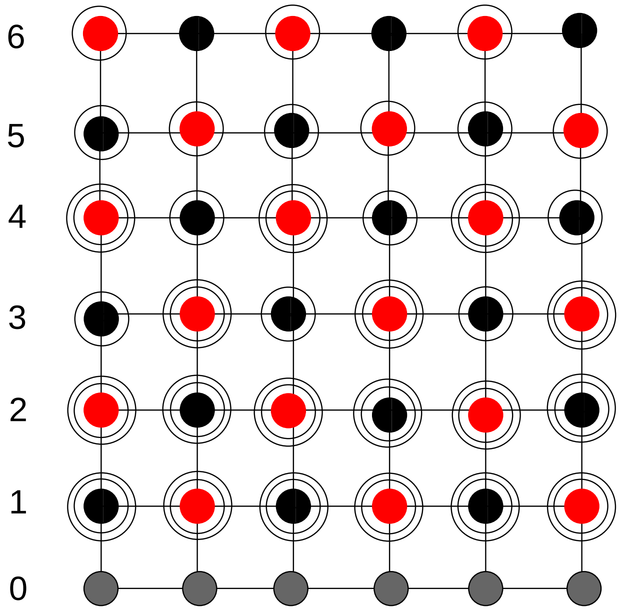
Step 6



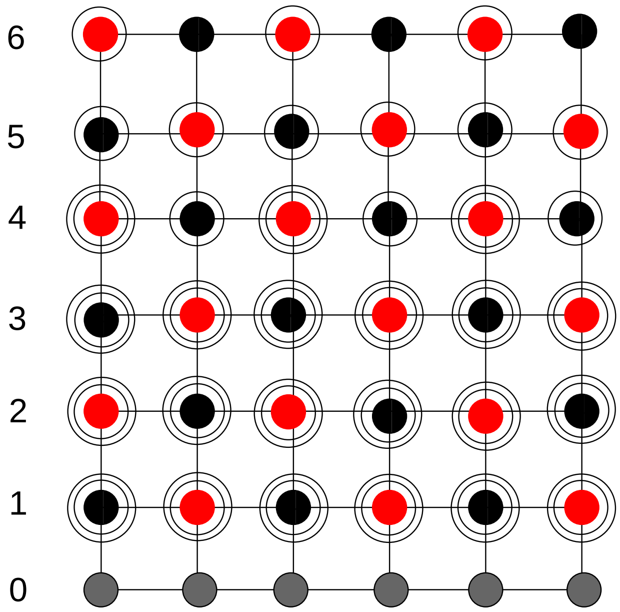
Step 7



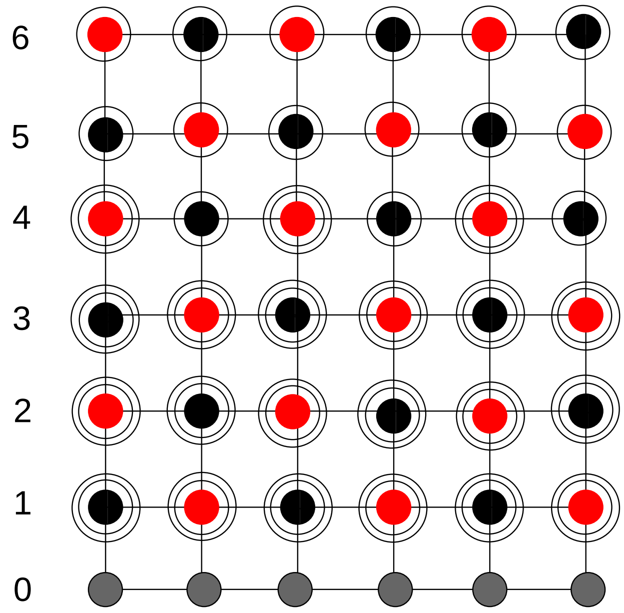
Step 8



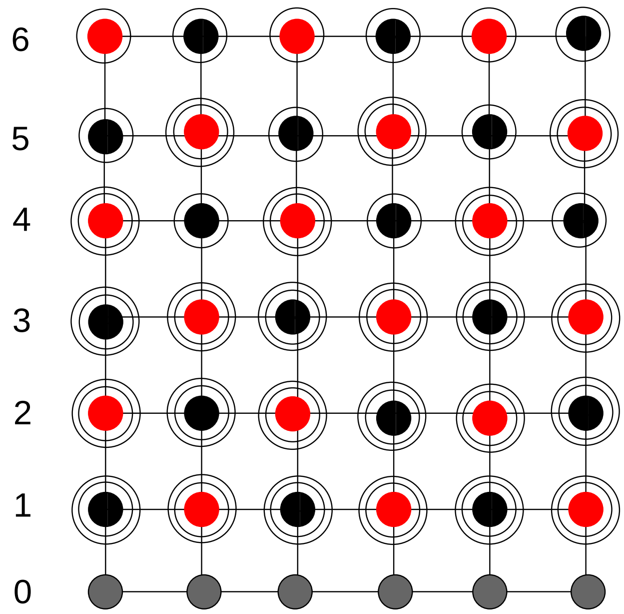
Step 9



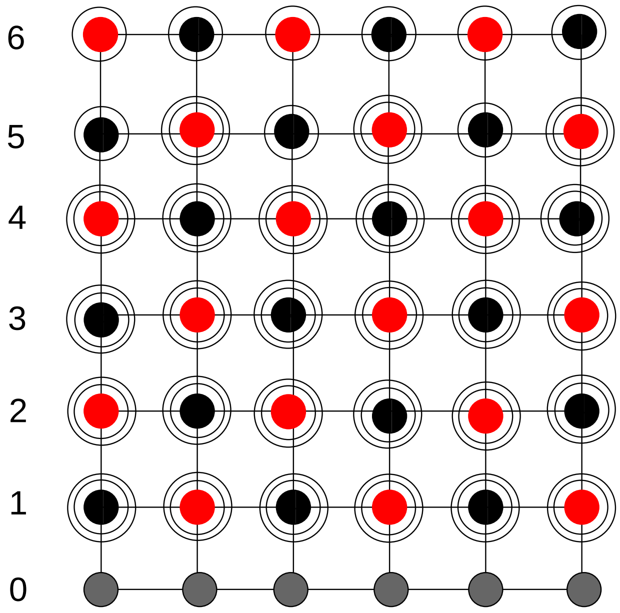
Step 10



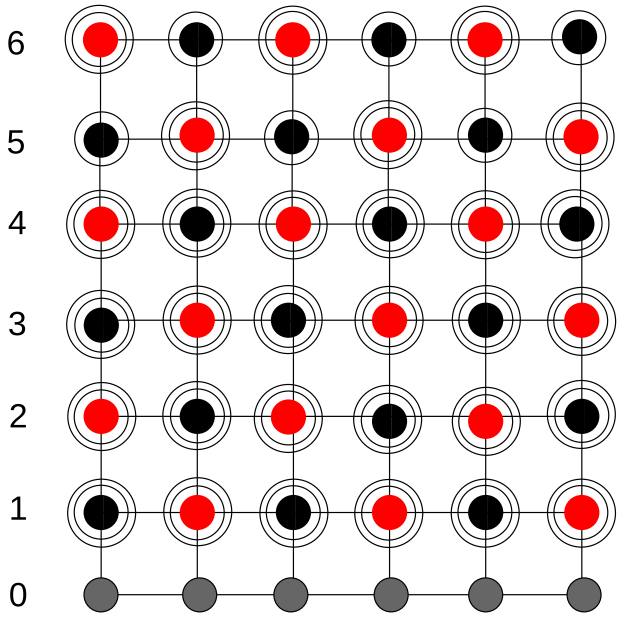
Step 11



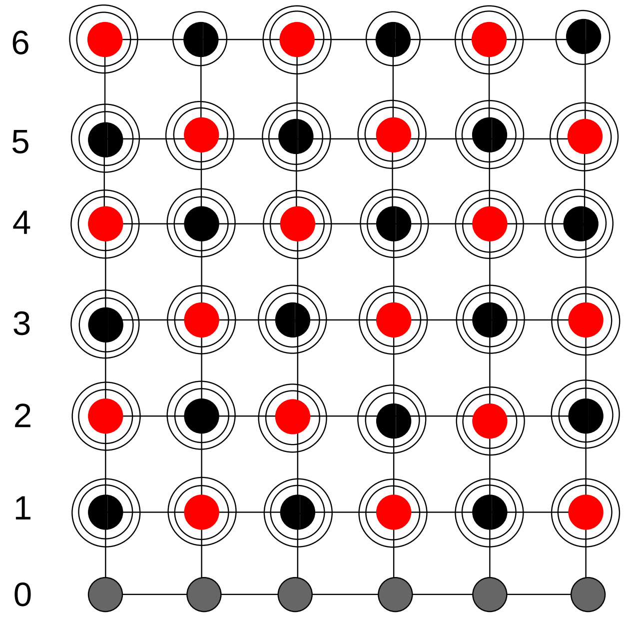
Step 12



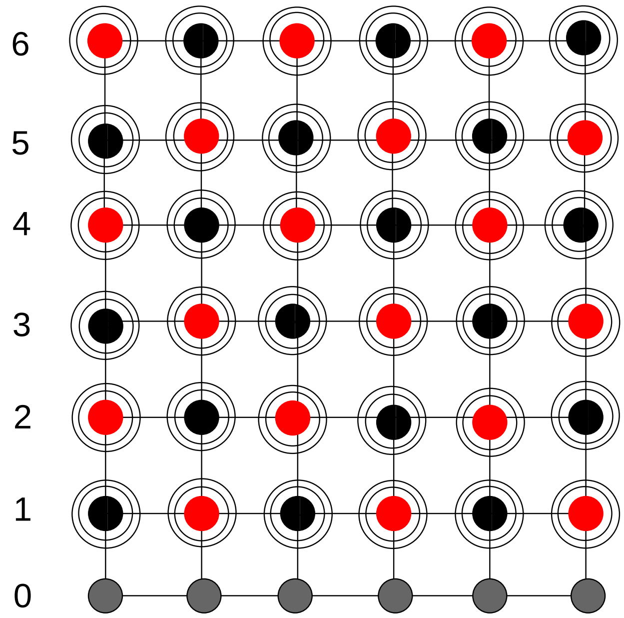
Step 13



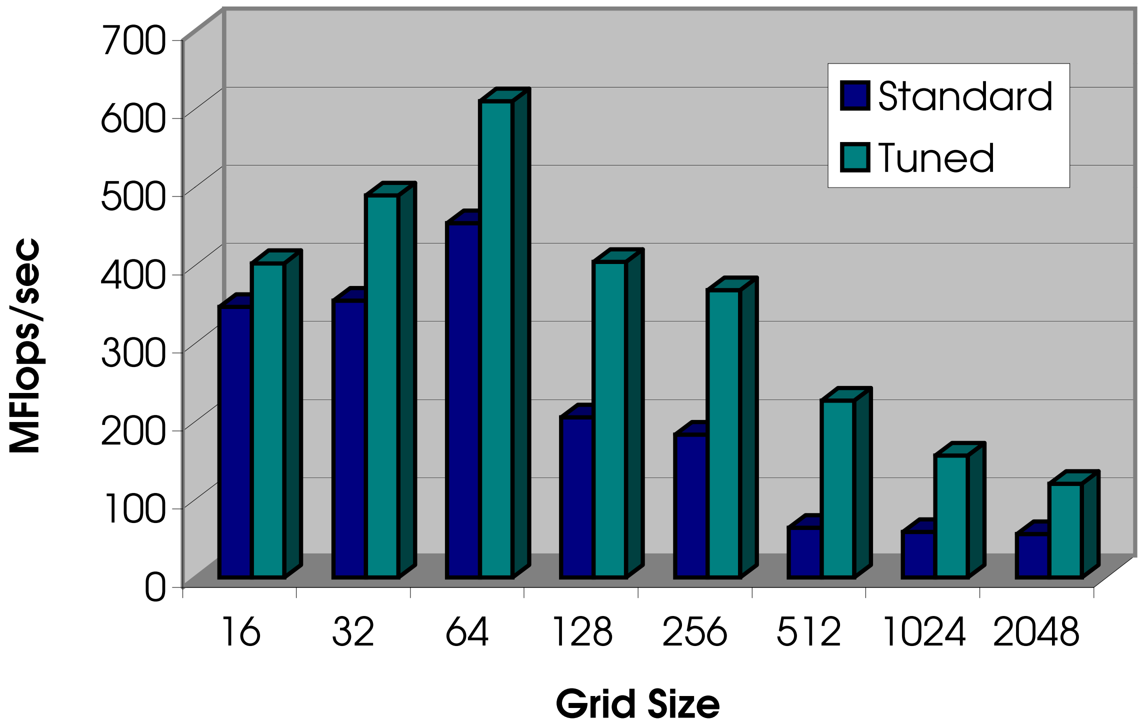
Step 14



Step 15



What can be achieved by now?

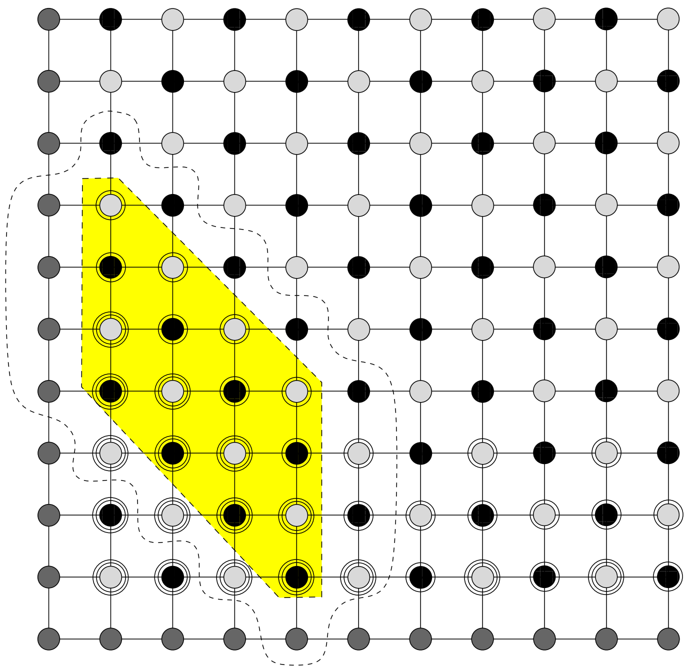


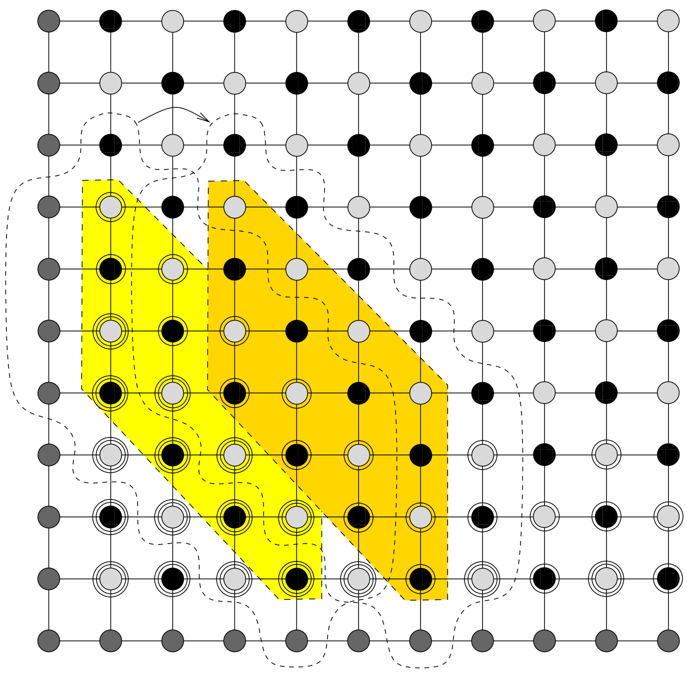
For each grid size the best result is shown.

How can we further improve performance
— especially on larger grids?

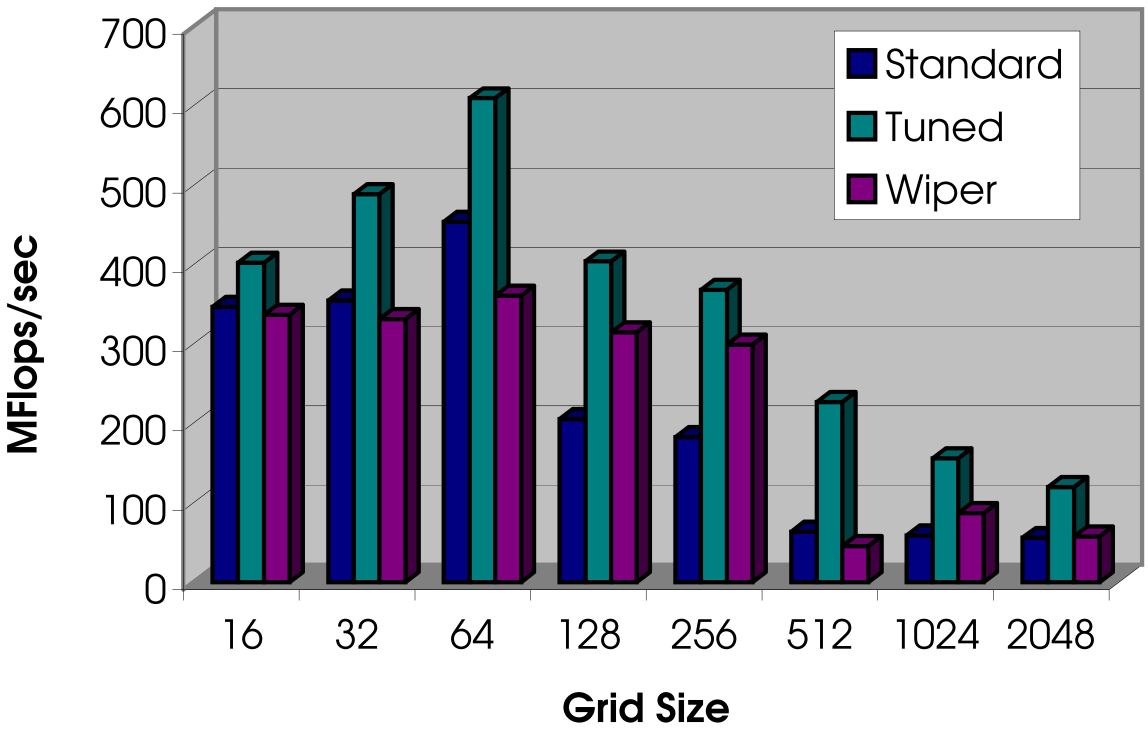
The windshield wiper technique

Idea: Blocking in both dimensions
(still respecting all data dependencies)





Poor performance results



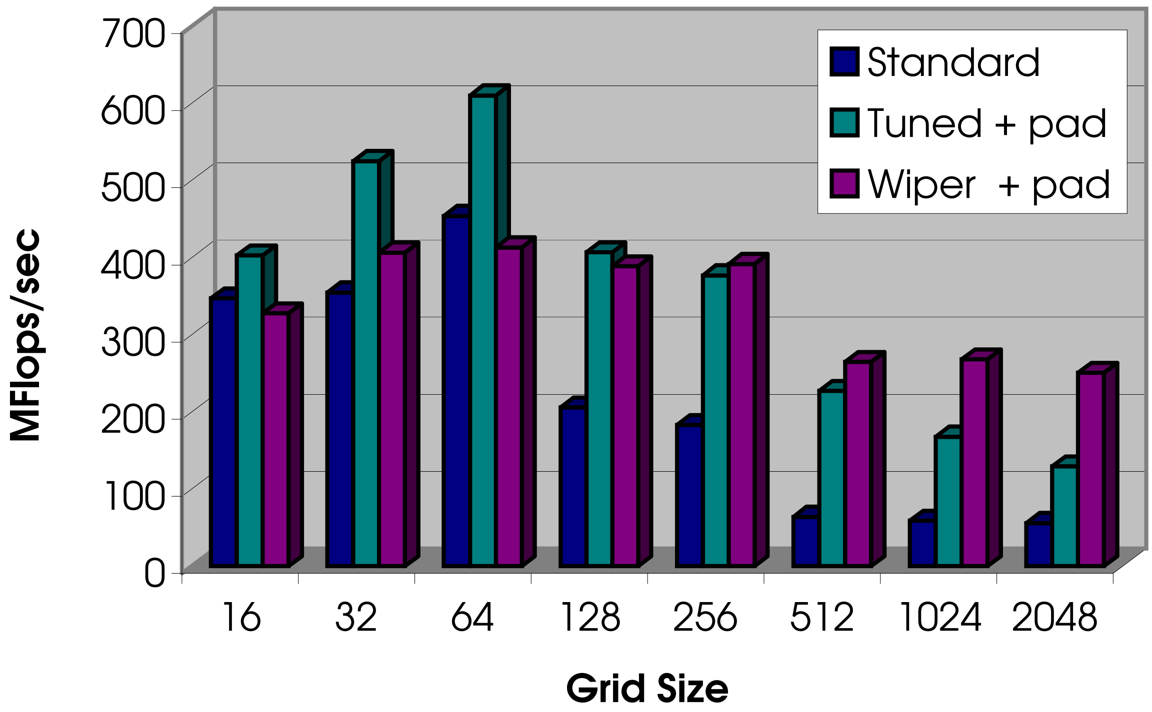
"Magic" trick: array padding

double precision u(0:N+pad_1,0:N)

double precision f(0:N+pad_2,0:N)

What does this lead to?

Significant speedups



Padding is not that useful for basic optimization techniques.

However, there are significant speedups for our windshield wiper technique!

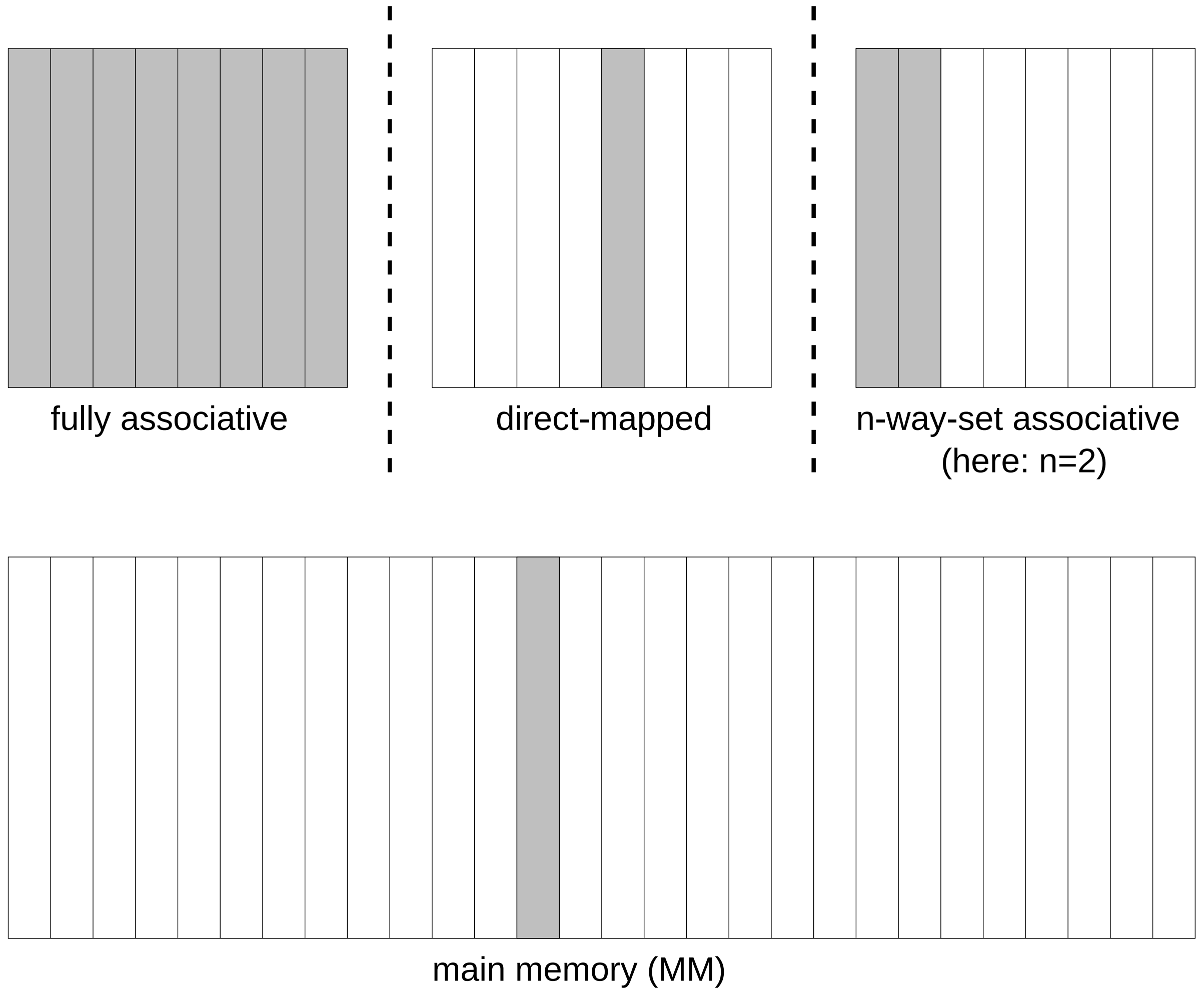
Why?

A closer look at cache memories

Where are MM blocks placed in the cache?

Classification of cache types by their

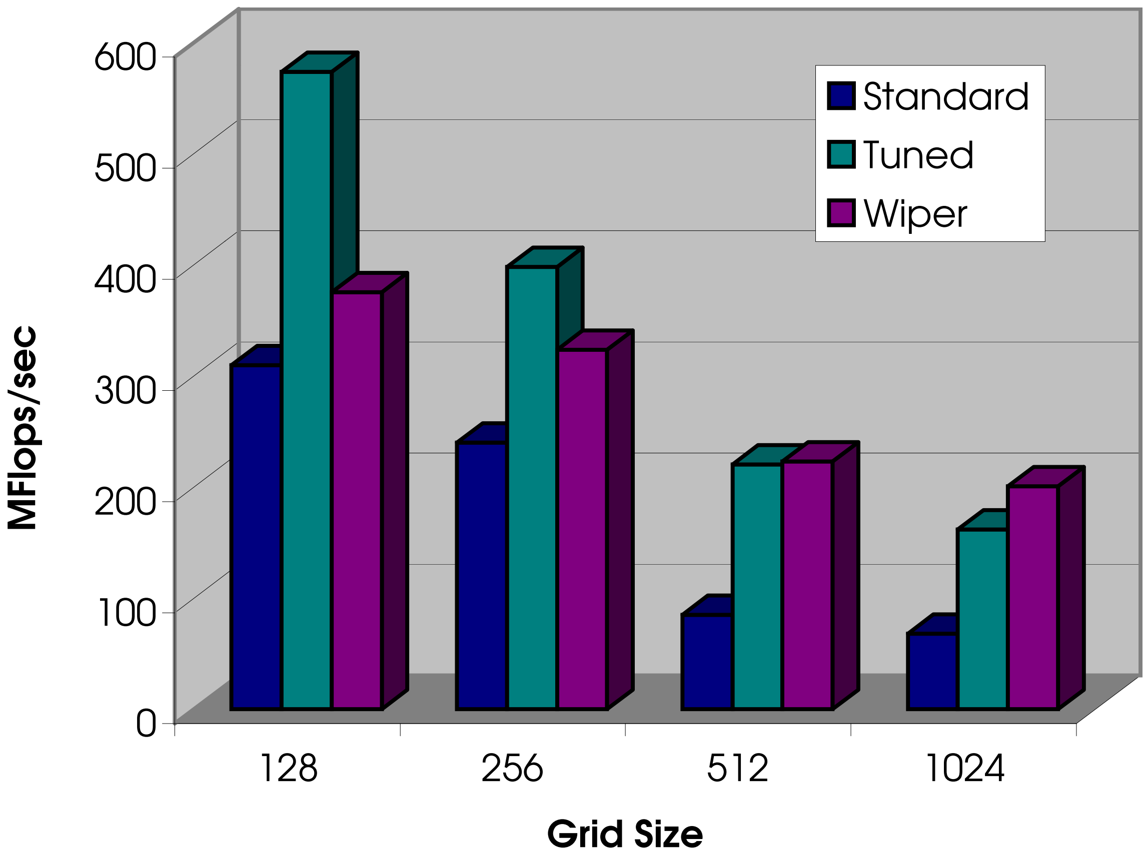
$\boxed{\text{degree of associativity}}$



Classification of cache misses

1. cold start misses (= compulsory misses)
   - occur when data items are accessed for the first time
   - inevitable
2. capacity misses
   - occur when cache is not large enough to hold all the blocks needed during program execution
   - caused by data items being discarded and later on retrieved
3. conflict misses
   - occur when too many cache lines map to the same set
   - depend on the cache associativity
   - may be reduced by array padding

Performance of the 2D MG V–cycle



- 4 × r/b GS per grid level, 5–point stencil
- half injection restriction operator
- linear interpolation operator
- to do: merging of these components

http://wwwbode.in.tum.de/Par/arch/cache

In 3D things get worse!

$\boxed{\text{First idea}}$

Adapt the 2D basic optimization techniques to 3D by replacing “grid line” by “grid plane”.

$\boxed{\text{Problem}}$

Caches in general are not big enough to hold four adjacent grid planes, which is needed to run efficiently.

Example:

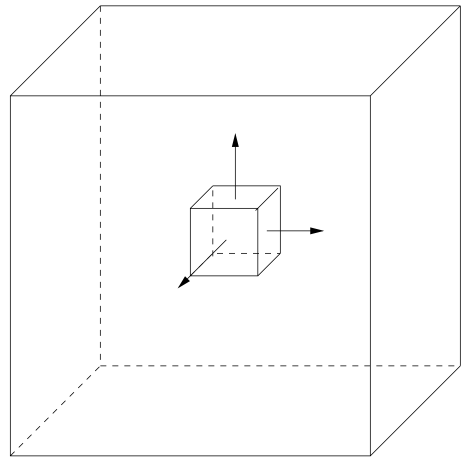
A single plane in a cubic grid with 64^3 double precision values contains 64^2 points and therefore occupies 32 KByte of memory!

What can we do?

A 3D blocking technique

Remedy (?)

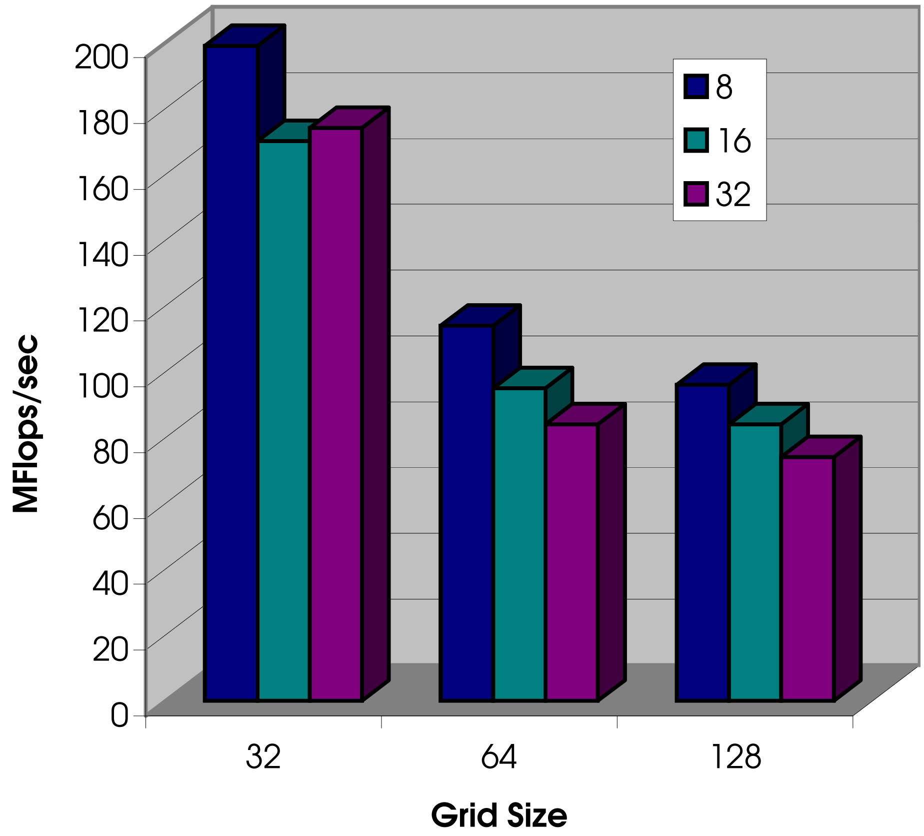
Use a blocking technique based on a subcube moving through the grid, respecting the data dependencies.



- various implementations are conceivable
- our algorithm:

```
for each subcube position within the grid
        update red points in subcube
        update black points in subcube
```

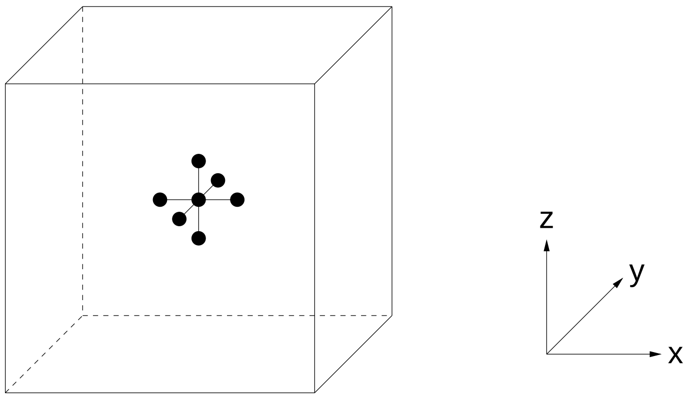
Poor performance results



Why does this technique perform so badly?

Reason: conflict misses

Assume the following situation:



- 3D grid (array) with 64^3 points (double precision FP values)
- 8 KByte direct–mapped L1 cache (e.g. Alpha 21164)
- x is leading, z is trailing array dimension

Consequences:

- x-adjacent grid points are adjacent in MM
- y-adjacent grid points are 512 Byte away from each other in MM
- z-adjacent grid points are $4 \cdot$ 8 KByte away from each other in MM and therefore map to the same cache location: conflict misses

What can we do to reduce conflict misses?

- $\boxed{\text{(standard) array padding is useless}}$
  in most of the cases padding doesn't help, but just changes the subsets of grid points that cause conflict misses

- $\boxed{\text{sophisticated padding techniques}}$
  place the lines of the grid individually in memory, so that conflict misses within the subcube are avoided

- $\boxed{\text{wrapped data types}}$
  usage of data types that are structured differently, e.g.

  ```
  double precision u(0:d-1,0:d-1,0:d-1,0:N/d,0:N/d,0:N/d)
  ```

  Tests with synthetic kernels show speedups from 90 to 360 MFlops/sec!

Conclusion and future work

- 2D:
  - sufficiently well understood
  - remarkable performance results and speedups
  - further improvements by using 9–point stencils
  - further speedups by merging the MG components
- 3D:
  - harder than initially expected
  - future work is necessary to understand the basic problems and to achieve significant performance speedups
- application to "real scientific applications":
  - variable coefficients
  - unstructured grids
  - etc.