How much can be gained by Cache Aware Programming of Multigrid Methods

Markus Kowarschik, Christian Weiß,
Jonathan Hu, Craig C. Douglas,
Ulrich Rüde, Wolfgang Karl

Universität Erlangen-Nürnberg
Technische Universität München
University of Kentucky

9th Copper Mountain Conference
on Multigrid Methods
April 1999

Outline

1. Why we need cache aware programming

2. Optimization techniques for 2D

3. Optimization techniques for 3D

4. Closing remarks

5. Future work

What the problem is

Theoretically ...

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

In practice ...

we often obtain disappointing results

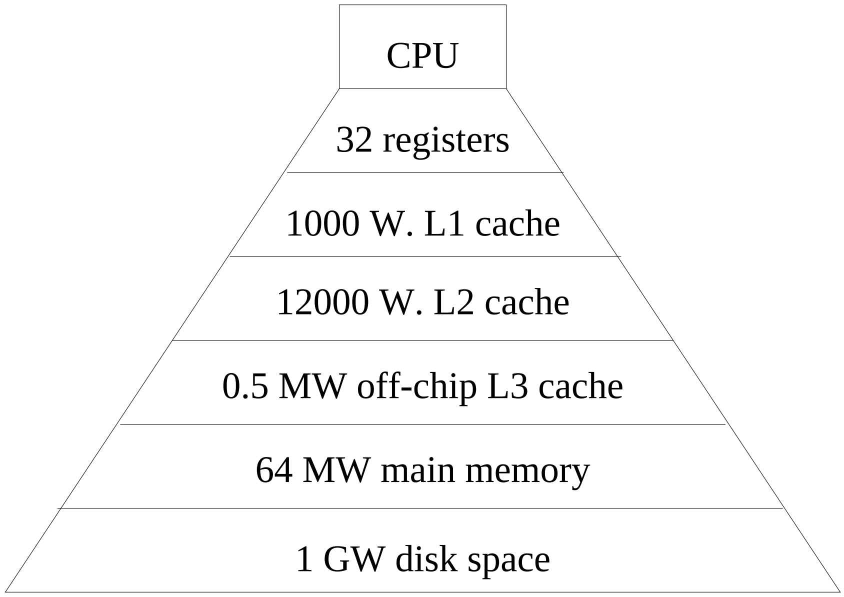
Example: Performance of 3D red/black Gauss–Seidel smoother on a Digital Alpha PWS 500au:

grid size	# unknowns	MFlops/sec
18^3	4096	382
34^3	32768	208
66^3	262144	138
130^3	$\approx 2.1 \cdot 10^6$	74

Why we need cache aware algorithms (1)

Example:

Memory hierarchy of the DEC Alpha PWS 500au:



Level	Capacity	Throughput	Latency
Register	512 Byte	24 GByte/s	2 ns
L1 Cache	8 KByte	16 GByte/s	2 ns
L2 Cache	96 KByte	8 GByte/s	6 ns
L3 Cache	4 MByte	888 MByte/s	24 ns
Main Mem	512 MByte	1 GByte/s	112 ns

Why we need cache aware algorithms (2)

$\boxed{\text{Fact:}}$

Data kept in "higher layers" of the memory hierarchy can be accessed much faster

$\boxed{\text{But:}}$

Iterative algorithms (e.g. MG methods) on grids usually perform entire sweeps through the whole data set

$\boxed{\text{Conclusion:}}$

Iterative algorithms should take care of the underlying memory architecture in order to be efficient!

Our model problems

$$\Delta u = f \ \text{ in } \ \Omega = (0,1)^2 \ \text{ or } \ (0,1)^3,$$
$$u = g \ \text{ on } \ \partial\Omega$$

Profile of a standard 3D MG V(2,2)–cycle:

- uniform grids, 7–point stencil
- size of the finest grid: 129^3 points
- restriction: half injection
- (tri–)linear interpolation
- red/black Gauss–Seidel smoother

subroutine	time consumption
relaxation	86.0%
interpolation	8.0%
restriction	5.1%

In 2D: similar profiles

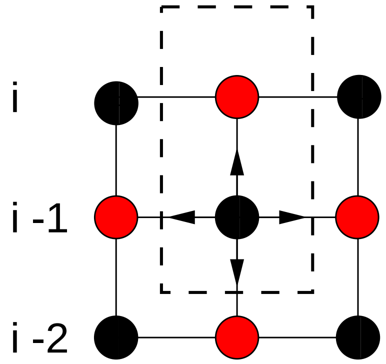
Tune the relaxation algorithm first to

optimize data locality behavior

Optimization techniques for 2D (1)

In the beginning: 60 MFlops/sec
(DEC Alpha PWS 500au, grid size 1024 $\times$ 1024)

1. Fusing relaxation of red and black points



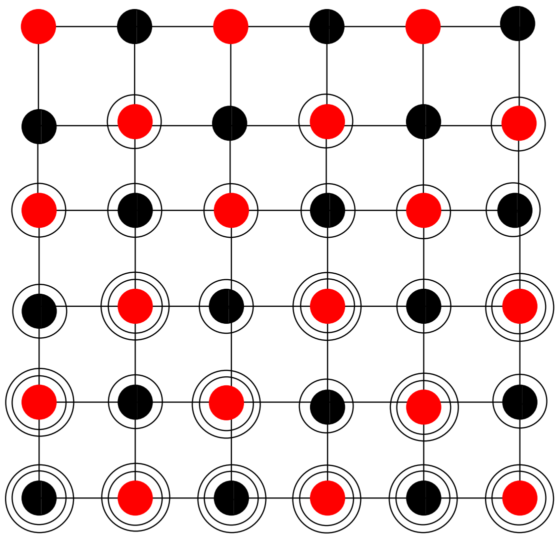
Result: 60 MFlops/sec $\longrightarrow$ 113 MFlops/sec

Optimization techniques for 2D (2)

2. Blocking of several Gauss–Seidel sweeps

By one pass through the whole grid several smoothing steps are performed

Illustration:

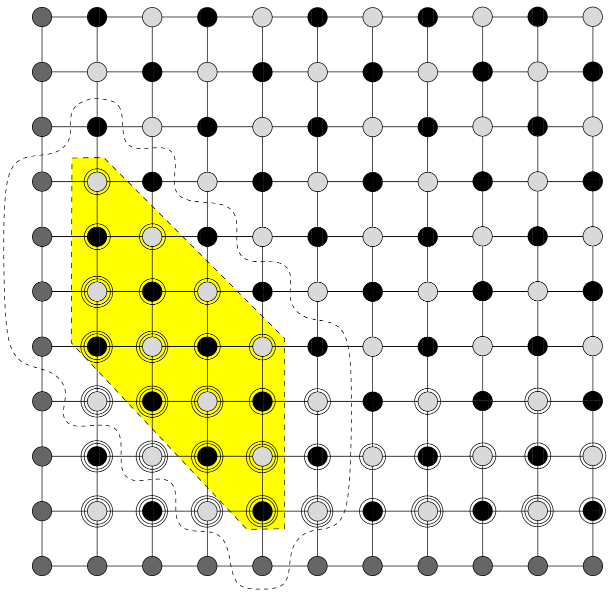


Result: 113 MFlops/sec $\longrightarrow$ 150 MFlops/sec

Optimization techniques for 2D (3)

Finally: Sophisticated data access patterns + ...

(still respecting all data dependencies)



... + array padding

```
double precision u(0:N+pad1,0:N)
double precision f(0:N+pad2,0:N)
```

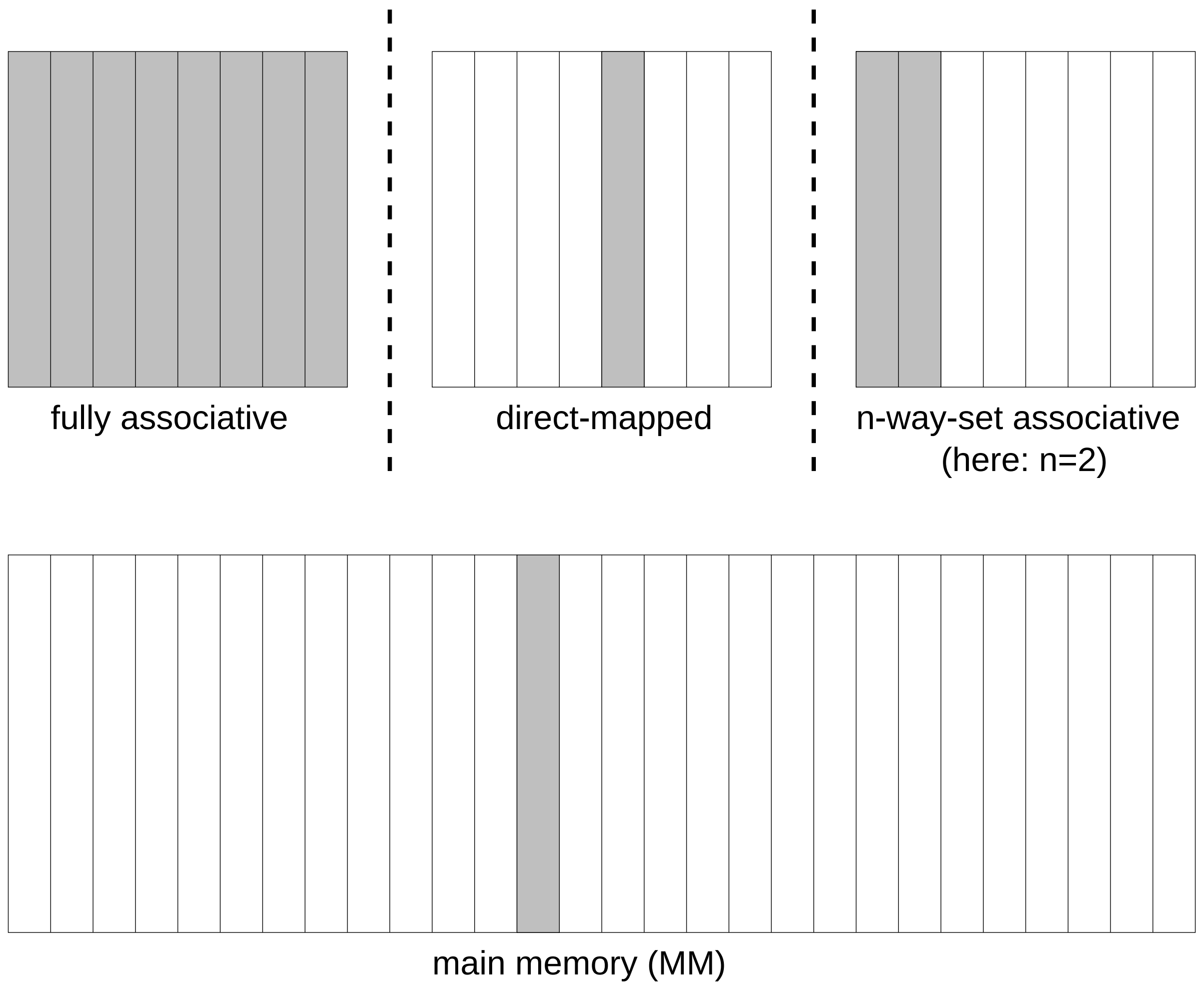
Result: 150 MFlops/sec $\longrightarrow$ 270 MFlops/sec

Why does array padding work?

Where are MM blocks placed in the cache?

Classification of cache types by their

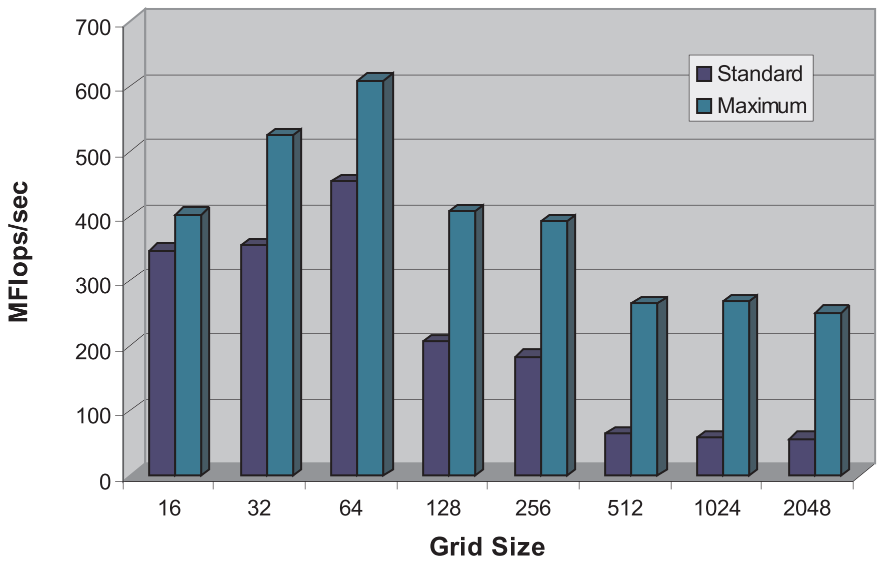
$\boxed{\text{degree of associativity}}$



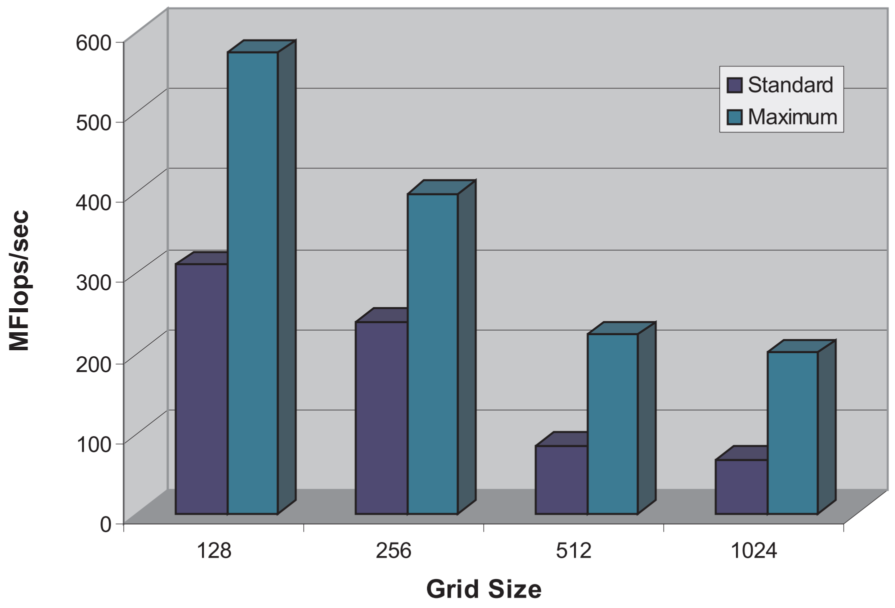
Padding changes the distance of blocks in MM and helps to avoid $\boxed{\text{cache conflict misses}}$

Speedups for 2D

Red/black Gauss–Seidel:



V(4,0)–cycle using cache optimized relaxation:



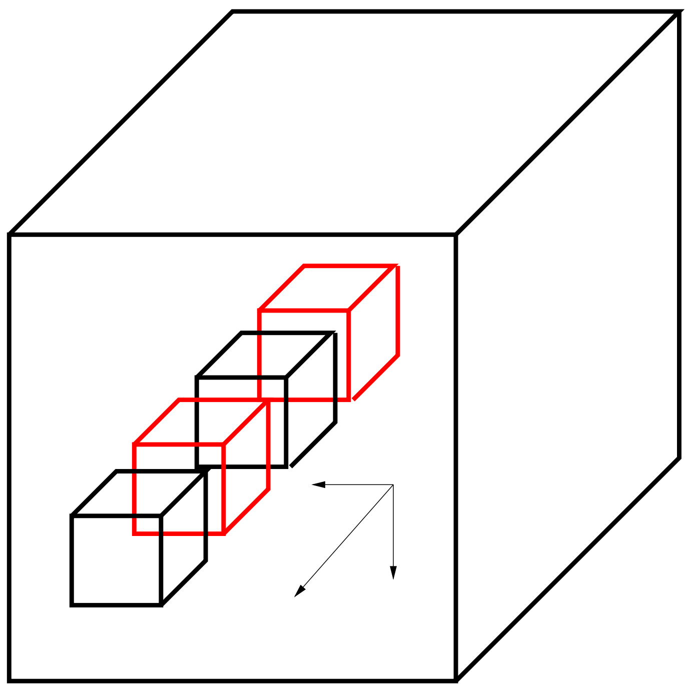
Optimization techniques for 3D (1)

Problem:

Usually caches are not big enough to hold several adjacent planes of large 3D grids

Idea:

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

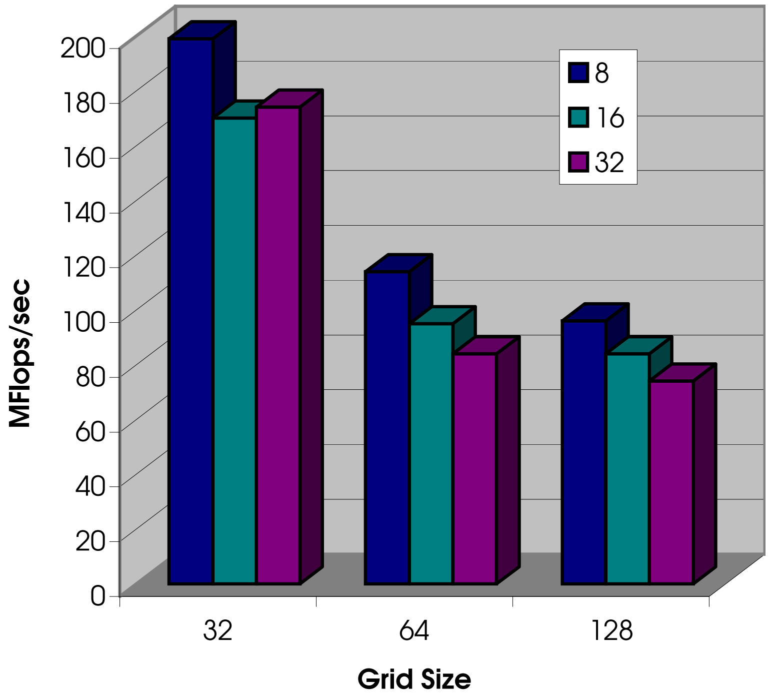


Algorithm:

```
for each subcube position within the grid
       update red points in subcube
       pos = pos + (-1,-1,-1)
       update black points in subcube
```

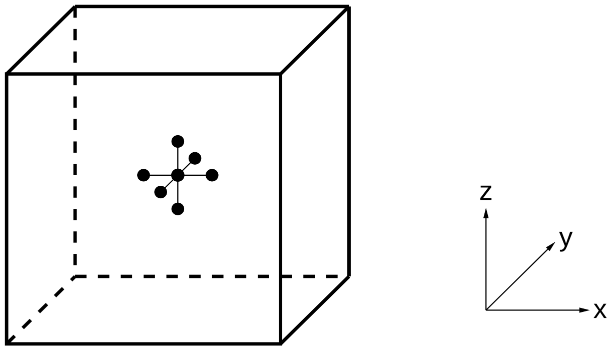
Optimization techniques for 3D (2)

Poor performance results:



This is due to a high rate of cache conflict misses and can easily be illustrated ...

Optimization techniques for 3D (3)



- 3D grid (array) with 64^3 elements
  `double precision u(0:63,0:63,0:63)`
- 8 KByte direct–mapped L1 cache
  (e.g. Alpha 21164)
- x is leading, z is trailing array dimension

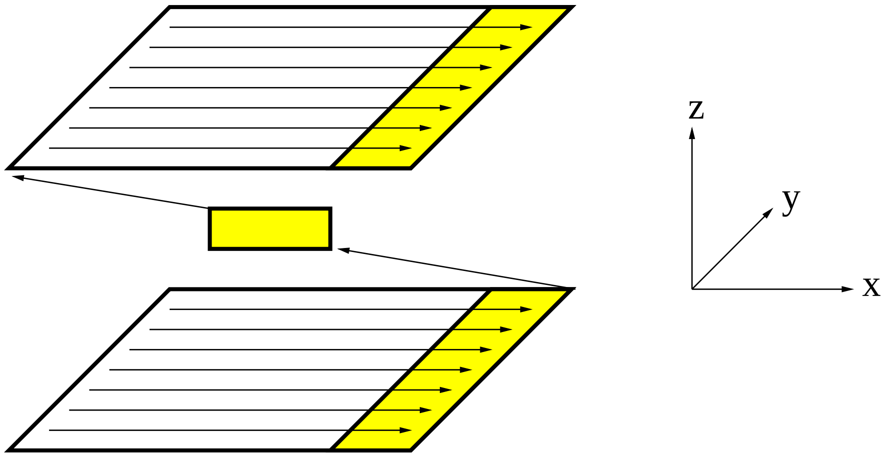
Consequences:

- 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: $\boxed{\text{conflict misses}}$

Optimization techniques for 3D (4)

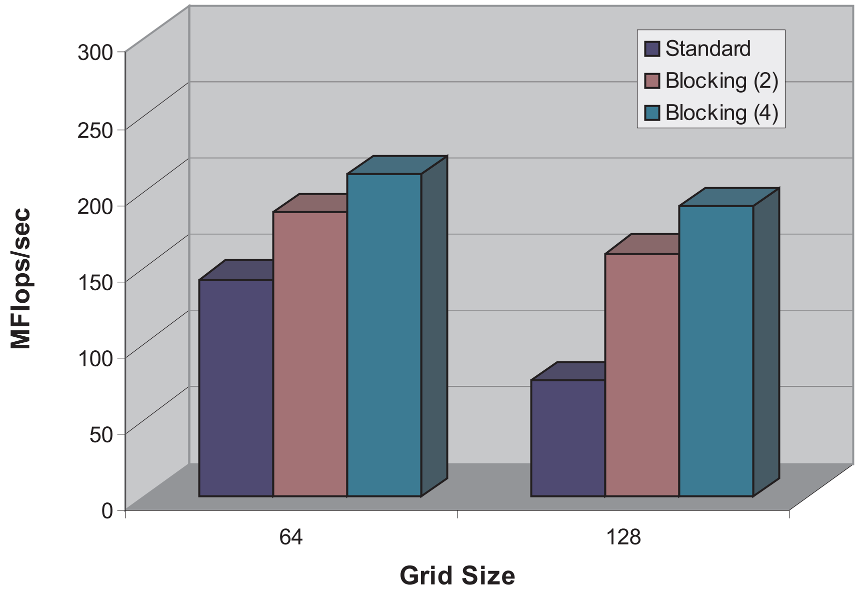
Remedy:

Multi–dimensional array padding:



Further speedups:

Blocking of several red/black Gauss–Seidel sweeps



- efficient particularly on the large grids of the hierarchy
- the relaxation of these large grids consumes most of the V–cycle execution time

Optimization techniques for 3D (5)

Performance of multigrid V–cycles (in MFlops/sec):

127^3 unknowns on the finest level

- V(2,2): only two GS sweeps can be blocked

standard	cache optimized
72	126

- V(4,0): four GS sweeps can be blocked

standard	cache optimized
73	138

Closing remarks (1)

Alpha 21264 — The Next Generation

First speedup results in MFlops/sec:

	standard	cache optimized
red/black GS	140	315
V(2,2)	184	224
V(4,0)	184	243

- Larger L1 cache (64 KByte)
- Higher memory bandwidth

Closing remarks (2)

<u>Mathematical tuning:</u>

- Higher–oder discretizations
  - More computational work
  - More accurate results
- Use of relaxation parameters
  - Possibility of applying local smoothers to moderately anisotropic problems (I. Yavneh) $\longrightarrow$ (almost) no loss of performance
  - Use of SOR smoothers can improve multigrid convergence speed

Closing remarks (3)

Solution of the model problem:

- 127^3 unknowns on the finest grid
- (tri–)linear interpolation
- 10 V(2,2)–cycles
- Maximum norm of the error:

full–weighting		half–injection
$\omega = 1.0$	$\omega{=}\omega_{\text{opt}}$	r/b GS
$9.69 \cdot 10^{-14}$	$1.88 \cdot 10^{-15}$	$1.38 \cdot 10^{-17}$

Future work

- Up to now: our efforts have focused on understanding (simple) special cases

- Extensions:
  - More general pde's (e.g. involving variable coefficients or more complicated smoothing methods)
  - Use of adaptively refined meshes
  - Application of inherently data–local algorithms (e.g. adaptive GS, U. Rüde)

- Goal: High–Performance Multigrid Library
  - Framework of cache optimized multigrid components
  - User–friendly interface