Karlsruhe Institute of Technology

DATE Friday Workshop

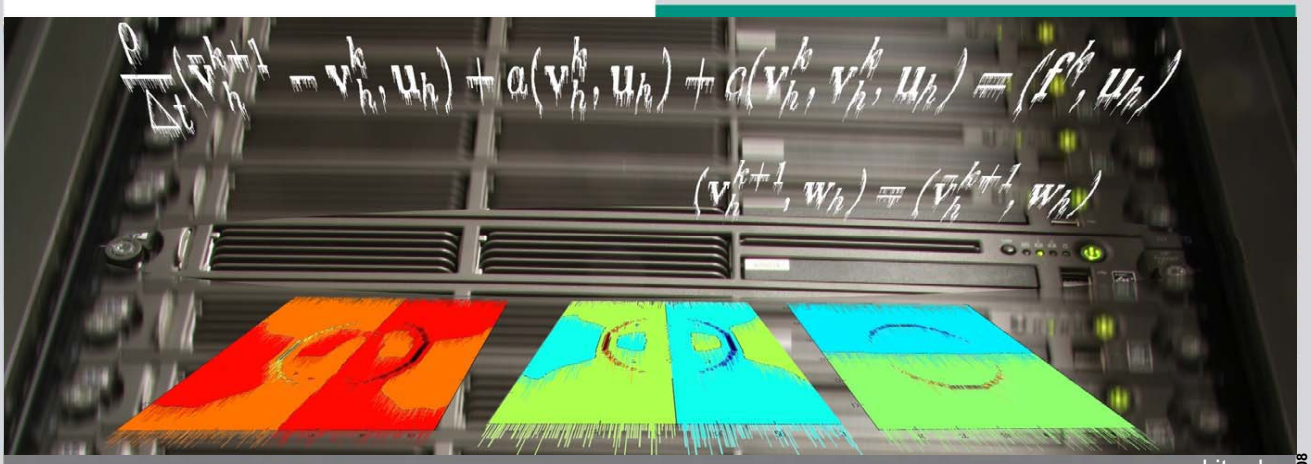
April 24, 2009

Computational Fluid Dynamics on Multicore Architectures

Jan-Philipp Weiss

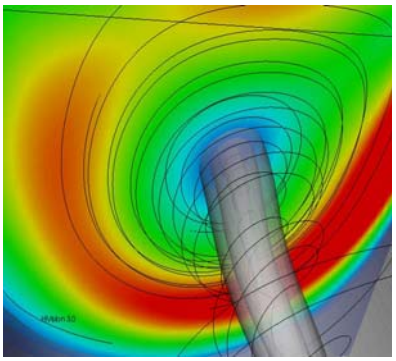
Forschungszentrum Karlsruhe in der Helmholtz-Gemeinschaft

Universität Karlsruhe (TH) Forschungsuniversität • gegründet 1825



Outline

- Motivation
  - Mathematical cognition vs. hardware reality
- Application prototype of computational fluid dynamics
  - Model problem
  - Algorithmic characteristics
  - Considerations for implementation
- Performance model and performance bounds
- Theoretical and practical performance results



Motivation

- Computational fluid dynamics generates complex problems
  - Mathematical modeling
  - Parallel implementation
  - Data layout
  - Memory consumption and bandwidth utilization
  - Computational load
  - Reliability: accuracy, reproducibility, system stability

Accelerators and coprocessors shall be used
for speeding up computations!

What are the key issues?

Mathematics vs. Reality

What mathematics is telling …

- Use locally adapted grids to resolve solution characteristics
  - Singularities, accumulated errors, huge domains,…
- Optimize number of degrees of freedom by redistribution
  - Adaptive error control and goal oriented meshing
- Benefits in results outweigh grid management overhead (70%?)
  - Indirect addressing, pointer chasing, …
- Implicit solution methods are more favorable than explicit methods
  - Stability and time step limitations

Good strategy for x86 (or related) multi-core CPUs
with automatic data transfers and caching

Mathematics vs. Reality

What accelerator reality is answering …

- Explicit memory transfers require predictable and structured memory access patterns
- Bandwidth bottleneck can only be mitigated by contiguous, coalesced and aligned memory access
  - No random or indirect access to memory
- Explicit methods allow temporal blocking techniques
  - Data reuse across several time steps

Algorithms need to be uniformly structured and simple
for best throughput on accelerators

Chorin-based Navier Stokes Solver

Model problem

- Incompressible Navier-Stokes equations

$$\partial_t u - \nu \Delta u + (u \cdot \nabla) u + \nabla p = f \quad \text{in } \Omega$$

$$\nabla \cdot u = 0 \quad \text{in } \Omega$$

- Chorin-type projection method: iterative scheme, k=0, 1, …
  - Compute $\tilde{u}^{k+1}$: $\dfrac{\tilde{u}^{k+1} - u^k}{\Delta t} + (u^k \cdot \nabla) u^k - \nu \Delta u^k = f^k \quad \text{in } \Omega$
  - Compute p^{k+1} : $\Delta p^{k+1} = \dfrac{1}{\Delta t} \nabla \tilde{u}^{k+1} \quad \text{in } \Omega$
  - Compute u^{k+1} : $\dfrac{u^{k+1} - \tilde{u}^{k+1}}{\Delta t} = -\nabla p^{k+1} \quad \text{in } \Omega$

Chorin-based Navier-Stokes Solver
Algorithm characteristics

- Finite-difference discretization on staggered grids
- Global coupling between all variables

- Advection step
- Projection step
  - Solve Poisson equation for the pressure
  - Most time-consuming part
  - Sparse matrices or stencils
  - Use (preconditioned) conjugate gradient (CG) method (or multigrid solver)
- Velocity update step

Chorin-based Navier Stokes Solver
Algorithm characteristics

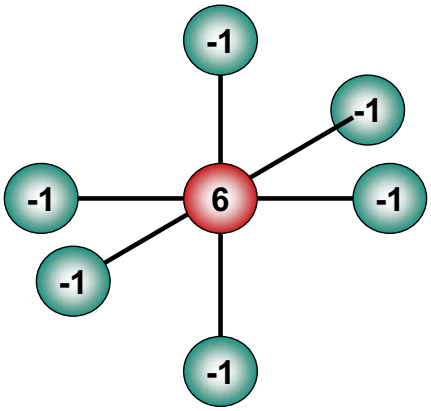
- Four basic operations (typical of solution of PDEs)
  - Sparse matrix-vector multiplication / stencil
  - (Non)-linear stencil operations
    - Nearest neighbor interaction
  - SAXPY / DAXPY vector updates
  - Dot products
- Encapsulated by iterative methods and time stepping schemes

- Characteristics
  - Huge memory requirements / fine mesh resolution
  - Low computational intensity of order O(1)
  - Bandwidth-bound algorithms
  - Non-uniform treatment of boundary conditions

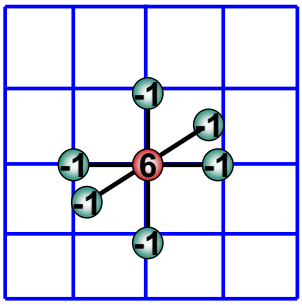
3D Laplace stencil

- Each node associated with a single value
- Each data touched seven times
- Implicit matrix-vector multiplication

$$6u_{i,j,k} - u_{i+1,j,k} - u_{i,j+1,k} - u_{i,j,k+1}$$
$$- u_{i-1,j,k} - u_{i,j-1,k} - u_{i,j,k-1} = h^2 f_{i,j,k}$$

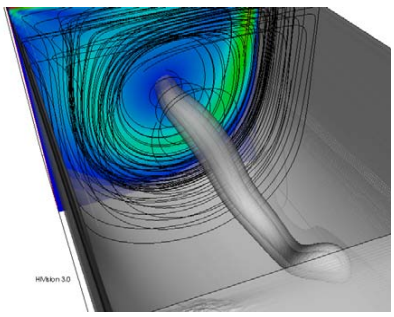


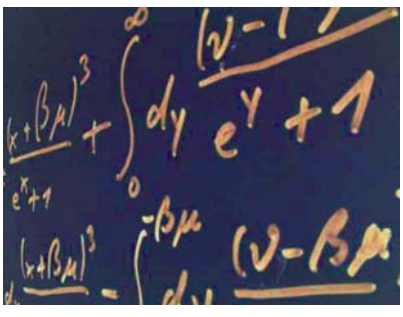
Blocking strategies required for data reuse



Observation

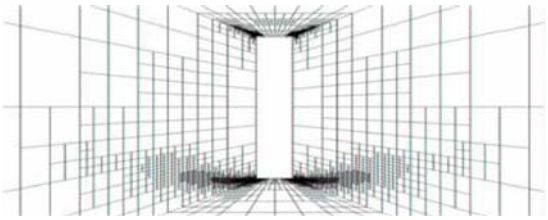
- Problems of interest are bandwidth-bound
- Dedicated strategies need to be applied to achieve good performance
- Algorithms need to be designed structured and simple for best throughput on accelerators





Memory transfer reduction strategies

- Reduction of precision
  - 64-bit (double) – (32-bit) single – lower-bit
  - Higher memory throughput
  - May also speed up computations (e.g. in vectorization)
  - But what about mathematical quality?
- Uniform grids
  - Stencils instead of matrices / implicit matrices
  - No matrix data and addressing schemes transferred
  - Kernels computed on the fly



Memory transfer reduction strategies

- Increase spatial locality
  - Substructure operations
  - Cache blocking – register blocking – TLB
- Increase temporal locality
  - Keep local copies of data across several iterations
  - Time-skewing
  - Circular queue

- No redundant data transfers
- Inherently sequential
- Panel sizes vary
- Imbalanced work
- Difficult overlap

- Sizes of panels shrink in each step
- Redundant computation and transfers
- Easily parallelizable

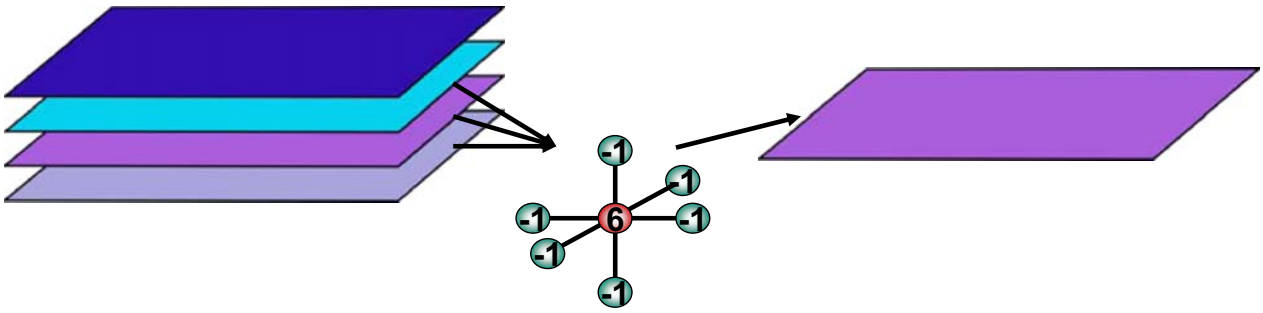
- Temporal blocking
  - Often referenced in many benchmarks
  - Limited practical applicability – only explicit schemes
  - Cannot be used in CG-method
  - Well suited for some parabolic problems (e.g. heat equation)

Stencil operations

- Easily parallelizable by domain sub-structuring
- Two different approaches for implementation:

- Streaming in and out planes
  - Reduces data exchange
  - Requires intermediate data exchange
  - Queuing planes for overlapping data transfers and computation
  - Data exchange in 2 or 4 directions

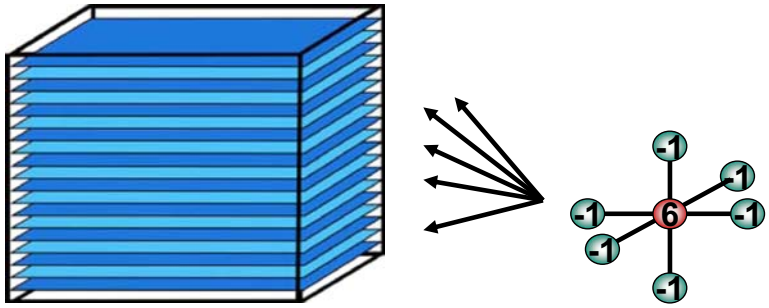
Applicable to stream processing?



Stencil operations

- Easily parallelizable by domain sub-structuring
- Two different approaches for implementation:

- Blocking in sub-cubes
  - No intermediate data exchange
  - Smaller planes
  - Worse surface-to-volume ratio / larger overlap
  - Data exchange in 6 directions / discontinuous memory access



Simple performance model

Run time : $T_R \geq \max \{ T_C , T_T \}$

- T_R … total run time of an algorithm
- T_C … compute time, $T_C \geq f / P$
- T_T … transfer time, $T_T \geq 8w / B$
- Classifies compute-bound or memory-bound algorithms

- Algorithm characteristics
  - f … number of floating point operations
  - w … number of memory transfers (words)
- Hardware characteristics
  - P … peak performance
  - B … memory bandwidth

- We find: effective performance $P_{eff} \leq f / T_R \leq fB / (8w)$
  - Ratio f / w defines computational intensity

Elements of projection step

- LSE in projection step solved with conjugate gradient method (cg)
- Number of cg-steps per time step depends on $N^{1/3}$
- Each cg-step consists of

Function	Occ.	f [#flop]	w [#words]	f / w Comp. int.
Stencil operation	1	8N	2N	4.0
Vector norm	1	2N-1	N+1	2.0
Dot product	1	2N-1	2N+1	1.0
Normalization	4	2N	2N	1.0
DAXPY vector update	3	2N	3N+1	0.66

- Worst performance for DAXPY vector update
  - Easy routine with no data dependencies
  - Perfectly parallelizable on coarse and fine-grained platforms

Performance bounds on hardware

- Flop per byte ratio on hardware (in double precision):

	# Cores	Peak Perform. P	Bandwidth B
AMD Opteron dual core	2	8.8 GFlop/s	10.6 GB/s
Intel Clovertown quad core	4	37.2 GFlop/s	10.6 GB/s
Sun Niagara T2 octo core	8	11.2 GFlop/s	42.6 GB/s
Cell (2nd gen.)	8+1	102.4 GFlop/s	25.6 GB/s
ClearSpeed CSX600	96	55.0 GFlop/s	3.2 GB/s
nVIDIA GeForce GTX 280	240	80.0 GFlop/s	140.2 GB/s

(Theoretical values)

DAXPY performance on hardware

- In theory, DAXPY performance is only limited by bandwidth

	Bandwidth	DAXPY	Experimental
Intel Clovertown quad core	10.6 GB/s	0.9 GFlop/s	0.54 GFlop/s
AMD Opteron dual core	10.6 GB/s	0.9 GFlop/s	0.34 GFlop/s
Sun Niagara T2 octo core	42.6 GB/s	3.5 GFlop/s	1.36 GFlop/s
Cell (2nd gen.)	25.6 GB/s	2.1 GFlop/s	1.70 GFlop/s
ClearSpeed CSX600	3.2 GB/s	0.3 GFlop/s	
nVIDIA GeForce GTX 280	140.2 GB/s	11.7 GFlop/s	9.60 GFlop/s

(Theoretical / experimental values)

DAXPY efficiency

- Theoretical DAXPY efficiency

	Performance	DAXPY	Efficiency
Intel Clovertown quad core	37.2 GFlop/s	0.9 GFlop/s	0.02
AMD Opteron dual core	8.8 GFlop/s	0.9 GFlop/s	0.10
Sun Niagara T2 octo core	11.2 GFlop/s	3.5 GFlop/s	0.31
Cell (2nd gen.)	100.0 GFlop/s	2.1 GFlop/s	0.02
ClearSpeed CSX600	55.0 GFlop/s	0.3 GFlop/s	0.01
nVIDIA GeForce GTX 280	80.0 GFlop/s	11.6 GFlop/s	0.14

(Theoretical values)

Memory considerations

- Main memory size
  - GPUs 512 MB to 4 GB
  - Cell 256 MB to 16 GB
  - ClearSpeed 512 MB to 2 GB

3D simulation?

- Local memory size
  - Cell 256 KB (private)
  - nVIDIA GPUs 16 KB (shared)
  - ClearSpeed 6 KB (private)

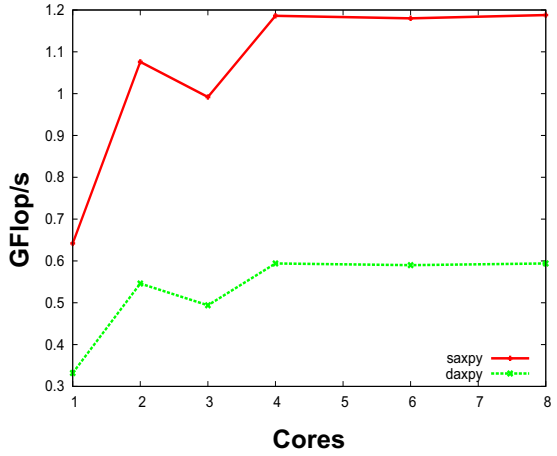
Spatial blocking?

- Memory layout
  - Data alignment
  - Memory access patterns
  - NUMA effects

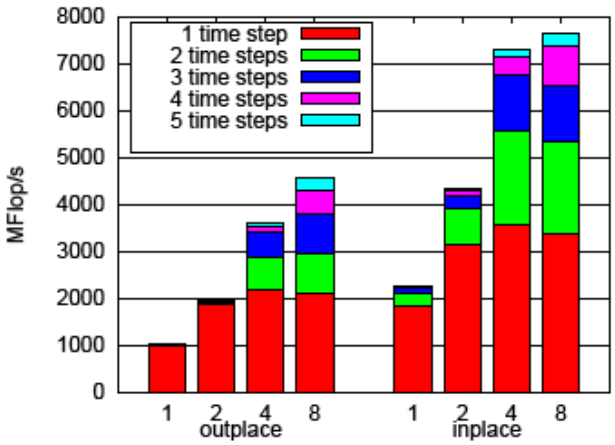
Performance on Multicore-CPUs

- Intel Xeon Cluster (X5355 Clovertown), 1 node, 2 sockets, 8 cores
- 2.7 GHz, 16 GB per node, 2 GB per core

DAXPY / SAXPY



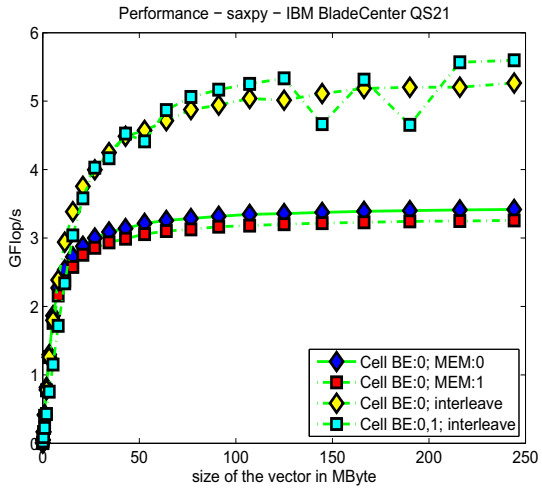
Stencil



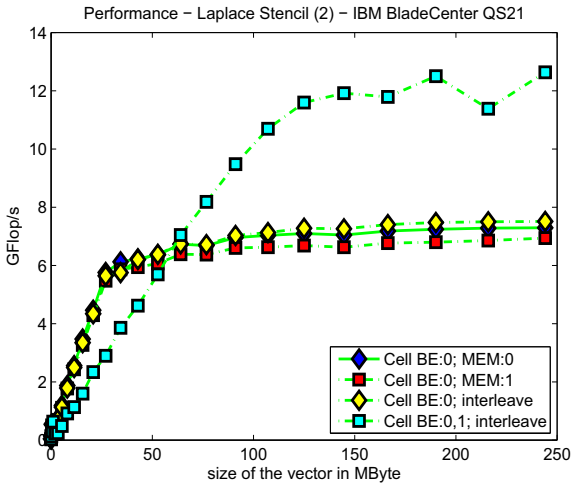
Performance on Cell

- BladeCenter QS21 / NUMA with RapidMind (single precision)
- Transfer cubes of size 16^3, N=268^3 (73 MByte)

SAXPY



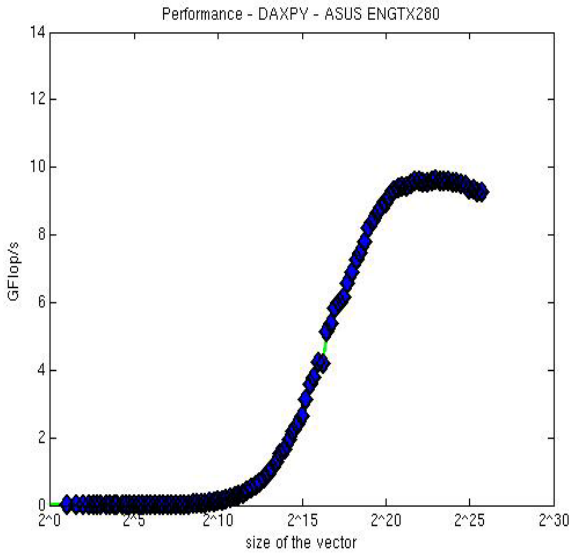
Stencil (3D CFD)



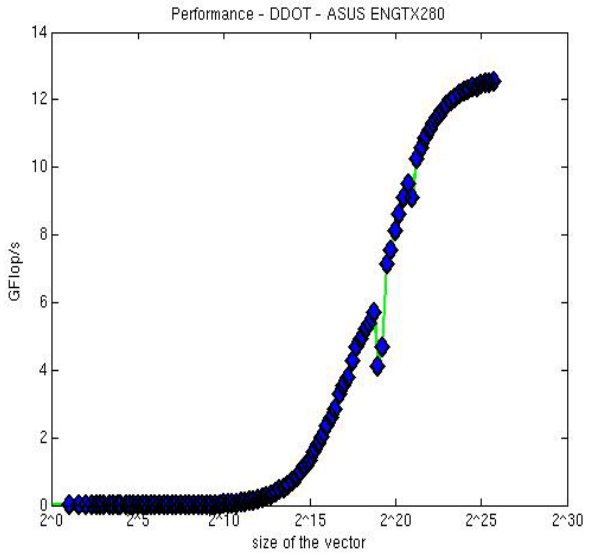
Performance on GPU

- nVIDIA GTX 280 GPU with 240 stream processors, CUDA

DAXPY



Dot product



Summary

- Don’t get blinded by pure GFlop/s performance numbers
  - What are BLAS 3 kernels good for?
  - Be aware of mathematical quality of the implementation!
- Bandwidth and its efficient utilization are far more important
- Critical issues
  - Data and memory layout
  - Coalesced and aligned memory access
  - Blocking techniques for data re-use
- Memory size and slow connection to host are limiting factors for accelerators in applications of computational fluid dynamics

Acknowledgements

- The SRG is granted by Hewlett-Packard and the Concept for the Future of the Karlsruhe Institute of Technology (KIT) within the framework of the German Excellence Initiative.

- Thanks to Werner Augustin, Dimitar Lukarski, and Michael Rückauer.

Contact

Further information available at

- http://srg-multicore.rz.uni-karlsruhe.de
- jan-philipp.weiss@kit.edu

Thank you for your attention.