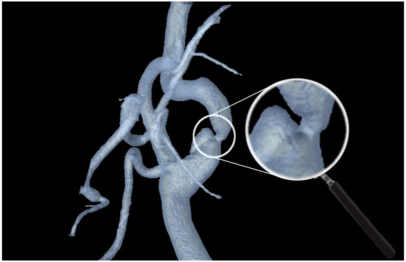
Fluid Simulation using the Lattice Boltzmann Method on the Cell Processor

Markus Stürmer

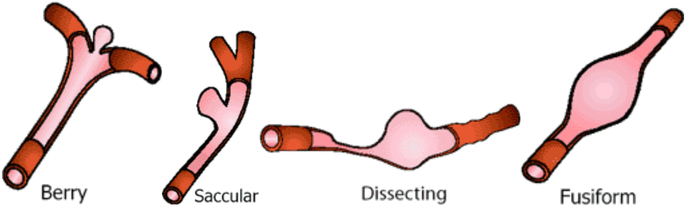
University Erlangen-Nuremberg – System Simulation

April 11th 2007 @ FZ Jülich



Aneurysms

- dilatation (local balooning) of the vessel
- localized mostly at larger arteries in soft tissue (e. g. aorta, brain)
- 3-5% of all people in Germany have an aneurysm
- if an aneurysm ruptures 33% of the patients are dying and 33% suffer a physical handicap



Problem Description

Goals

blood flow simulation

- to help in understanding the development of aneurysms
- to support planning of therapy

Problem Description

Goals

blood flow simulation

- to help in understanding the development of aneurysms
- to support planning of therapy

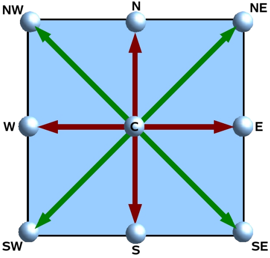
Challanges

- current imaging techniques (CT, MRI, Angiography) result in data sets of 512^3 and more
- long run times on desktop PCs and workstations

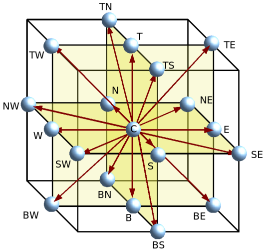
Lattice Boltzmann Method

- fluid is described as collection of particles
- domain is divided into quadratic (2D) or cubical (3D) lattices
- every cell (lattice) has a set of particle velocity distribution functions
- simple timestep consists of two steps
  1. stream – “move” of distribution to fluid neighbors
  2. collide – “particle collision” leads to new distribution functions
- cellular automata, interaction only with neighbors

D2Q9



D3Q19

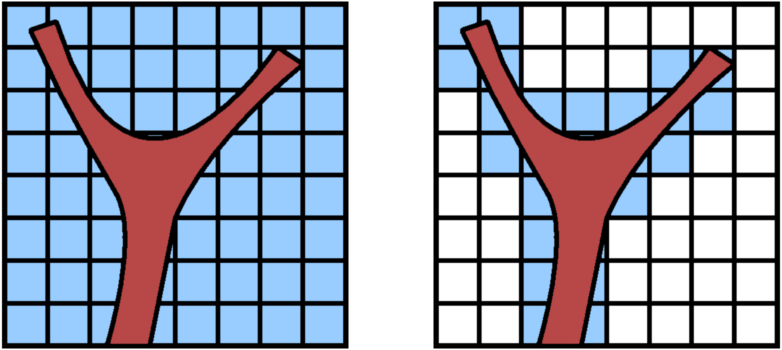


Outline

- memory layout
  - domain splitting
  - parallelization and data exchange
- SIMDization of stream step
- overview of execution on SPU
- performance
  - Rev. B Cell Blade
  - PS3
- outlook

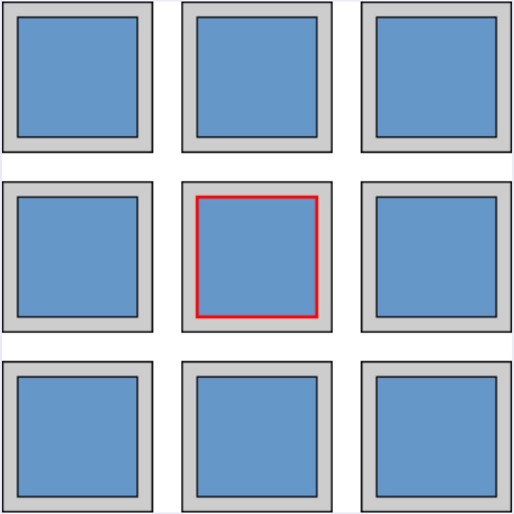
Domain Splitting

Divide whole domain into equally sized blocks ($8 \times 8 \times 8$) and only allocate and calculate blocks including fluid cells.



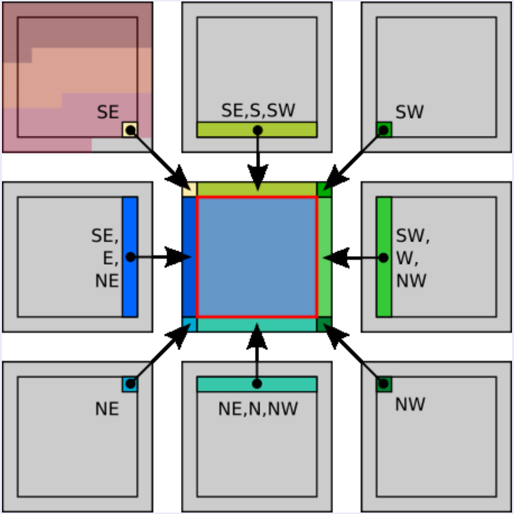
“Standard” Ghost Cell Approach

2D 8 × 8 example



"Standard" Ghost Cell Approach

2D 8 × 8 example revisited



“Standard” Ghost Cell Approach

3D case: D3Q19

- exchange of 18 distribution functions with 18 neighbors
- 30 complete planes
  - with the six “main” neighbors (N,S,E,W,T,B)
  - five distribution functions each
- with the other 12 neighbors a single line each

“Standard” Ghost Cell Approach

3D case: D3Q19

- exchange of 18 distribution functions with 18 neighbors
- 30 complete planes
  - with the six “main” neighbors (N,S,E,W,T,B)
  - five distribution functions each
- with the other 12 neighbors a single line each

Problems

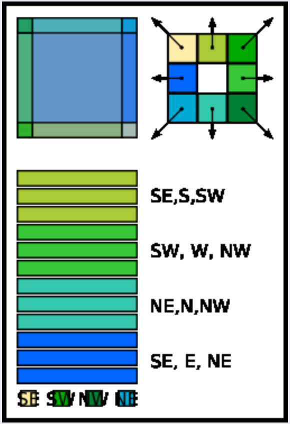
- full set of ghost cells nearly doubles the required memory in 3D (10^3 vs. 8^3)
- only few data of cache lines transferred is actually used
- scatter / gather of single values is ultra-annoying on SPUs
- lot of space wasted or badly aligned for SIMD
- parallelization will either need two versions of each block or complex memory organization to reuse data

Our Data Structure

- box' data structure contains
    - copies of outer cells to be exchanged
    - pointers to the neighbors' copies
- copies are re-ordered in SPU and stored contiguously with a single DMA transfer
- getting all necessary data from a neigboring box needs a single DMA transfer
- actually, two copy (halo) and two pointer structures (remote halo) are used

\- usage of "remote halo" data is tricky

2D example:



Parallelization

single node / interleaved memory

- data for all blocks necessary is allocated in one piece
- all SPU threads synchronize using atomic counting (libsync)

Parallelization

single node / interleaved memory

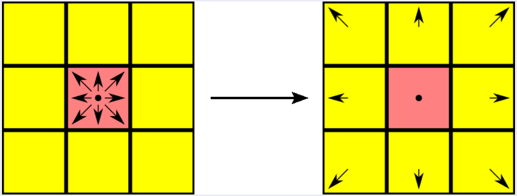
- data for all blocks necessary is allocated in one piece
- all SPU threads synchronize using atomic counting (libsync)

NUMA-aware

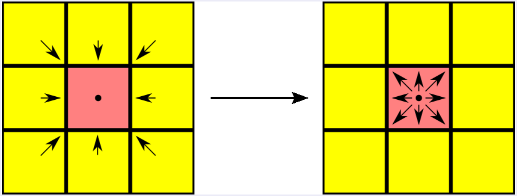
- on each node, about the same number of boxes is allocated
- SPUs on each node operate on local boxes and synchronize only locally
- remote memory access only necessary for “remote halo” transfers

Stream Step revisited

Push view

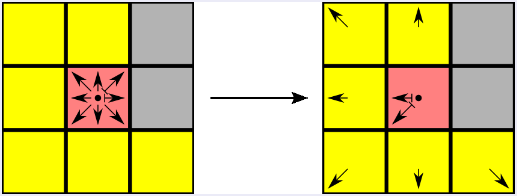


Pull view

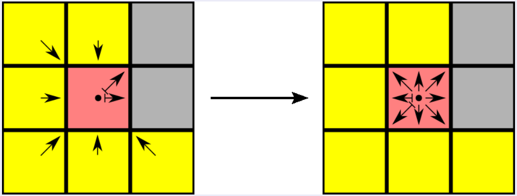


Bounce-Back – LBM No-Slip Boundary Condition

Push view



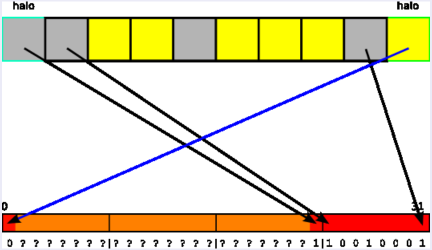
Pull view



Cell Types

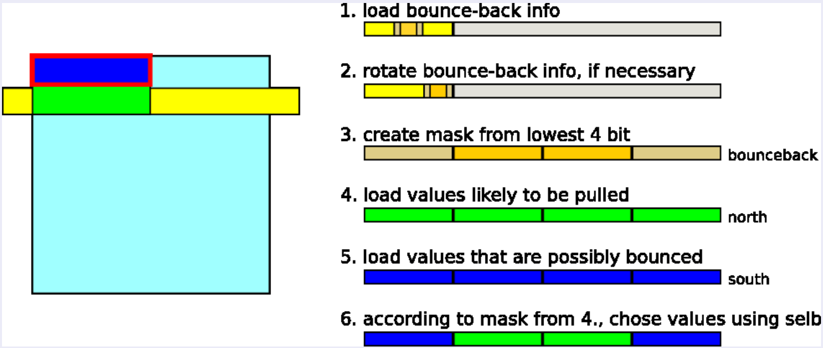
Cell type description

- $8 \times 8 \times 8$ array of 16-bit flag fields in each block
  - source (inflow), sink (outflow)
  - experimental outflow using pressure
- type halo and type remote halo structures analogous to distribution function halos
- as geometry is currently static, all $10 \times 10 \times 10$ bounce-backs information is cached
- 100 32-bit-fields per box, describing a single line each



SIMDized Bounce-Back

Example: Stream new north vector



1. load bounce-back info
2. rotate bounce-back info, if necessary
3. create mask from lowest 4 bit
4. load values likely to be pulled
5. load values that are possibly bounced
6. according to mask from 4., chose values using selb

SIMDized Bounce-Back

Remarks

- possibly bounced data is always aligned
- streaming must take halo data into account
    - complicated at boundaries
    - often shuffle operations to mix data from two SIMD vectors (with E or W)
- register blocking can enhance speed dramatically
    - streaming opposite directions together
    - reuse of bounce-back information
- $18 \times 8 \times 8 \times 8 = 9216$ "decisions" per block
    - scalar processing unbearably slow
    - SIMD implementations takes less than 2 clock cycles per decision
- streaming is most complicated part on SPU

Updating a Box on a SPU

Overview

1. fetch. . .
    - distribution functions
    - LBM-cell type information
    - remote halo pointer structure
2. fetch. . .
    - halo planes from neighbors (T, B, N, S, E, W)
    - halo lines from neighbors (TS, NE, BW . . .)
3. set source / sink
4. stream (and bounce) values (including values from halo planes)
5. correct pressure outflow
6. calculate collision
7. prepare new halo structure from calculated values
8. store box and halo data

celllbm Performance – MFLUPS, FLUPS, FLOPS and Bandwidth

Performance numbers

MFLUPS	number of lattices containing fluid updated per second
MLUPS	number of lattice updates per second lattice number = box number $\times 8^3$
FLOPS	the stream step usually has NO floating point operations; every collision needs 167 floating point operations (115 instructions due to 52 fused operations)
bandwidth	for every lattice update at least 152B, but usually 187B and more have to be transferred

2.4 GHz Rev B Cell Blade at Böblingen

single and dual node MLUPS performance

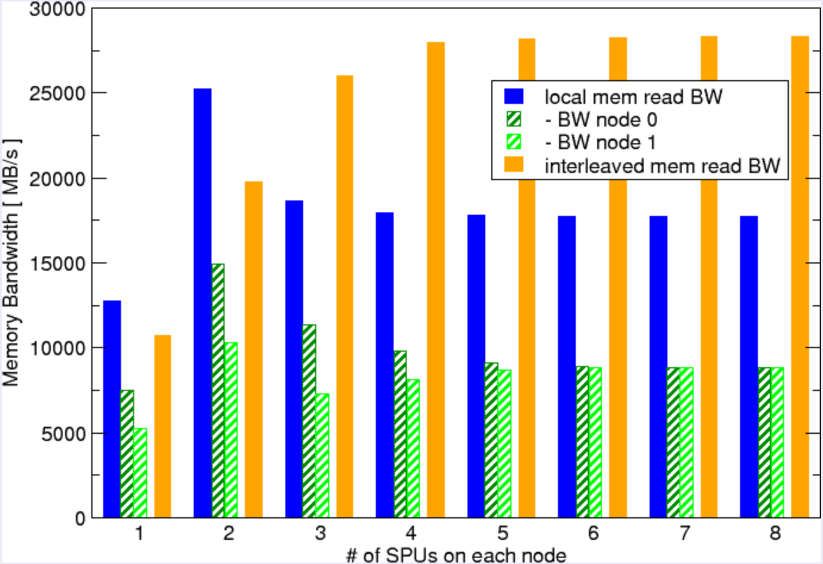
computation memory	node0 node0	parallel interleaved	parallel distributed
1 SPU/CPU	31	55	53
2 SPUs/CPU	61	89	98
3 SPUs/CPU	72	99	**88**
4 SPUs/CPU	72	104	**87**

96^3 canal flow

- strange behavior after newer kernel was installed (also using 4kB instead of 64kB pages)
- before that, maximum performance around 150 MLUPS for large domains

Problem on the Rev B Cell Blade

“Symmetric” read benchmark



PS3

“development system”

- 1 Cell Processor@3.2 GHz
- 6 SPUs available (1 disabled, 1 used from/as hypervisor)
- 256 MB Rambus XDR RAM
- 60 GB HD, gigabit ethernet
- only framebuffer graphics available
- currently running YDL 5

PS3

“development system”

- 1 Cell Processor@3.2 GHz
- 6 SPUs available (1 disabled, 1 used from/as hypervisor)
- 256 MB Rambus XDR RAM
- 60 GB HD, gigabit ethernet
- only framebuffer graphics available
- currently running YDL 5

Memory benchmarks on the Playstation

	bandwidth in GB/s		
page	read	write	mixed
4kB	19.7	19.4	17.6
16MB	22.6	22.6	20.3

6 SPUs doing 16k DMA transfers into a single 16MB buffer

cellbm on PS3

scale up (hard)

SPUs	MFLUPS	MLUPS
1	38	41
2	73	78
3	80	85
4	80	85
5	80	85

96^3 canal flow

cellbm on PS3

scale up (hard)

SPUs	MFLUPS	MLUPS
1	38	41
2	73	78
3	80	85
4	80	85
5	80	85

96^3 canal flow

4kB pages vs. huge pages (16MB)

	MFLUPS	MLUPS
4kB	65	72
16MB	71	81

48^3 canal flow on 4 SPUs

Outlook

More to do...

- examine and improve performance especially on Cell blade
  - influence of page size
  - kernel flaws?
  - huge pages
- MPI parallelization
  - overlapping exchange of Halo data
- examine influence of rounding mode

Outlook

More to do. . .

- examine and improve performance especially on Cell blade
  - influence of page size
  - kernel flaws?
  - huge pages
- MPI parallelization
  - overlapping exchange of Halo data
- examine influence of rounding mode

Acknowledgements

- cellbm is based on Jan Götz' Master Thesis
- also lots of graphics have been shamelessly copied from him
- thanks to IBM development center at Böblingen for access to Cell Blades

The End

Thank you very much for your attention!