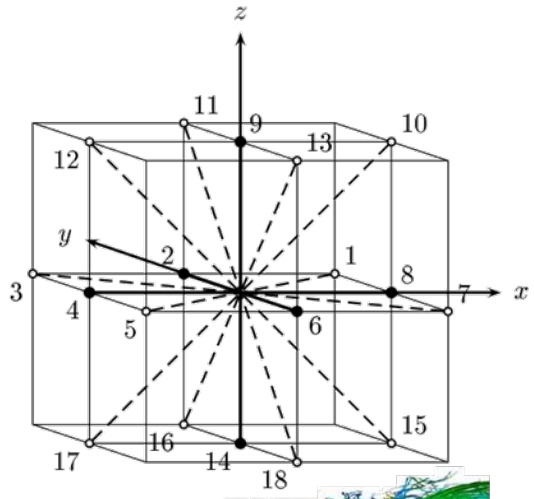
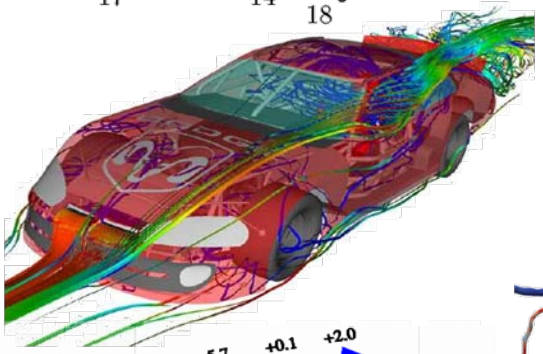
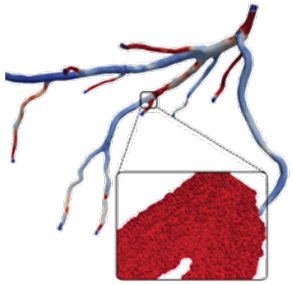
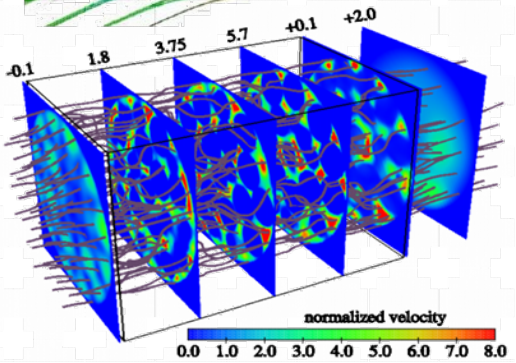
Lattice Boltzmann for CFD and beyond









The lattice Boltzmann method:

- **roots in statistical physics**
  - →velocity discrete Boltzmann equation
- **used to solve incompressible fluid flows**
- **also used beyond classical CFD: e.g. MHD, multiphase, …, civil engineering, computational steering**
- **iterative stencil scheme with explicit time-step**
  - →vector data (e.g. D3Q19) – no reuse of data as in simple Jacobi-type schemes
  - →low computational intensity; high memory intensity
  - →only next neighbor communication
  - →weak scaling drama: number of time steps scales with resolution

Figures by courtesy of EXA GmbH, Thomas Zeiser; Succi et al.

KONWIHR II

Federal Ministry of Education and Research

through grant SKALB (01IH08003)

SKALB

Performance Engineering – Our approach

1. **Analyze** the minimum computational requirements (data volume, FLOP-ops) of the algorithm

2. **Analyze** the computational requirements (data access in cache/main memory, FLOPS, instruction mix,..) of the implementation. Optimize if they do not fit to data from 1.

3. **Analyze** the available computational resources of the target hardware: Cache/Memory bandwidth, SIMD capabilities,..

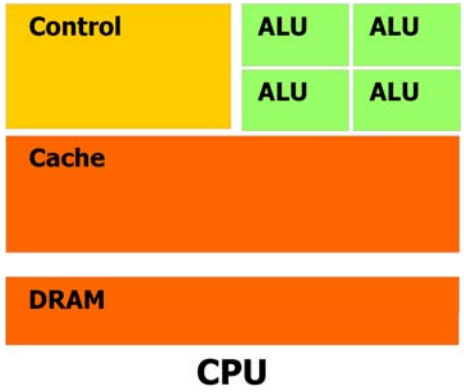
4. **Determine** max. performance (min. runtime) based on 2 and 3.

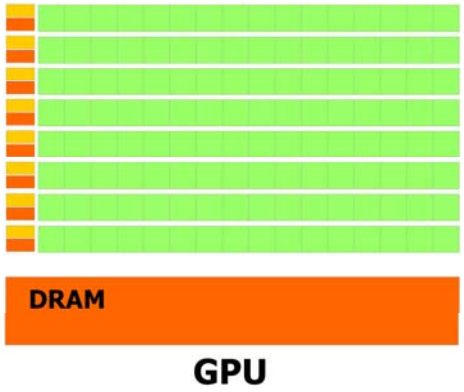
5. **Measure performance** and compare with 4. Go back to 2. / 3. if numbers differ substantially

Performance Engineering – Hardware capabilities

GPU vs. CPU light speed estimate:

1. **Compute bound: 4-5 X**
2. **Memory Bandwidth: 2-4 X**





	Intel Core i5 – 2500 ("Sandy Bridge")	Intel X5650 DP node ("Westmere")	NVIDIA C2070 ("Fermi")
Cores@Clock	4 @ 3.3 GHz	2 x 6 @ 2.66 GHz	448 @ 1.1 GHz
Performance⁺/core	52.8 GFlop/s	21.3 GFlop/s	2.2 GFlop/s
Threads@stream	4	12	8000 +
Total performance⁺	210 GFlop/s	255 GFlop/s	1,000 GFlop/s
Stream BW	17 GB/s	41 GB/s	90 GB/s (ECC=1)
Transistors / TDP	1 Billion* / 95 W	2 x (1.17 Billion / 95 W)	3 Billion / 238 W

⁺ *Single Precision*

* *Includes on-chip GPU and PCI-Express*

Complete compute device

Lattice Boltzmann method
Analysis of prototype implementation

```
double precision F(0:iMax+1,0:jMax+1,0:kMax+1, 0:18,0:1)
do k=1,kMax
   do j=1,jMax
      do i=1,iMax
         if( fluidcell(i,j,k) ) then
            LOAD F( i  ,j  ,k  , 0,t)
            LOAD F( i+1,j+1,k  , 1,t)
            …
            LOAD F( i  ,j-1,k-1 ,18,t)
            Relaxation (complex computations)
            STORE F(i, j, k, 0, t+1)
            STORE F(i, j, k, 1, t+1)
            …
            STORE F(i, j, k,18, t+1)
         endif
      enddo
   enddo
enddo
```

Data layout
F(I , J , K, Q)

Collide Step

~200 FLOPs / Update

Stream Step

38 cache lines (~2.5 KB) must be held in cache

If cache line of store operation is not in cache it must be loaded first ("write allocate") – avoid them by NT stores

#loads from main memory: (19 + 19) * 8Byte

#store to main memory: 19 * 8Byte

456 [304] Byte /Update

Lattice Boltzmann method

Analysis of prototype implementation

Our baseline version contains all basic optimizations (fuse-stream collide; work reduction,…) which are still ignored by many people..

- **F(Q,I,J,K)** Bad, but still widely used data layout
- **F(I,J,K,Q)** Data layout with min. main memory transfers
- **SPLIT** Split up inner most loop into several loops
- **SIMD** SIMD intriniscs kernel
- **NT stores** SIMD NT store intriniscs writing result to main memory bypassing cache → 304 Byte/Update

Lattice Boltzmann method
Performance model (1)

- **Performance measure: Million Fluid Lattice cell Updates Per second**

$$\textbf{MFLUP/s} = \frac{\text{sweeps}*\text{iMax}*\text{jMax}*\text{kMax}}{10^6 * \text{Time}_{\text{sweeps}}}$$

Wallclock time to perform *sweeps* LBM iterations

- **Roofline model**
  - Assumption: **Arithmetic (FP) or main memory bandwidth (BW)** limits application performance
  - Determine max. LBM performance for given floating point performance and for main memory bandwidth separately
  - Minimum of both performance numbers limits LBM performance

Lattice Boltzmann method
Performance model (2)

- **Arithmetic limit (FP):**
  - A good implementation of a simple LBM step requires approx. 200 FLOP

Performance estimate (FP): $$\frac{\text{FP\_PeakPerformance}}{200\ \text{FLOP/FLUP}}$$

  - "FP_PeakPerformance": Which one? DGEMM, arithmetic mix, SSE/AVX,…

- **Memory bandwidth limit (BW):**
  - Determine attainable memory bandwidth: **Mem_BW [MByte/s]** (e.g. stream benchmark)

Performance estimate (BW): $$\frac{\text{Mem\_BW}}{456\ [304]\ \text{Byte/FLUP}}$$

  - 19 Concurrent READ and 1 WRITE streams (STREAM: 1 READ; 1 STORE) Ignoring intra cache data transfers
  - Perfect prefetching and associativity conflicts assumed

Lattice Boltzmann method
Performance model (3)

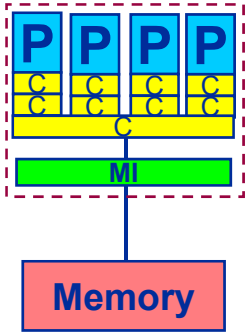
- **Single Intel Sandy Bridge (SNB) CPU (4-cores; 3,2 GHz):**
  - Mem_BW = 17,000 MByte/s (stream copy)
  - PeakPerformance = 105 GFLOP/s (dp) [210 GFLOP/s (sp)]

	Memory bandwidth (BW)		FP performance	
	Basic	**NT stores**	**Peak**	**scalar ADD**
Double precision	**37 MFLUP/s**	**56 MFLUP/s**	**500 MFLUP/s**	**62 MFLUP/s**
Single precision	**74 MFLUP/s**	**112 MFLUP/s**	**1,000 MFLUPs**	**62 MFLUP/s**

  - AVX SIMD instructions are a must at least for SP kernels for SNB!
  - Performance estimates are upper qualitative boundaries
  - Single socket numbers, i.e. 4-cores

Lattice Boltzmann method
Prototype performance (DP): Latest Intel desktop CPU

Double precision (DP): **Lid driven cavity (230^3)**

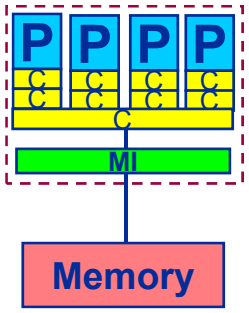


Intel Core i5 – 2500 (“Sandy Bridge”)

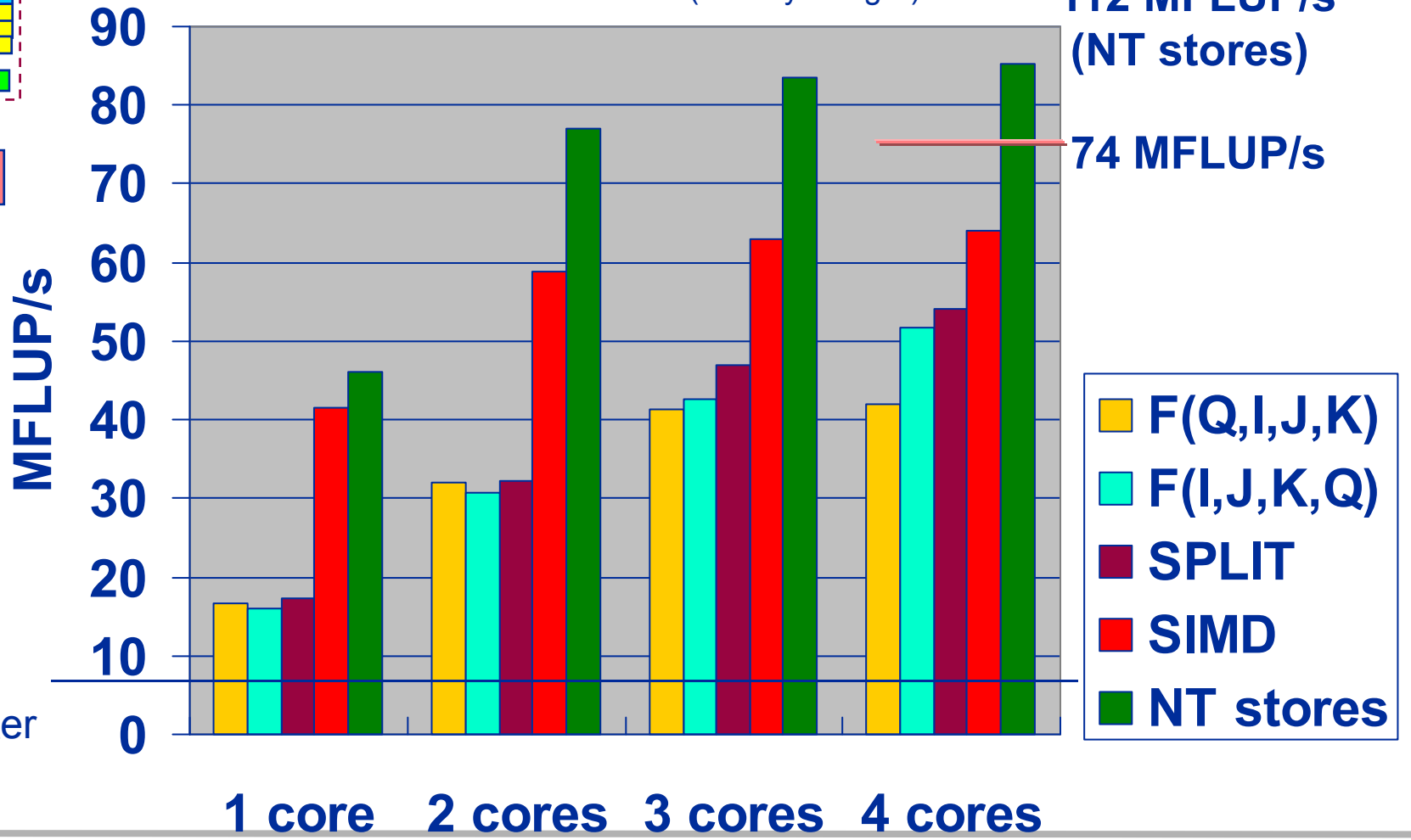
2nd year CE master student

Lattice Boltzmann method
Prototype performance (DP): Latest Intel desktop CPU

Single precision (SP): Lid driven cavity (230^3)



Intel Core i5 – 2500 (“Sandy Bridge”)



Performance model

112 MFLUP/s (NT stores)

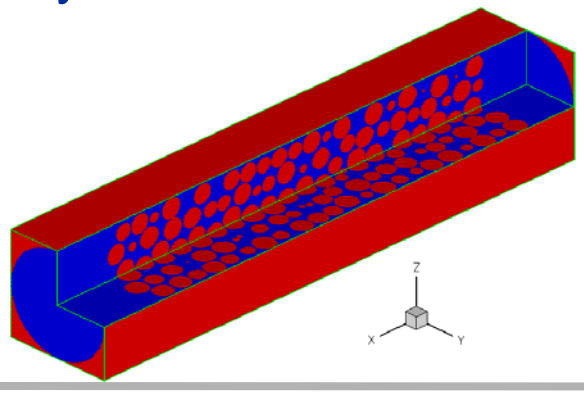
74 MFLUP/s

2nd year CE master student

From kernels to full applications

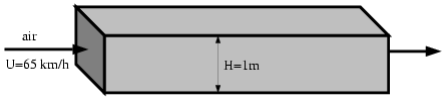
- **waLBerla: Widely applicable LB solver from Erlangen (Uli Rüde's group) uses "prototype" kernel**
  - "Patch-based" approach
  - Large C++ framework with highly optimized C/FORTRAN/SIMD kernels

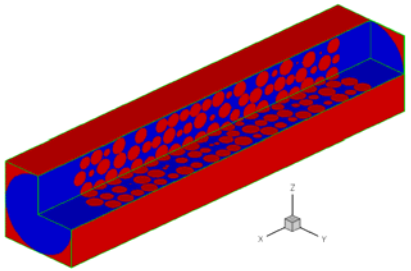
- **What about complex geometries? ("The tough boys play")**
  → ILBDC:
  - Sparse data structure: store fluid cells + adjacency list
  - Indirect addressing
  - NT stores can be used
    but **"AA pattern"** approach is more efficient

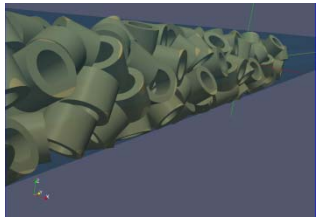


Lattice Boltzmann solver for complex geometries
Close to optimal parallel performance

Compute node performance for different geometries







Questions

- **Who knows the theoretically attainable performance of their most important application on their standard production machine?**
- **Who is using this process in code development / code optimization?**
- **Who has sufficient insight into computer architecture to go beyond simple main memory bandwidth models?**