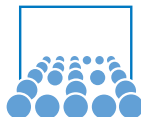


Multi- and Many-Core Data Mining with Adaptive Sparse Grids

CF'11, Ischia, Italy

Alexander Heinecke and Dirk Pflüger

May 4th 2011



Motivation - How to deal with high(er)-dimensional problems today? Sparse Grids!

- vector and multi-core architectures are state-of-the-art
- algorithms on adaptive sparse grids require a recursive structure

⇒ exploiting today's hardware is a difficult task!

But, in case of Data Mining:

- using sparse grids: complexity increases only linear with number (M) of training instance
- huge training data sets allow many parallel tasks

Outline

Sparse Grids in a nutshell

- Sparse Grids - Theory

- Data Mining with Sparse Grids - A bird-eye's view

- Data Mining with Sparse Grids - Algorithms

Benchmark results

- CPU-results (SSE and AVX)

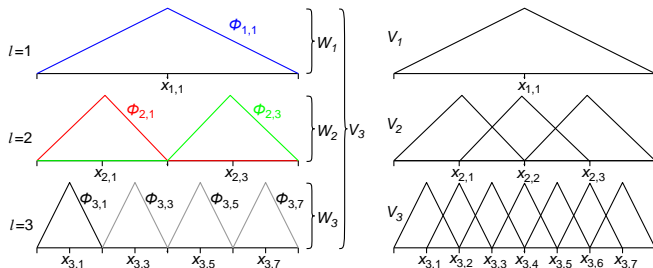
- (Multi-)GPU-results (NVIDIA Fermi)

- Hybrid CPU / GPU

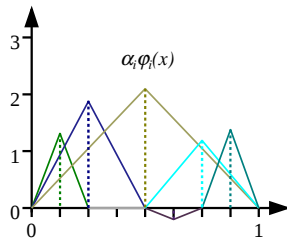
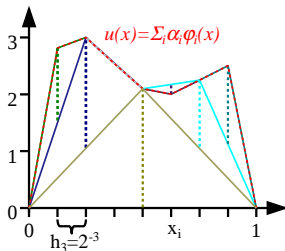
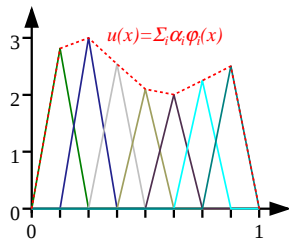
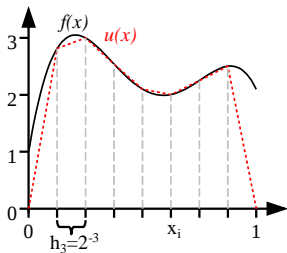
Conclusion

Sparse Grids in a nutshell

- Curse of Dimensionality: $O(N^d)$ grid points
- Therefore: Sparse Grids
 - Reduce cost to $O(N \log(N)^{d-1})$
 - Similar accuracy, if problem sufficiently smooth
- Basic idea: hierarchical basis in 1d (here: piecewise linear)

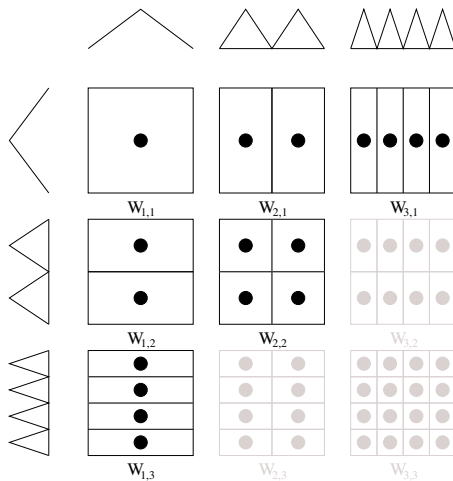


Example: Interpolation 1d



d-dimensional Sparse Grids (2D)

using a tensor-product approach we get:



Algorithm: Learning data sets with adaptive sparse grids

```
1:  $D := \text{data set}$ 
2:  $G := \text{start grid}$ 
3:  $\vec{u} := \vec{0}$ 
4: while TRUE do
5:    $\text{trainGrid}(G, \vec{u}, D)$ 
6:    $\text{acc} = \text{getAccuracy}(G, \vec{u}, D)$ 
7:   if  $\text{acc} \geq \text{acc}_{\text{needed}}$  then
8:     return  $\vec{u}, G$ 
9:   end if
10:   $\text{refineGrid}(G, \vec{u})$ 
11: end while
```

Data Mining with SG - In a nutshell

d -dimensional instances and ansatz-functions $f_N(\vec{x})$:

$$S = \left\{ (\vec{x}_m, y_m) \in \mathbb{R}^d \times K \right\}_{m=1, \dots, M} \quad f_N(\vec{x}) = \sum_{j=1}^N \vec{u}_j \varphi_j(\vec{x})$$

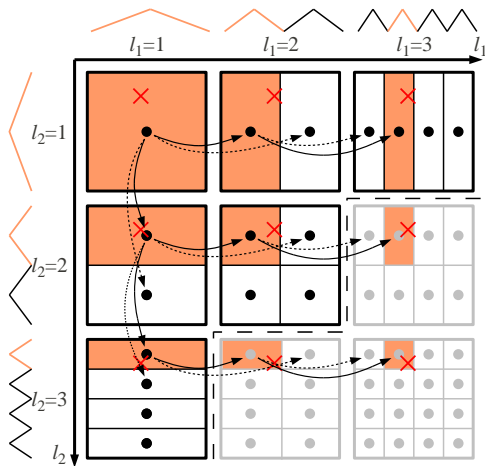
problem to be solved:

$$f_N \stackrel{!}{=} \arg \min_{f_N \in V_N} \left(\frac{1}{M} \sum_{m=1}^M (y_m - f_N(\vec{x}_m))^2 + \lambda \|\nabla f_N\|_{L^2}^2 \right)$$

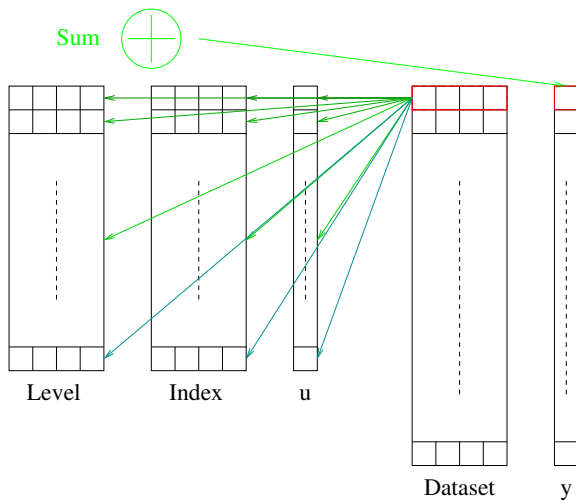
results into SLE:

$$\left(\frac{1}{M} BB^T + \lambda C \right) \vec{u} = \frac{1}{M} B\vec{y} \quad \text{here } C = I, b_{ij} = \varphi_i(\vec{x}_j)$$

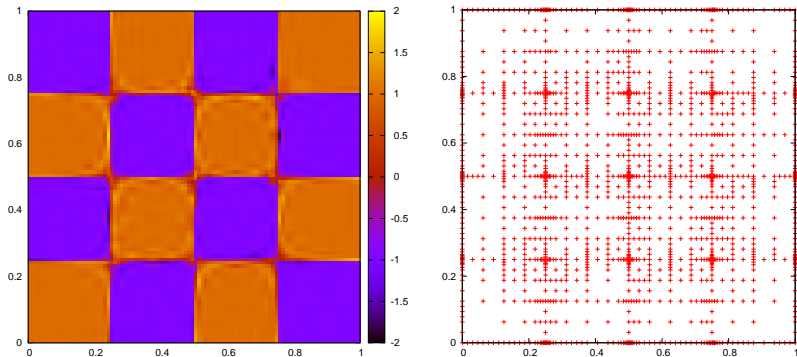
classical, recursive function evaluation



iterative function evaluation



Learning a 2D checkerboard pattern



Benchmark data sets & platforms

Benchmark data sets:

- 5D DR5, Astrophysics, ≈ 410000 instances, using 60000 as test instances
- 5D 3-field checkerboard, 2^{18} training instances, 2^{17} test instances

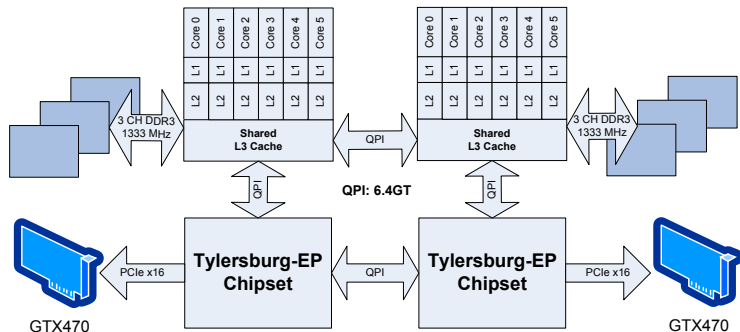
Programming languages:

- C++ with intrinsics supporting SSE, AVX
- OpenCL for GPUs
- (Intel Array Building Blocks)

Platforms:

- Intel Westmere, AMD Shanghai
- NVIDIA Fermi
- Hybrid: Intel Westmere and NVIDIA Fermi

Test-platform layout



recursive vs. iterative application of B^T (function evaluation)

```

for all  $\vec{x}_m \in S$  do
   $\vec{y}_m \leftarrow \text{recEval}(\vec{x}_m, \vec{u}, \text{grid})$ 
end for
  
```

```

for all  $\vec{x}_m \in S$  do
   $\vec{y}_m \leftarrow 0$ 
  for all  $g \in \text{grid}$  do
     $s \leftarrow \vec{u}_g$ 
    for  $k = 1$  to  $d$  do
       $s \leftarrow s \cdot \varphi_{l_k, i_k}(\vec{x}_{m_k}) =$ 
       $s \cdot \max(1 - |2^{g_{l_k}} \cdot \vec{x}_{m_k} -$ 
       $g_{i_k}|; 0)$ 
    end for
     $\vec{y}_m \leftarrow \vec{y}_m + s$ 
  end for
end for
  
```

baseline, SSE & AVX

system type		DR5	checkerboard
CPU	SIMD	time [s]	time [s]
baseline (2x Intel X5650)	rec. DP	15800	37000
baseline (2x AMD 2378)	rec. DP	59250	145000
baseline (1x Sandy Bridge)	rec. DP	56300	135100
Intel X5650	SSE SP	2300	5700
Intel X5650	SSE DP	2800	10620
AMD 2378	SSE SP	6700	11500
AMD 2378	SSE DP	7300	24000
Intel SNB	SSE SP	4850	12650
Intel SNB	SSE DP	6500	23500
Intel SNB	AVX SP	2730	7800
Intel SNB	AVX DP	3650	14000

Intel VTune Amplifier measurements (SSE, X5650)

Hardware Event Count:	408,386,000,000
CPU_CLK_UNHALTED.THREAD:	1.16642e+11
INST_RETIRED.ANY:	2.91744e+11
CPI Rate: ⓘ	0.400
Retire Stalls: ⓘ	0.153
LLC Miss: ⓘ	0.000
LLC Load Misses Serviced By Remote DRAM: ⓘ	0.000
Contested Accesses: ⓘ	0.001
Instruction Starvation: ⓘ	0.018
Branch Mispredict: ⓘ	0.001
Execution Stalls: ⓘ	0.025
Data Sharing: ⓘ	0.002
Paused Time: ⓘ	0s

OpenCL - NVIDIA Fermi

system type		DR5	checkerboard
GPU	SP/DP	time [s]	time [s]
baseline (2x X5650, rec.)	rec. DP	15800	37000
NVIDIA GTX470	SP	740	1280
NVIDIA GTX470	DP	2040	7800
2x NVIDIA GTX470	SP	410	690
2x NVIDIA GTX470	DP	1070	3970

OpenCL & SSE, Hybrid

currently:

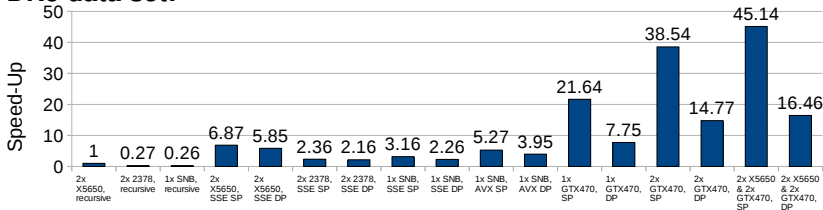
- powerful GPU \Rightarrow powerful CPU is required
- OpenCL still slow on CPU

\Rightarrow custom tailored implementation using OpenMP tasks

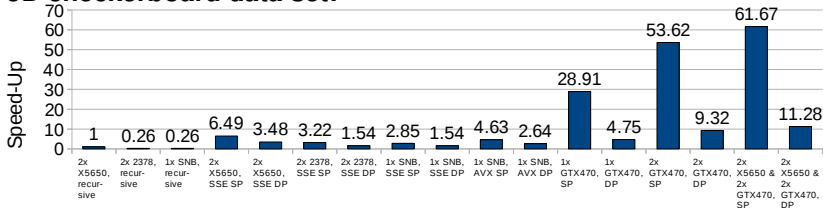
system type		DR5	checkerboard
CPU/GPU	SP/DP	time [s]	time [s]
baseline (2x X5650, rec.)	rec. DP	15800	37000
2x NVIDIA GTX470	SP	410	690
2x NVIDIA GTX470	DP	1070	3970
2x X5650, 2x GTX470	SP	350	600
2x X5650, 2x GTX470	DP	960	3280

Data Mining, concluding results

DR5 data set:



5D checkerboard data set:



Conclusion and Outlook

Takeaways:

- using simple algorithms on vector architectures is crucial
- they may be able to outperform complexity-optimal versions
- (sustained) performance gap between GPU and CPU is small
- hybrid executions should be considered nowadays
- a converged computing model/language is needed
- checking the needed precision might be useful

Future Work:

- testing other ansatzfunctions (different boundary handling, polynomials)
- analyzing application with respect to Intel's MIC Platform
- MPI parallelization in order to exploit (huge) CPU/GPU clusters