Towards a highly-parallel PDE-Solver using Adaptive Sparse Grids on Compute Clusters

HIM - Workshop on Sparse Grids and Applications

Alexander Heinecke
Chair of Scientific Computing

May 18th 2011

Motivation

Sparse Grids have been successfully applied to following fields:

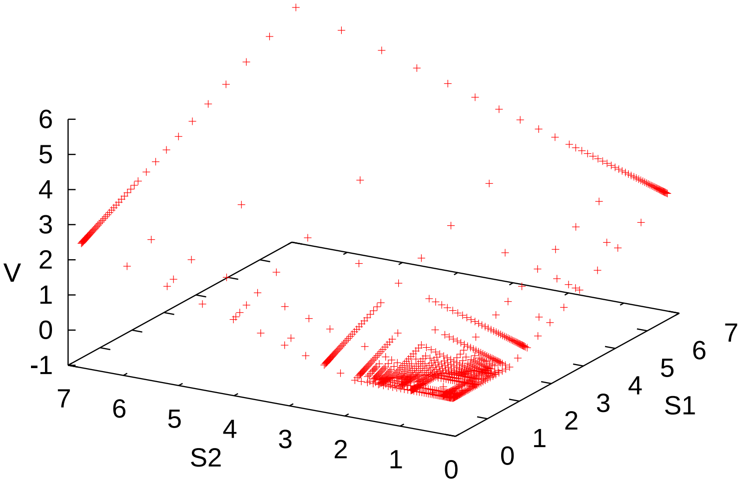
- Interpolation
- Data Mining
- Solving PDEs
- and many more

Adaptivity!

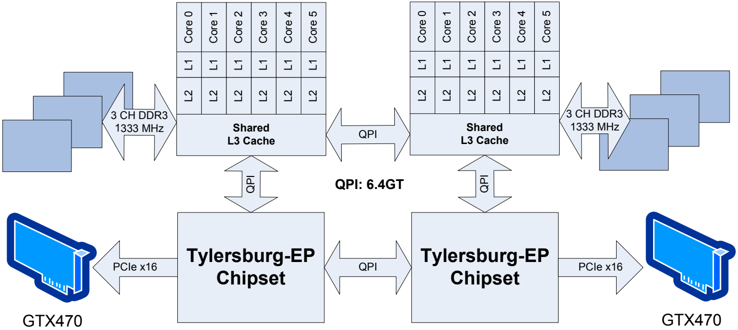
Performance?

HPC $\Leftrightarrow$ Spatially Adaptive Sparse Grids?

Motivation - We Need Adaptive Grids!



Motivation - We Need to be Hardware Aware!



Outline

- Overview
- parallel solution of PDEs
  - shared memory systems
    - results
  - distributed memory systems
    - results
- additional possibilities
- Conclusion

Solving PDEs - An Overview

- second huge application domain of our SG-code (SG^{++})
- currently available: Poission equation, heat equation, Black-Scholes equation, Hull-White equation
- solution done by finite elements

- currently available: highly optimized shared memory implementation
- execution times still too high

$\Rightarrow$ currently “under construction”: hybrid parallelization on distributed and shared memory systems

Some Basic Definitions...

- $\phi_{\vec{l},\vec{i}}$ are some suitable Sparse Grid basis functions (e.g. piecewise linear)
- where needed: ϕ_j with a suitable hierarchical ordering
- $\vec{u}$ contains the grid's coefficients (hierarchical surpluses)
- N is the number of grid points
- there is a way of addressing basis functions based on level and index (hashmap)
- there are refinement and coarsening algorithms based on the hierarchical surplus in order to allow spatially adaptive sparse grids
- in case of adaptive sparse grids a “well-formed” grid is assumed

A Parallel PDE Solver, I of II

$G :=$ *initial grid*
$p(\vec{x}) :=$ *function for initial condition*
refineGrid$(G, \vec{u}, p(\vec{x}))$
for $t = 0$ to T **do**
    solveTimeStep$(G, \vec{u}, \delta t)$
    restructureGrid$(G, \vec{u})$
    $t \leftarrow t + \delta t$
end for
return $\vec{u}, G$

- CG/BiCGSTAB-solver consumes $\approx$ 100% of solver time
- grid manipulation can also be done in parallel

A Parallel PDE Solver, II of II

$i := 0$
$r := b - Lx$
$d := r$
$\delta_0 := r^T r$
while $i < i_{max}$ AND $\delta > \varepsilon^2 \cdot \delta_0$ **do**
    $q = Ld$
    update $x, d, delta, i$
end while

- MV-Product consumes $\approx$ 100% of solver time
- MV-Product can be parallelized (Up/Down scheme)

Application of a PDE's FEM System Matrix

Calculating the 1D-L_2 scalar-product:

$$\sum_{n=1}^{N} u_n \cdot \int_x \phi_n(x)\phi_m(x)dx$$

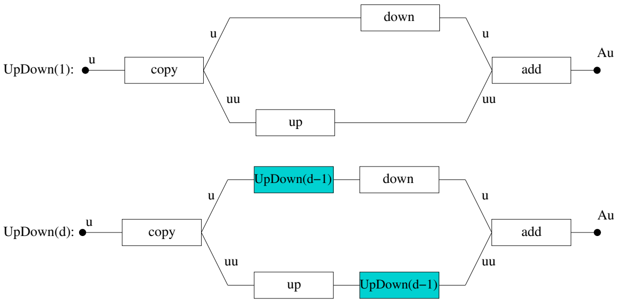
$$= \int_x \sum_{n\leq m}^{N} u_n\phi_n(x) \cdot \phi_m(x)dx + \int_x \sum_{n>m}^{N} u_z\phi_n(x) \cdot \phi_m(x)dx, \quad \forall m = 1,\dots,N$$

and assuming a suitable ordering $\phi_{1,1}, \phi_{2,1}, \phi_{2,3}, \phi_{3,1}, \dots$.

matrix equation that follows:

$$A\vec{u} = A^{Down}\vec{u} + A^{Up}\vec{u}.$$

Application of a PDE's FEM System Matrix



- Up/Down parallelization constructs 2^d tasks
- Barriers: Just for more than 5d: up to 12 cores usable

⇒ Up/Down parallelization only for moderate core counts suitable
⇒ Today's HPC-Boxes (up to 80 HW-threads) cannot be exploited

Some Code: Up/Down Parallel

```
int size = u.getSize();
Vector temp(size),result_temp(size),temp_two(size);

#pragma omp task shared(u, result)
{
    up(u, temp, dim);
    updown(temp, result, dim-1);
}
#pragma omp task shared(u, result_temp)
{
    updown(u, temp_two, dim-1);
    down(temp_two, result_temp, dim);
}
#pragma omp taskwait

result.add(result_temp);
```

Example: Heat Equation

Laplacian: calculated by sum of 1d-Operators B:

$$A\dot{\vec{u}}(t) = \underbrace{\sum_{i=1}^{d} B_i}_{:=L} \vec{u}(t)$$

Implicit Euler:

$$(A - \delta tL)\underbrace{\vec{u}(t+1)}_{\vec{x}} = \underbrace{A\vec{u}(t)}_{\vec{b}}.$$

$\Rightarrow$ more parallelism: $\mathbf{d+1}$ additional summands!

Black-Scholes Parallelized in Shared Memory

Black Scholes Equation (for Basket-Options) reversed in time:

$$\frac{\partial V}{\partial \tau} - \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_i\sigma_j\rho_{i,j}S_iS_j\frac{\partial^2 V}{\partial S_i \partial S_j} - \sum_{i=1}^{d}\mu_i S_i \frac{\partial V}{\partial S_i} + rV = 0$$

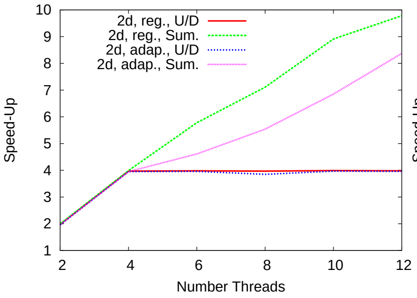
FEM matrix equation:

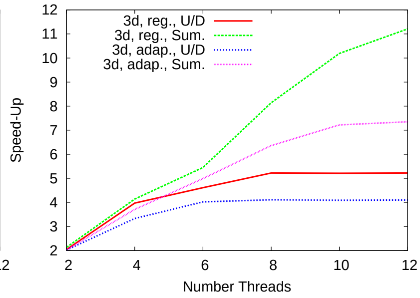
$$A\dot{\vec{u}}(\tau) \quad = \quad \underbrace{\left(\sum_{i=1}^{d} \nu_i \cdot F - \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_i\sigma_j\rho_{i,j}\cdot E + r\cdot A\right)}_{:=L}\vec{u}(\tau)$$

with $\nu_i = \mu_i - \sum_{j=1}^{d}\left(\frac{1}{2}\sigma_i\sigma_j\rho_{i,j}(1+\delta_{i,j})\right)$

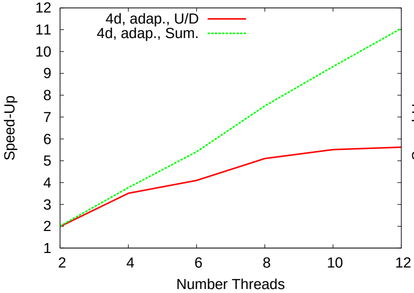
$\Rightarrow \mathbf{\frac{d^2+3d}{2}} + \mathbf{2}$ summands (including symmetry)!

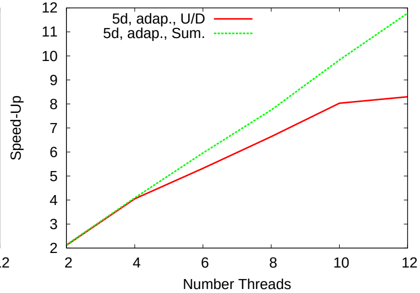
Black-Scholes: OpenMP, I of II





Black-Scholes: OpenMP, II of II





PDEs parallel: Hybrid parallelization

current status:

- solution of higher dimensional BS still too slow
- only shared memory systems can be used

“under construction”:

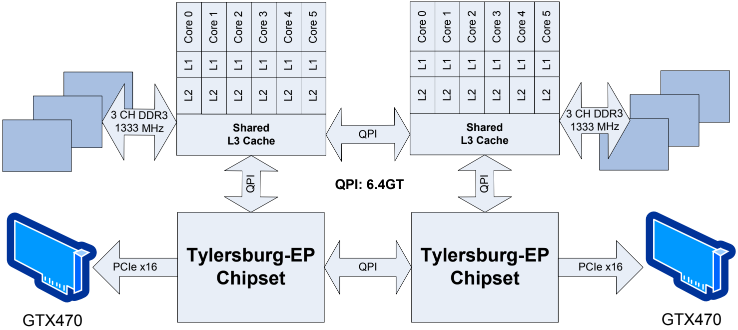
- sum-parallelization suitable for MPI
- Up/Down parallelization only in shared memory

Goal:

- better use of NUMA
- solving on compute clusters

Testplatforms - Revisited

- Now: Use hardware threading for latency hiding (incl. pinning)
- Now: GPU's cannot be exploited (no data parallelism)



PDEs parallel: First Results Poisson (reg. Grids)

For a current high-end workstation:

2x Xeon X5650 2.66GHz	12 Thr. no HT	24 Thr. HT	2 Pr. 6 Thr. no HT	2 Pr. 12 Thr. HT
6D-Poisson (8l)	1390s	1050s	1250s	905s

SGI ICE (2x Xeon E5540 per Blade, HT, Infiniband):

Xeon E5540 2.53GHz	2 Pr. 8 Thr. 1 node	6 Pr. 8 Thr. 3 nodes
6D-Poisson (8l)	1270s	630s

SGI UV (2x Xeon X7550 per Blade, SGI NUMALink):

Xeon X7550 2.00GHz	2 Pr. 8 Thr. (no HT)	6 Pr. 8 Thr. (no HT)
6D-Poisson (8l)	2050s	1200s

SGI UV: MPI-Pinning not possible :-(

PDEs parallel: First Results Poisson (reg. Grids)

For a current high-end workstation (8D-Poisson):

2x Xeon X5650 2.66GHz	12 Thr.	24 Thr. (HT)
8D-Poisson (6l)	4700s	3300s (-30 %)

SGI ICE (2x Xeon E5540 per Blade, no HT, Infiniband):

Xeon E5540 2.53GHz	2 Pr. 8 Thr. 2 Knoten	4 Pr. 8 Thr. 4 Knoten	8Pr. 8Thr. 8 Knoten
8D-Poisson (6l)	4400s	2600s	1400s

PDEs parallel: First Results Poisson (adapt. Grids)

For a current high-end workstation:

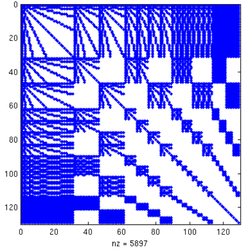
2x Xeon X5650 2.66GHz	12 Thr. no HT	24 Thr. HT	2 Pr. 6 Thr. no HT	2 Pr. 12 Thr. HT
6D-Poisson (adapt.)	1200s	890s	1050s	800s

SGI ICE (2x Xeon E5540 per Blade, HT, Infiniband):

Xeon E5540 2.53GHz	2 Pr. 8 Thr. 1 node	6 Pr. 8 Thr. 3 nodes
6D-Poisson (adapt.)	1530s	790s

PDEs parallel: Beyond Up/Down-Parallelism

- TifaMMy: support for hybrid matrices (neither dense nor sparse)
- TifaMMy: (fast) ILU preconditioner



- parallel (hierarchical) preconditioner?

Conclusion and Future Work

Conclusion

- proposal for a parallelization approach of the Up/Down-scheme with respect to today's hardware
- successful demonstration with several prototypes

Future Work

- reduce boundary overhead
- implementing BS-Equation with MPI/OpenMP parallelization scheme
- trying TifaMMy (hybrid matrices)
- preconditioning