10th Copper Mountain Conference on Multigrid Methods

Enhancing the Cache Performance of Multigrid Codes on Structured Grids

Markus Kowarschik

Universität Erlangen–Nürnberg

Our project is presently funded by the *Deutsche Forschungsgemeinschaft* (DFG)

Outline

1. Introduction
2. Data locality optimizations for stencil–based operations
   (a) Data layout optimizations + performance results
   (b) Data access optimizations + performance results
3. Conclusions and future work

Introduction

Performance results for a DEC PWS 500au (Alpha 21164 CPU), theoretically available peak performance: 1000 MFLOPS

We consider a standard Gauss–Seidel C code:

- Structured grid, 5–point stencil, $\approx$ 64.000 unknowns, $\approx$ 3.5 MB ($<$ size of L3 cache $=$ 4 MB): 45 MFLOPS
- Unstructured grid, compressed row storage scheme, $\approx$ 64.000 unknowns: 16 MFLOPS

Profiling analysis of the structured grid code:
$>$ 50% of the execution time are spent for moving data!

$\Rightarrow$ Need for cache–aware codes

$\Rightarrow$ Our project: *DiME* — **D**ata–local **i**terative **ME**thods

Profiling multigrid codes

5–point stencil, MG V–cycles, CGC, finest grid: 1025$\times$1025 nodes, full–weighting, linear interpolation, red–black GS smoother:

% of CPU time for V(4,4) cycles	% of CPU time for V(2,2) cycles	function name
82.63	70.12	smooth
7.83	13.36	restriction
7.32	12.73	prolong
2.23	3.80	init_grid

$\Rightarrow$ Optimize the smoother first!

Data locality optimizations:

Data layout optimizations + Data access optimizations

Data layout optimizations: array padding

Cache architecture:

- Cache memories are split into *cache lines*
- *Associativity*:
  - n–way set–associative, special cases:
  - $n = 1$: direct mapped
  - $n =$ max: fully associative

Cache thrashing:

- Repeatedly used data are mapped to the same cache lines and mutually throw each other out of the cache
- Very likely and dramatic for stencil–based codes, esp. for grid dimensions close to powers of 2 (cache sizes!)

Array padding:

Allocate arrays larger than necessary — i.e. change the relative distances in memory — in order to avoid/mitigate cache thrashing

Cache–aware data storage schemes

Many possibilities, example: scalar equation, 5–point stencil:

Equation–oriented:

u(0)	ce(0)	we(0)	ea(0)	no(0)	so(0)	f(0)	...	u(n)	ce(n)	we(n)	ea(n)	no(n)	so(n)	f(n)

Band–wise:

u(0)	u(1)	...	u(n)	f(0)	f(1)	...	f(n)	ce(0)	ce(1)	...	ce(n)	we(0)	we(1)	...	we(n)	...

...	ea(0)	ea(1)	...	ea(n)	no(0)	no(1)	...	no(n)	so(0)	so(1)	...	so(n)

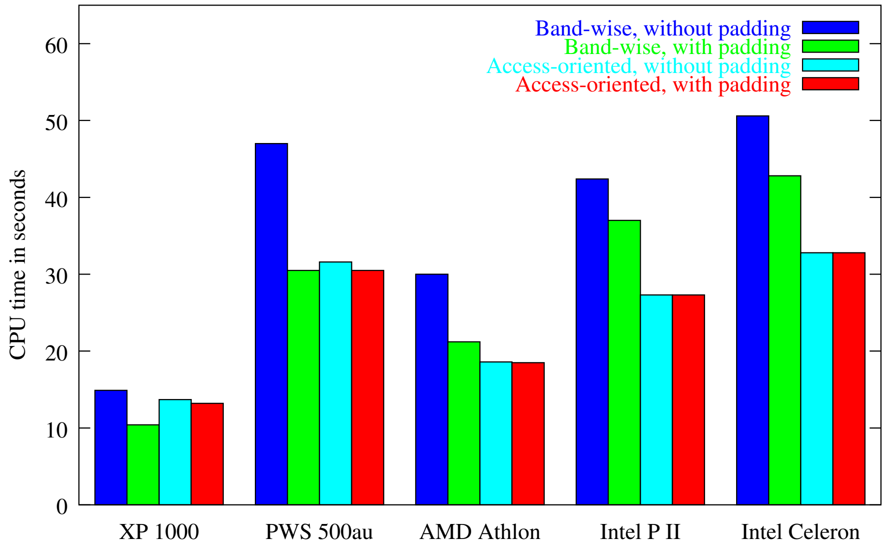
Access–oriented: cache–aware!

u(0)	u(1)	...	u(n)	f(0)	ce(0)	we(0)	ea(0)	no(0)	so(0)	...	f(n)	ce(n)	we(n)	ea(n)	no(n)	so(n)

Execution times for multigrid codes

- Equation–oriented data storage scheme yields poor performance (much currently unnecessary data is fetched into cache)
- CPU times for the multigrid codes w/wo suitable array paddings (finest grid: 1025×1025 nodes, 10 V(2,2)–cycles)

ca. 25 MFLOPS



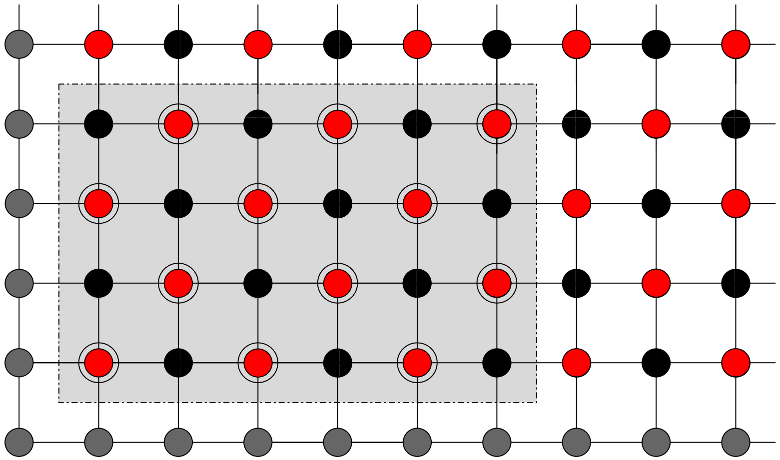
$\Rightarrow$ The band–wise storage scheme is very sensitive, the access–oriented storage scheme is rather insensitive to array padding

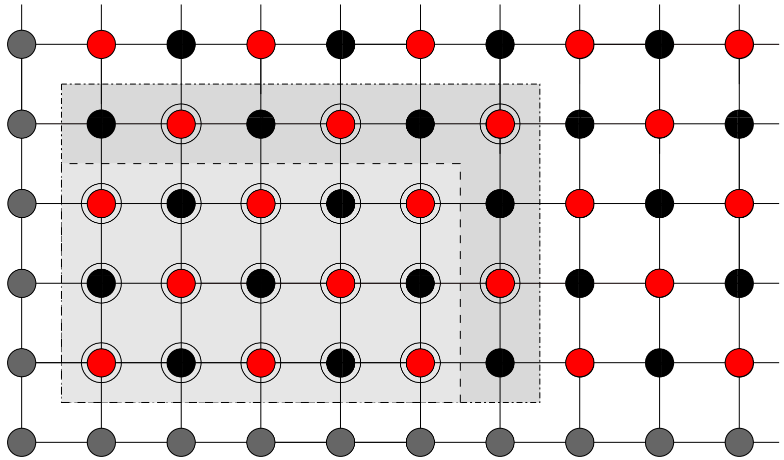
Survey: data access optimizations

Our access optimization techniques have been presented previously, e.g. *ACM/IEEE Supercomputing Conference 1999*

Loop transformations: minimize #passes through the data set

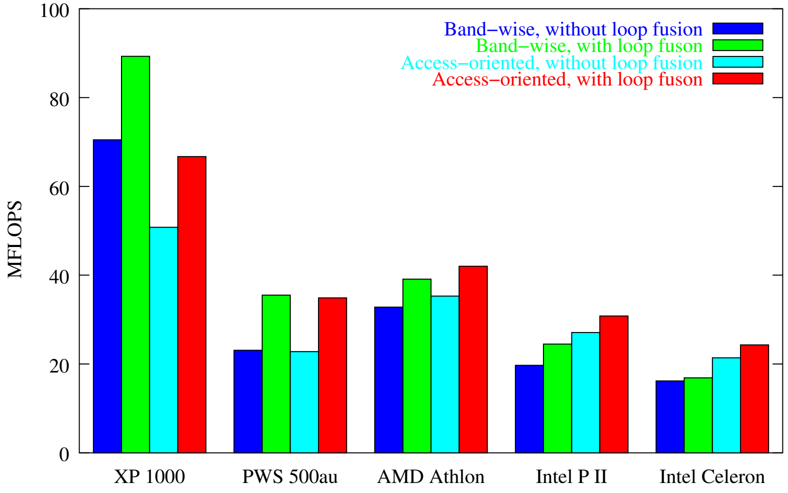
- Loop fusion: one sweep per smoothing iteration
- 1D loop blocking (tiling): several iterations per sweep
- 2D loop blocking, e.g., square blocking:

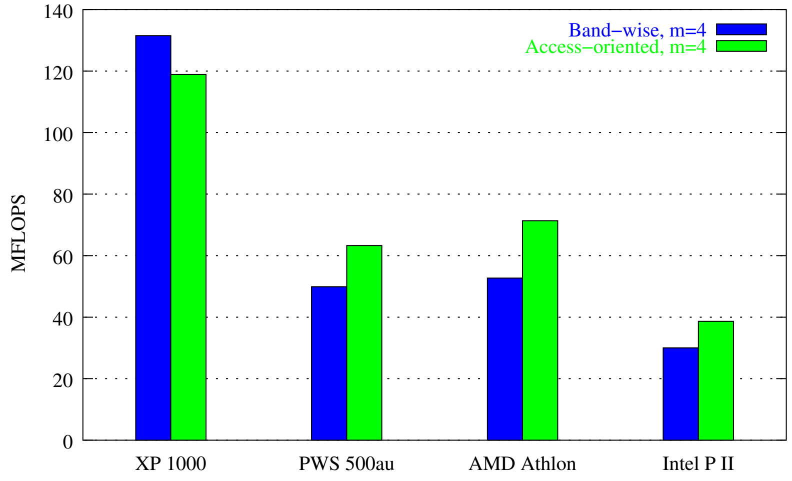




Performance results

- MFLOPS for r/b GS (w padding), grid size: 1025×1025 ($\approx$56 MB), left: loop fusion, right: 2D blocking





- Cache misses + CPU times, 40 GS steps, A21164 machine:

	Standard	Fusion (wp)	1D bl. (wp)	2D bl. (wp)
L1 misses	$3.7 \cdot 10^8$	76%	73%	78%
L2 misses	$1.5 \cdot 10^8$	87%	93%	41%
L3 misses	$7.5 \cdot 10^7$	52%	19%	16%
CPU time	20.0s	13.3s	10.0s	8.0s

The DiMEPACK library

DiMEPACK: a cache–aware multigrid library

- Correction scheme: V–cycles, full multigrid
- Rectangular domains
- So far: constant 5–/9–point stencils
- Dirichlet/Neumann boundary conditions
- C++ interface, fast FORTRAN 77 subroutines
- Direct solution of the problems on the coarsest grid (LAPACK)
- Single precision/double precision floating–point arithmetic
- Cache–optimized intergrid transfer operations
- Various array padding heuristics
- etc.

Email: `cache@in.tum.de`

WWW: `wwwbode.in.tum.de/Par/arch/cache`

Conclusions and future work

- Speedup factors of 2–3 are possible for stencil–based computations on structured grids
- Even higher speedups for constant coefficient problems
- Inherently data–local algorithms and data structures:
  - Patch–based grid structures
  - Multigrid techniques based on adaptive patch processing (cf. Fully adaptive multigrid methods, U. Rüde)
- Practical applications: chip layout optimization (EDA, TUM)
- Unstructured grids: C.C. Douglas, University of Kentucky
- Optimizing preprocessors (ROSE): D. Quinlan, LLNL

Email: `cache@in.tum.de`
WWW: `wwwbode.in.tum.de/Par/arch/cache`