

# Optimizing Fluid Simulation and other scientific applications on the Cell

Markus Stürmer

University Erlangen-Nuremberg – System Simulation

June 14th 2007 @ SFB 716 Stuttgart



# Thinking about performance

## Intel 8086



introduced 1978  
up to 10 MHz  
0.75 MIPS@10MHz

## Intel Core 2 Duo



introduced 2006  
up to 3000 MHz  
22305 MIPS@1830 MHz

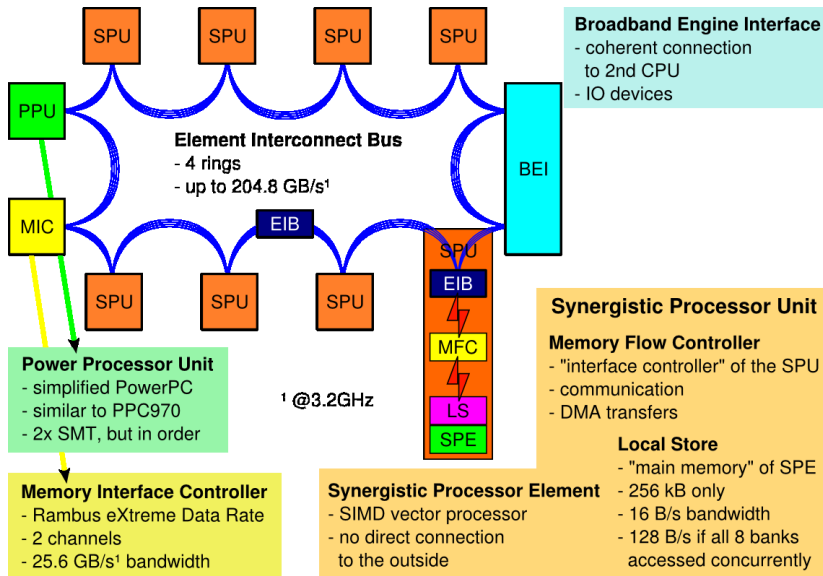
Chip design is as important as the manufacturing technology!

wider buses • caches • pipelining • superscalarity  
wider execution units • hardware prefetcher  
homogenous and **hybrid multicores**

# Outline

- ① Introduction to the Cell processor
- ② Blood-flow simulation in aneurysms
  - ▶ the Lattice Boltzmann Method
  - ▶ memory layout and optimization
  - ▶ performance
- ③ Bandwidths and latencies
  - ▶ data transfer
  - ▶ communication
- ④ Outlook & conclusions

# Cell processor overview



# The Synergistic Processor Element

## Synergistic Processor Unit + Memory Flow Controller

### Synergistic Processor Unit

- very small computer
  - 128 128-bit all-purpose registers
  - all data operations are SIMD (single instruction multiple data)
- *so one scalar operation is usually more expensive than a full SIMD one*
- 25.6 GFlops (single-precision fused-multiply-add) each
  - direct access only to 256 kB of *local store*, but not to main memory
  - only 16 byte aligned 16 byte accesses to the local store
  - branch prediction only in software, 20 cycles branch miss penalty
  - no privileged software or direct system calls

### Memory Flow Controller

...

# The Synergistic Processor Element

## Synergistic Processor Unit

...

## Memory Flow Controller

- communication interface (to the PPE and other SPEs)
  - ▶ mailboxes and signal notification
  - ▶ performs memory mapping of local store and register file
  - ▶ utilized by PPU to load programs to and control SPE
- asynchronous data transfers (DMA)
  - ▶ local store  $\leftrightarrow$  main memory, other local stores or devices
  - ▶ 16 DMAs in-flight
  - ▶ list transfers possible (gather / scatter)
  - ▶ only naturally aligned transfers of 1, 2, 4, 8,  $16 \cdot n$  bytes up to 16kB
  - ▶ optimally transfers of  $128 \cdot n$  bytes, 128-byte-aligned
  - ▶ need to interleave transfers, typically double- or multibuffering

*MFC is connection between its SPE and the rest*

# Example applicaton: Simulation of blood-flow in aneurysms

## Problem

- dilatation (local ballooning) 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

# Example applicaton: Simulation of blood-flow in aneurysms

## Problem

- dilatation (local ballooning) 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

## Goals

- understanding the development of aneurysms
- support for planning of therapy



# Example applicaton: Simulation of blood-flow in aneurysms

## Problem

- dilatation (local ballooning) 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

## Goals

- understanding the development of aneurysms
- support for planning of therapy

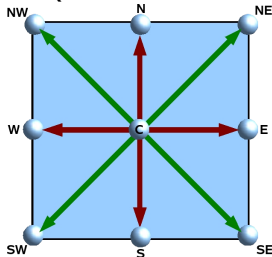
## Challanges

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

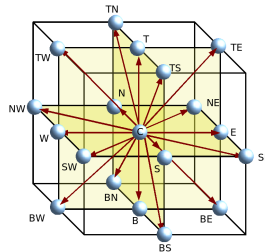
# Lattice Boltzmann Method

- 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

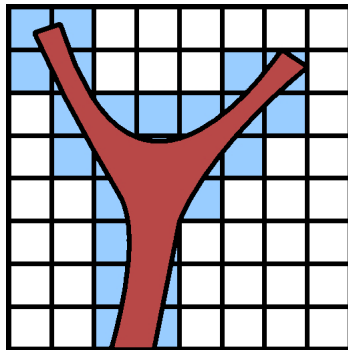
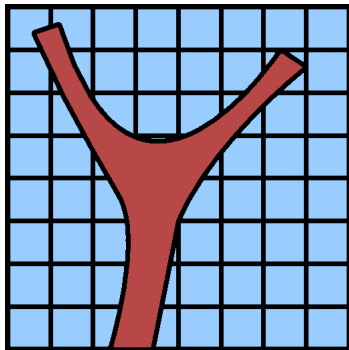


D3Q19



# Domain splitting

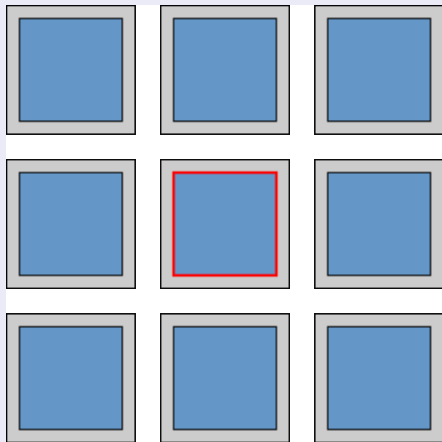
Divide whole domain into equally sized blocks ( $8 \times 8 \times 8$ ) and only allocate and calculate blocks including fluid cells.



Every resulting block is small enough that it can be processed in local store.

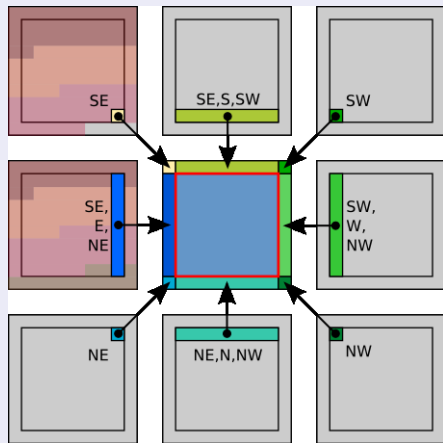
# “Standard” ghost cell approach

## 2D $8 \times 8$ example



# “Standard” ghost cell approach

## 2D $8 \times 8$ example



### Problems:

- memory requirements (nearly doubles in 3D)
- only few data of cache lines transferred is actually used
- scatter / gather of single values is “ultra”-annoying on SPUs
- badly aligned for SIMD
- complex or memory consuming parallelization

# Our data structure for one box

- **Box itself**

- 2D  $8 \times 8 \times 9$  distribution functions

- 3D  $8 \times 8 \times 8 \times 19$  distribution functions

- **Halo**

- ▶ copies of values that need to be exchanged
  - ▶ one copy for odd and for even timesteps
  - ▶ reordered on SPU and sequentially DMAed

- 2D 12 lines and 4 single lattices

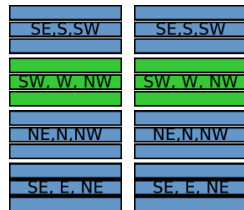
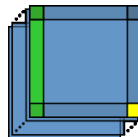
- 3D 30 planes and 12 lines

- streaming becomes quite tricky

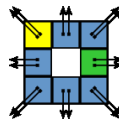
- **Remote Halo**

- ▶ pointers to the neighbors' copies

- similar data for lattice type information

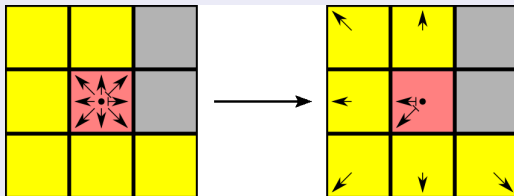


SE SW NW NE SE SW NW NE

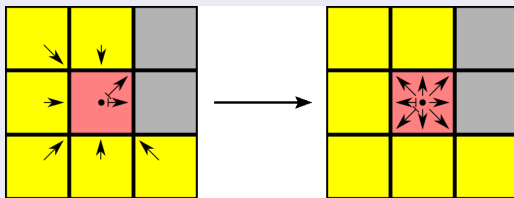


# Streaming and bounce-back

## Push view

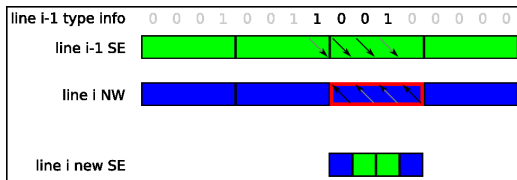


## Pull view



# SIMDized bounce-back

## Example: Calculate new SE distribution functions



1. load values likely to be streamed (unaligned)



2. extract desired vector



3. load values possibly to be bounced



4. load solid info bitfield



5. rotate accordingly



6. create mask from 4 bits



7. use mask to select streaming and bouncing values





# 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

# cellbm 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

## 3.2 GHz Cell Blade at Jülich

### single and dual node MLUPS performance

computation memory	node0 node0	parallel interleaved	parallel distributed
1 SPU/CPU	40	67	64
2 SPU <sub>s</sub> /CPU	78	111	121
3 SPU <sub>s</sub> /CPU	97	130	143
4 SPU <sub>s</sub> /CPU	98	136	164

96<sup>3</sup> canal flow

$$MFLUPS = MLUPS \cdot \frac{94^3}{96^3}$$

## “development system”

- 1 Cell Processor@3.2 GHz
- 6 SPU's 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, added SDK2.0 and updated libspe2

## “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, added SDK2.0 and updated libspe2

## Memory benchmarks on the Playstation

page	bandwidth in GiB/s		
	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	MLUPS
1	41
2	78
3	85
4	85
5	85
96 <sup>3</sup> canal flow	

# cellbm on PS3

## scale up (hard)

SPUs	MLUPS
1	41
2	78
3	85
4	85
5	85
96 <sup>3</sup> canal flow	

## 4kB pages vs. huge pages (16MB)

	MLUPS
4kB	72
16MB	81
48 <sup>3</sup> canal flow on 4 SPUs	



# Bandwidth on Juice node

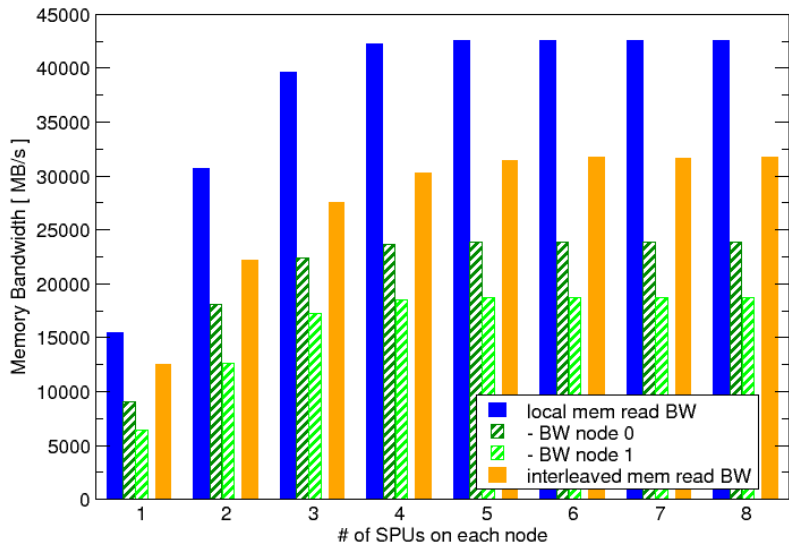
## Main Memory Bandwidth

<i>SPU node</i> <i>mem node</i>	0 0	0 1	both interleaved	both local
1 SPU read	12.0 (11.2)	7.0 (6.6)	15.6 (14.6)	19.2 (17.9)
1 SPU write	12.6 (11.8)	11.9 (11.1)	19.1 (17.8)	20.1 (18.7)
best read	25.2 (23.4) <sub>4</sub>	14.4 (13.4) <sub>3</sub>	36.0 (33.5) <sub>7</sub>	50.2 (46.7) <sub>4</sub>
best write	24.8 (23.1) <sub>3</sub>	14.3 (13.3) <sub>3</sub>	34.5 (46.7) <sub>6</sub>	48.8 (45.4) <sub>4</sub>

Bandwidths are GB/s (GiB/s).

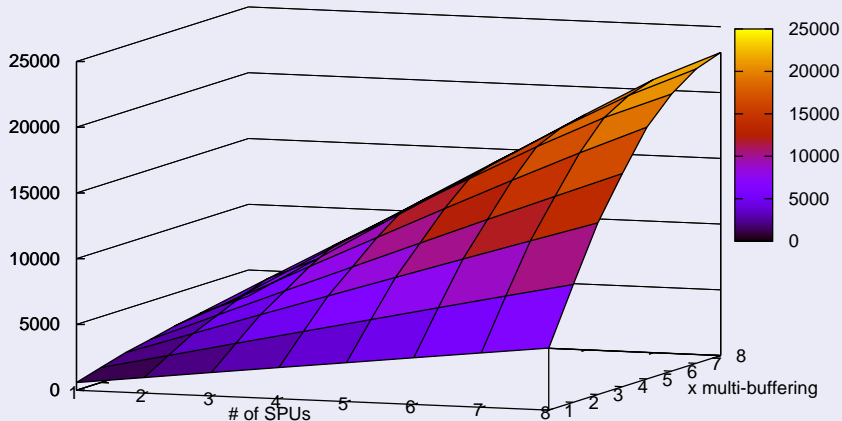
# Bandwidth on Juice node

## "Symmetric" read benchmark

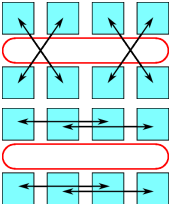
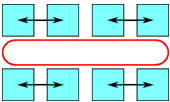
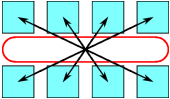


# Bandwidth on Juice node

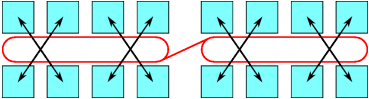
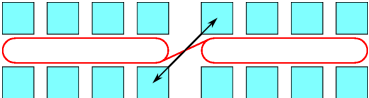
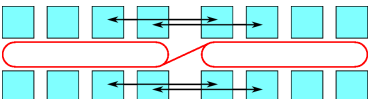
## 128B reads



# Memory bandwidth between SPUs

Scenario	read BW		write BW	
	cumulative	per SPU	cumulative	per SPU
	204.8 (190.7)	25.6 (23.8)	204.8 (190.7)	25.6 (23.8)
	188.6 (175.7)	~23.5 (22.0)	189.2 (176.2)	~23.6 (22.1)
	82.5 (76.8)	~10.3 (9.6)	82.5 (76.8)	~10.3 (9.6)

# Memory bandwidth between SPUs

	read BW	write BW
	409.5 (381.4)	409.5 (381.4)
	10.3 (9.6) 6.1 vs 4.2	9.5 (9.1) 6.1 vs 3.6
	21.2 (19.7) 10.91 vs 10.25	17.9 (16.7) 9.16 vs 8.78

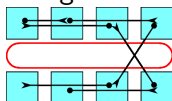
# Memory bandwidth between SPUs

- Naive Chain:



~13.0 (12.1) BW per SPU (get and put), 91.0 (84.7) cumulative

- Using all four rings:



25.6 (23.9) BW per SPU (get and put), 179.0 (166.9) cumulative

- a single SPU cannot get more than 4.4 (4.1) over the interconnect

# Signal latency between SPUs

s/r	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		424	408	420	412	424	416	428	1848	1852	1838	1842	1832	1836	1830	1834
1	424		420	408	424	412	428	416	1852	1856	1842	1847	1836	1840	1834	1838
2	408	420		416	400	420	404	416	1837	1840	1827	1831	1823	1824	1820	1824
3	420	408	416		420	400	416	404	1840	1846	1831	1835	1824	1827	1824	1826
4	412	424	400	420		416	400	412	1831	1836	1822	1826	1816	1821	1814	1818
5	424	412	420	400	416		412	400	1835	1840	1826	1830	1820	1824	1818	1823
6	416	428	404	416	400	412		408	1826	1832	1819	1824	1812	1817	1811	1815
7	428	416	416	404	412	400	408		1833	1837	1824	1826	1817	1823	1815	1819
8	1848	1852	1837	1840	1831	1836	1826	1832		424	404	420	412	424	416	428
9	1851	1856	1840	1846	1836	1840	1832	1837	424		420	408	424	412	428	416
10	1838	1842	1827	1831	1822	1826	1819	1824	408	420		416	400	420	404	416
11	1842	1847	1831	1835	1826	1830	1824	1826	420	408	416		420	400	416	404
12	1832	1836	1823	1824	1816	1821	1812	1817	412	424	400	420		416	400	412
13	1836	1840	1824	1827	1821	1824	1817	1823	424	412	420	400	416		412	400
14	1830	1834	1819	1824	1814	1818	1811	1815	416	428	404	416	400	412		408
15	1834	1838	1824	1826	1818	1823	1815	1819	428	416	416	404	412	400	408	

- times include setup of message and response
- signal ping pong on same CPU: 400–428 clock cycles
- signal ping pong between CPU: 1811–1856 clock cycles
- 125–580 ns vs. 4.3  $\mu$ s for ping pong on our Opteron Infiniband connect

# Mailbox latency between SPUs

## “correct” ping pong

	PPU on 0	PPU on 1
SPU on 0	596–611	2364–2379
SPU on 1	2729–2744	495–611

PPU checks availability of message first, receives then.

## “fast” ping pong

	PPU on 0	PPU on 1
SPU on 0	358–366	1419–1438
SPU on 1	1772–1778	358–366

PPU polls until it gets a result not equal to “0”, so it does not detect “0” messages.



# Outlook & conclusions

## More to do...

- examine influence of rounding mode
- MPI parallelization

# Outlook & conclusions

## More to do...

- examine influence of rounding mode
  - MPI parallelization
- 
- good potential
  - memory alignment and SIMD vectorization are very important
  - there's no performance without parallelization
  - software support gets better, but good knowledge of the architecture still necessary

# The End

## 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 Blade
- thanks to ZAM / FZ Jülich for access to JUICE

Thank you very much for your attention!