Introducing a Cache-Oblivious Blocking Approach for the Lattice Boltzmann Method

G. Wellein, T. Zeiser, G. Hager
HPC Services
Regional Computing Center

A. Nitsure, K. Iglberger, U. Rüde
Chair for System Simulation
Department of Computer Science

Friedrich-Alexander-University Erlangen-Nuremberg

International Conference for Mesoscopic Methods in Engineering and Science 2006

Survey

- **The party is over –**
  Trends in High Performance Computing

- **Implementing iterative LBM –**
  achieving spatial locality

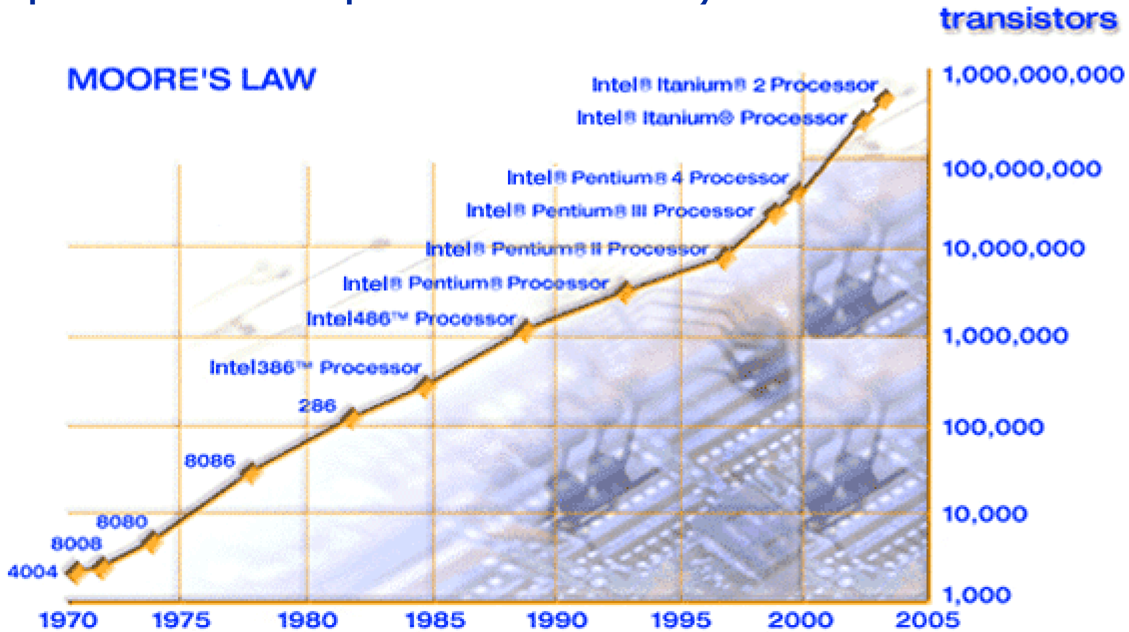
- **Cache-Oblivious Blocking Approach for the LBM –**
  improving temporal locality

- **Summary**

The party is over
Ever growing processor speed & Moore´s Law

- **1965 G. Moore (co-founder of Intel) claimed**
  #transistors on processor chip doubles every 12-24 months

Intel Corp.

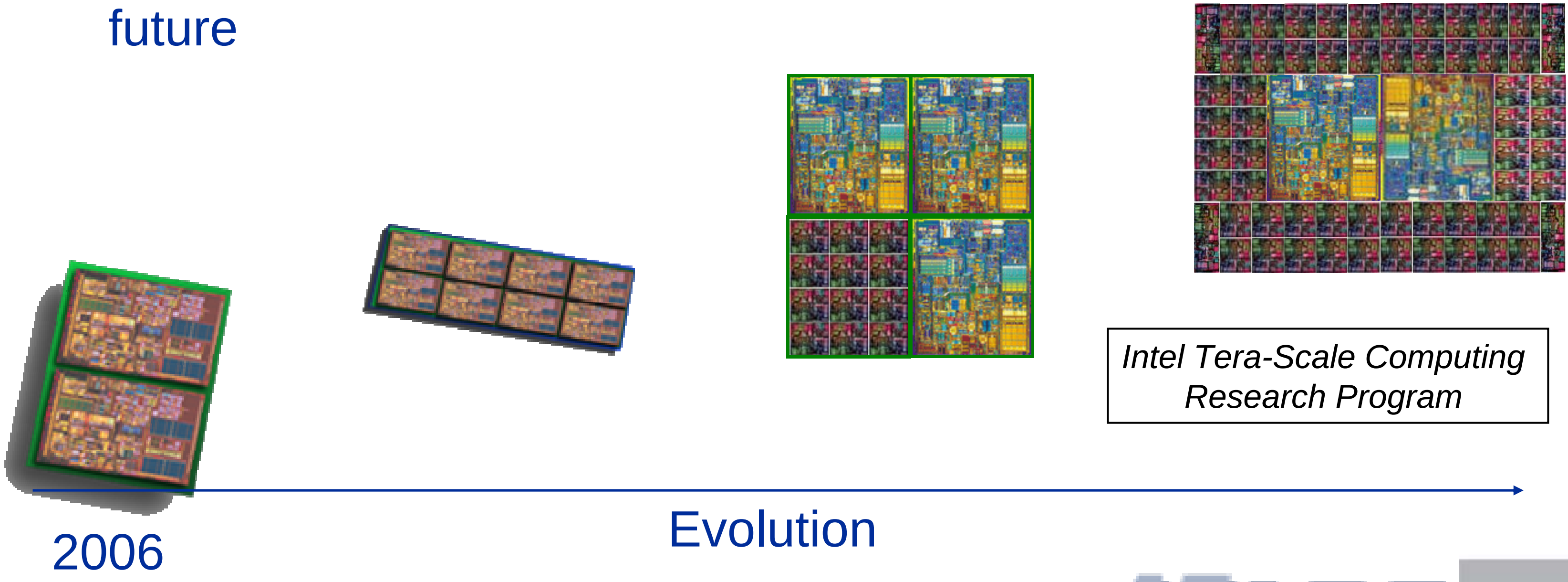


- **Processor speed grew roughly at the same rate**
  My computer: 350 MHz (1998) – 3,000 MHz (2004)
  Growth rate: 43 % p.y. -> doubles every 24 months

- **Does this trend continue?**

Multi-Core Processors – The party is over…

- Problem: Moore's law is still valid but increasing clock speed hits a technical wall (heat)
- Solution: Reduce clock speed of processor but put 2 (or more) processors (cores) on a single silicon die
- We will have to use many less powerful processors in the future

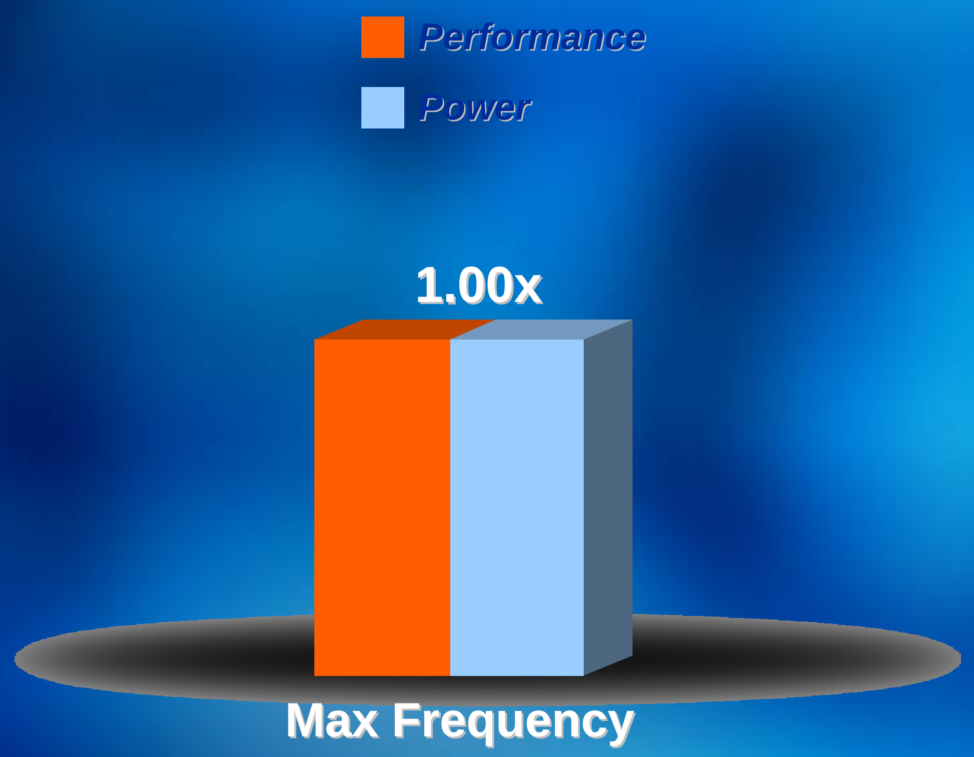


Intel Tera-Scale Computing Research Program

The party is over
Why will Multi-Core succeed?

By courtesy of D. Vrsalovic, Intel

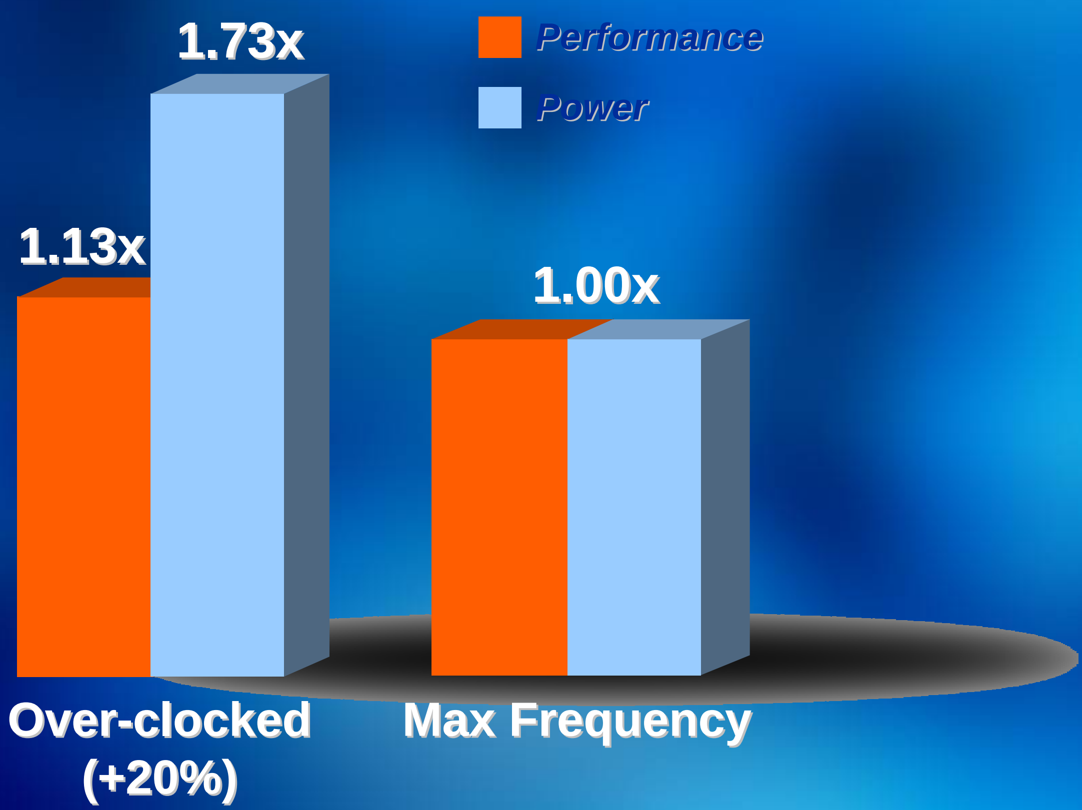
Power ~ Frequency ^{3}



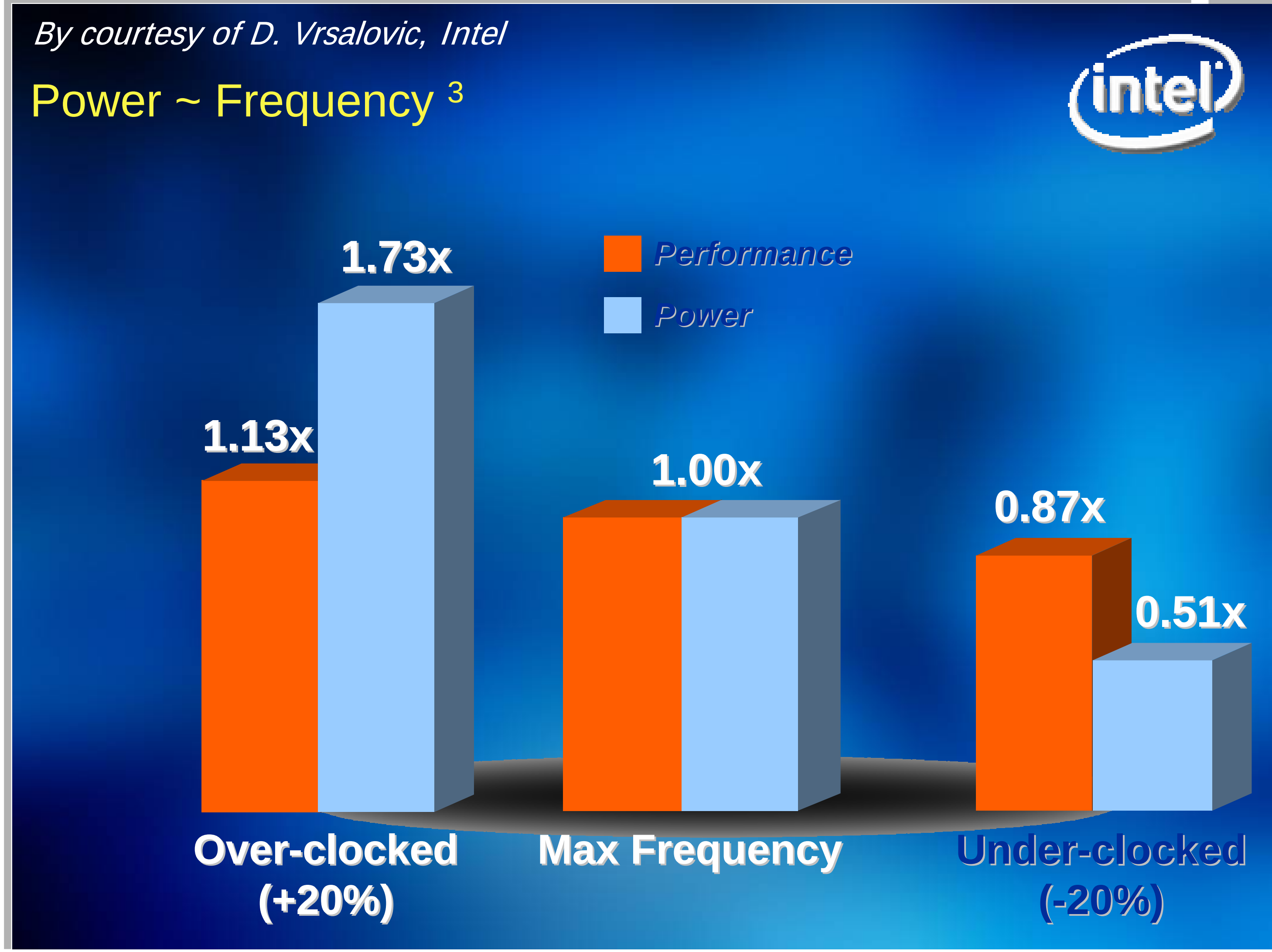
The party is over
Why will Multi-Core succeed?

By courtesy of D. Vrsalovic, Intel

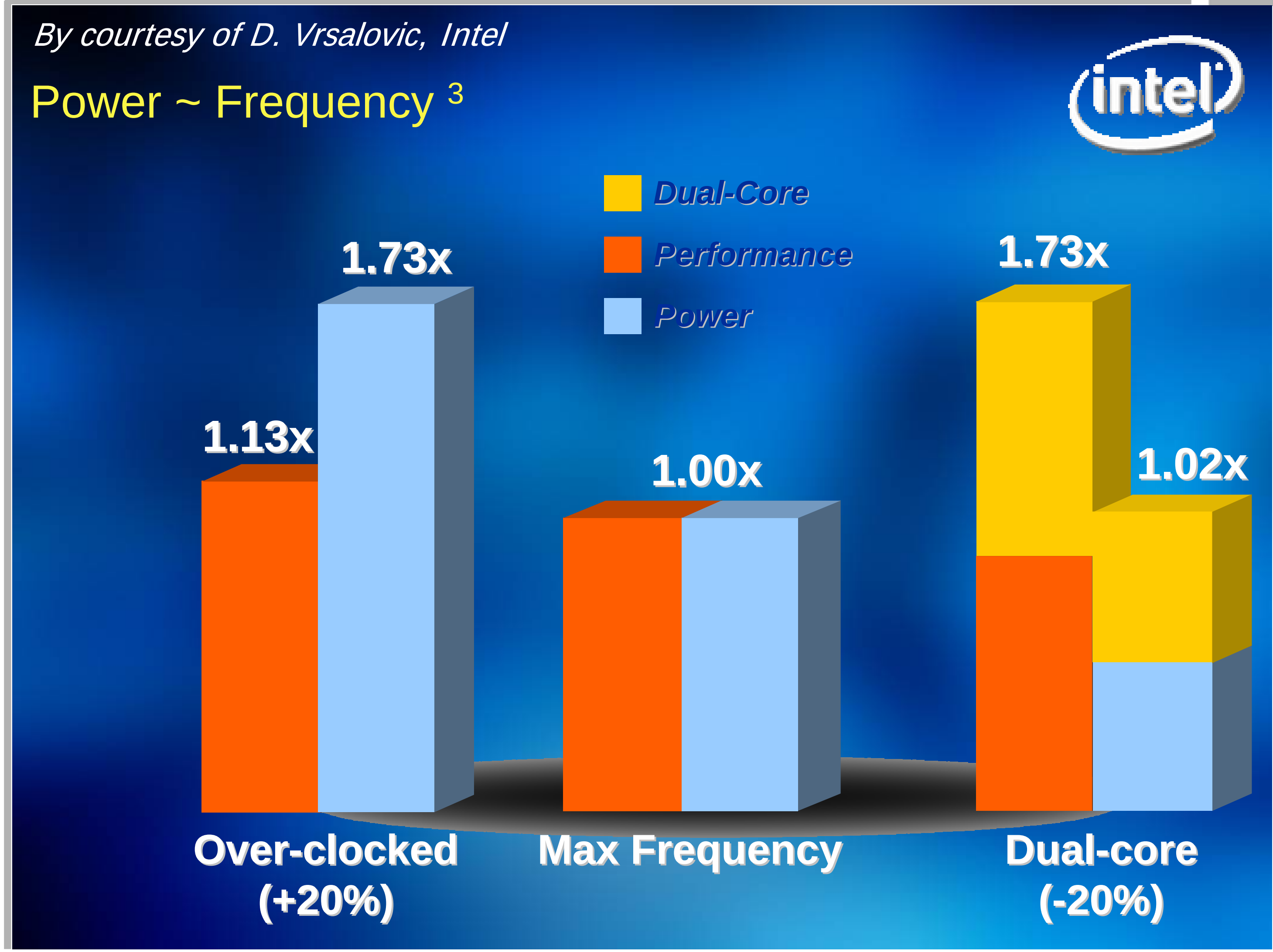
Power ~ Frequency ^{3}



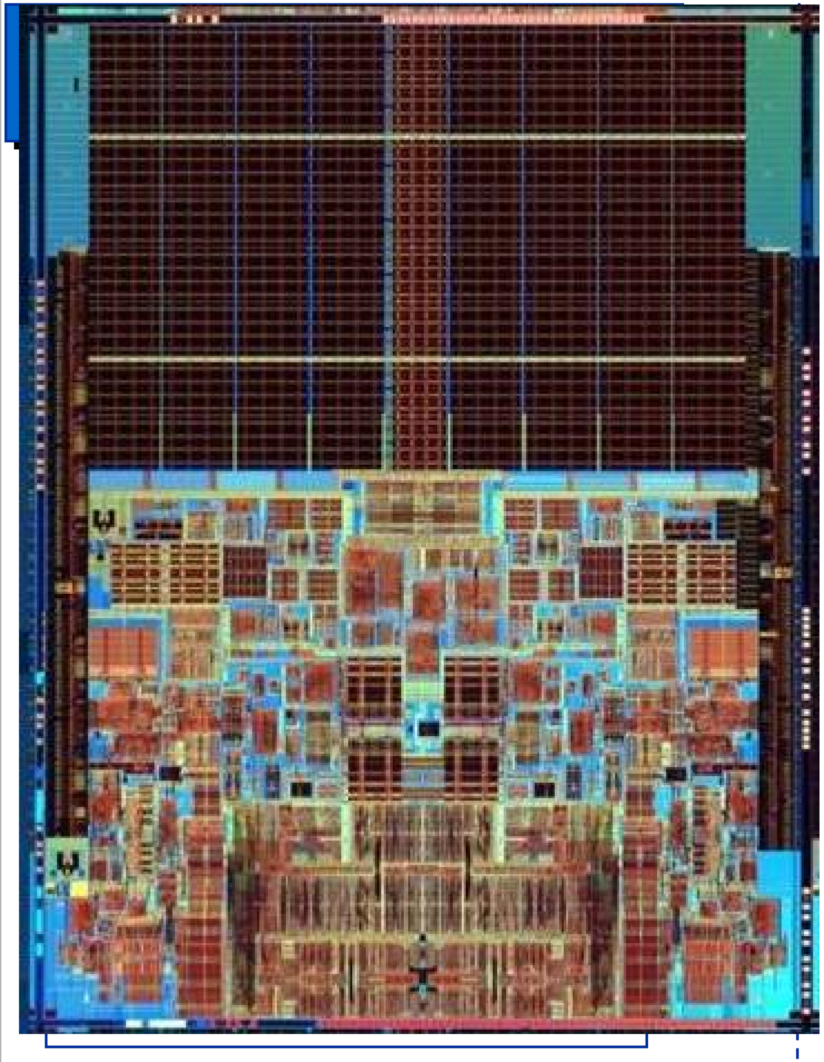
The party is over
Why will Multi-Core succeed?

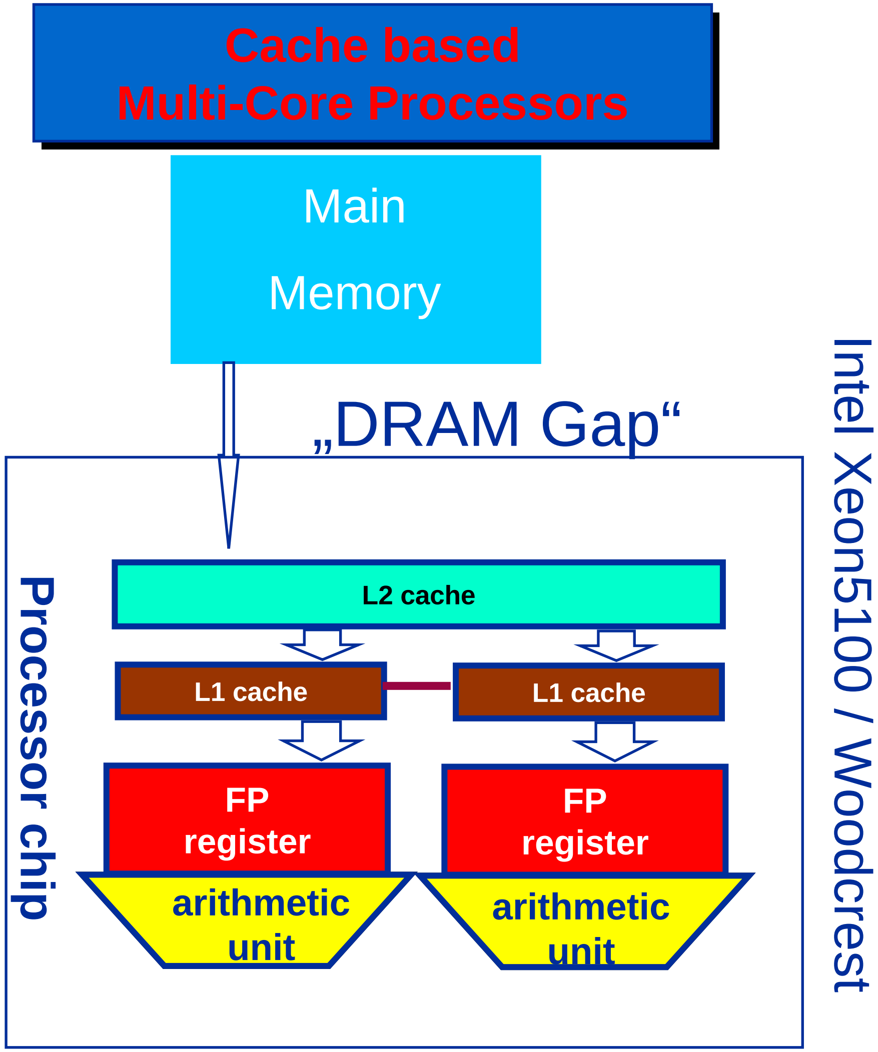


The party is over
Why will Multi-Core succeed?



Cache-Oblivious Blocking Approach
Multi-Core Memory hierarchies





- **Multi-Core Processors – The party is over…**
  - Single core performance will remain constant or decrease in the future
  - Several core on a silicon-die will share resources, e.g. caches
  - Main memory bandwidth will not scale with the number of cores
  - Heterogeneous cores on a silicon-die (see IBM Cell)

Lessons to be learned:

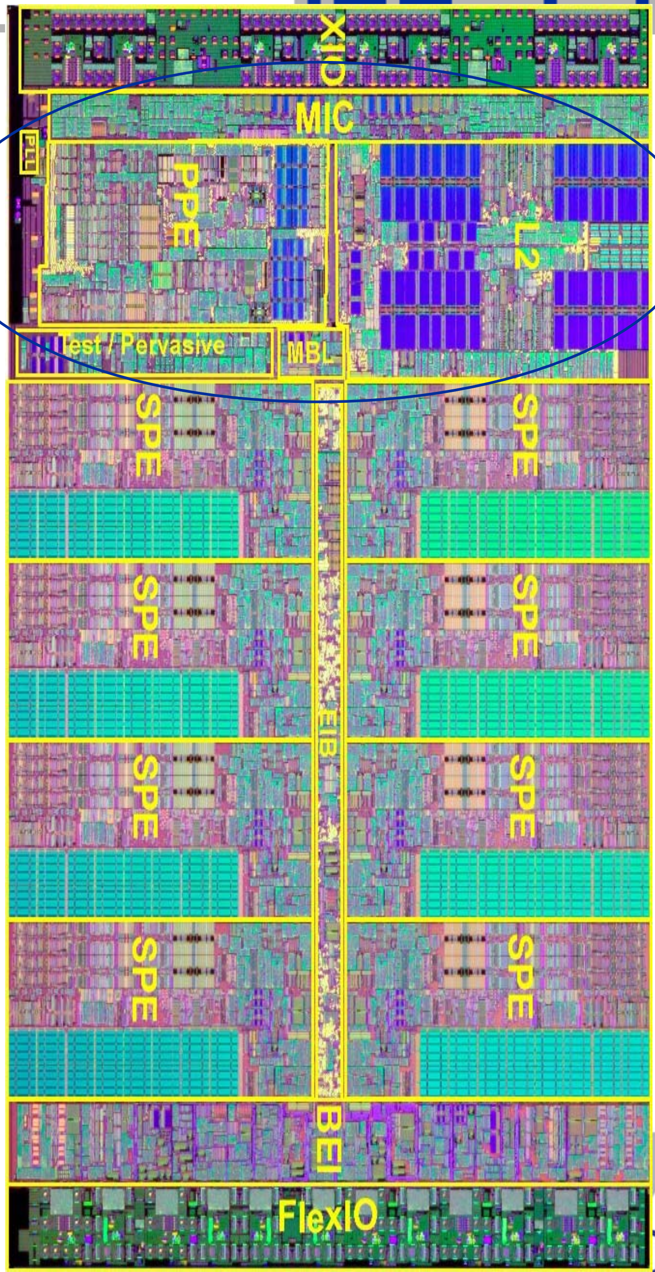
**Reduce bandwidth requirements
(Cache blocking to improve spatial and temporal locality)**

Parallelization will be mandatory for most applications in the future

Hybrid programming approaches for large scale simulations?!

The party is over
Other directions

- IBM Cell processor
  - To be used in Sony Playstation3
  - 221mm^2 die size - 234 million transistors
  - Eight synergistic processor elements (SPE) plus Power processor
  - Clock speed ~ 4 GHz
  - Peak performance (single precision) ~ 256 GFlops
  - Peak performance (double precision) ~ 26 GFlops
  - **Roundoff = Cutoff**
  - **Programming Model?**



- Acceleration Cards
  - Clearspeed acceleration board: 50 GFlop/s DGEMM at 25 Watt
  - *Memory bandwidth*
  - *Use of highly optimized offload libraries*
  - *Built for special purpose (e.g. long range MD simulation)*
- Field Programmable Gate Arrays (FPGAs)
  - “Configurable Processor” (at moderate speed 200-500 MHz)
  - Can provide massive parallelism (100`s of Bit operations/cycle)
  - *Not useful for DP floating point operations*
  - *Memory bandwidth*
  - *Not a conventional programming approach*

Implementing iterative LBM – achieving spatial locality

Cache-Oblivious Blocking Approach
Discretization of LBM

- **Boltzmann Equation**

$$\partial_t f + \xi \cdot \nabla f = -\frac{1}{\lambda}[f - f^{(0)}]$$

ξ ... particle velocity
$f^{(0)}$ ... equilibrium distribution function
λ ... relaxation time

- **Discretization of particle velocity space**
  (finite set of discrete velocities)

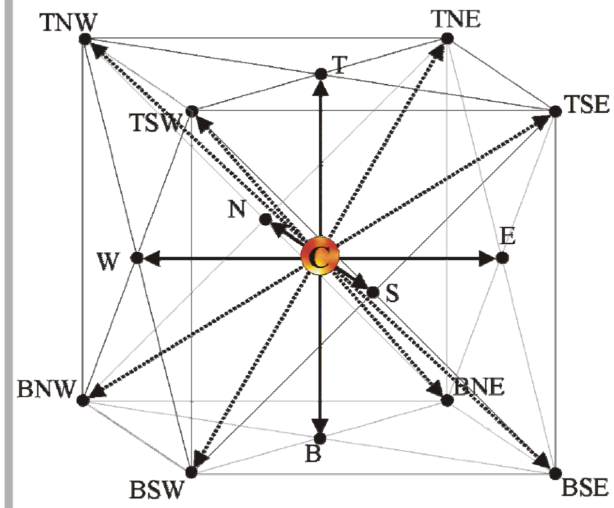
$$\partial_t f_\alpha + \xi_\alpha \cdot \nabla f_\alpha = -\frac{1}{\lambda}[f_\alpha - f_\alpha^{(eq)}]$$

$$f_\alpha(\vec{x},t) = f(\vec{x},\xi_\alpha,\mathrm{t})$$
$$f_\alpha^{(eq)}(\vec{x},t) = f^{(0)}(\vec{x},\xi_\alpha,\mathrm{t})$$

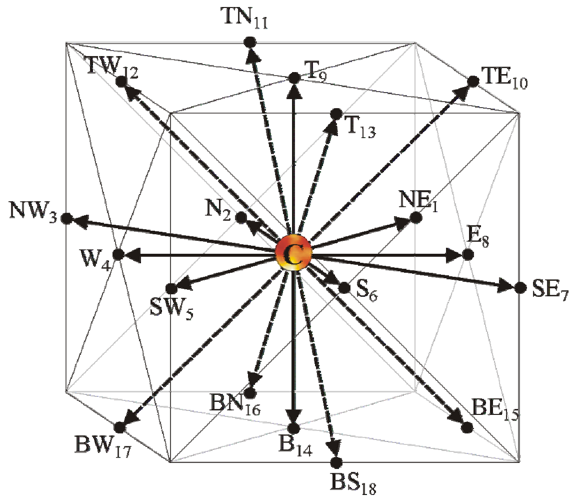
ξ_α – determined by discretization scheme

Cache-Oblivious Blocking Approach
Discretization schemes for LBM

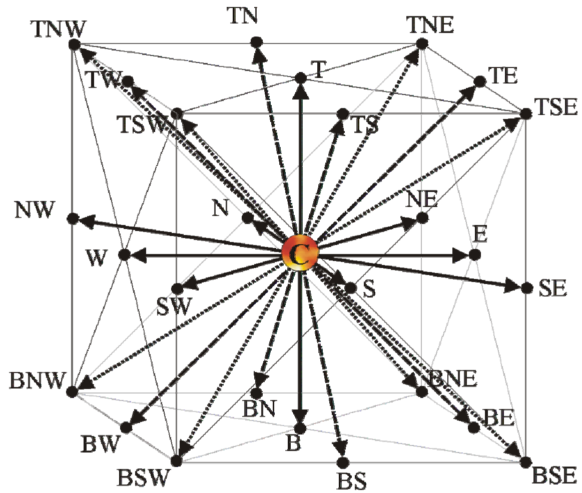
- **Different discretization schemes in 3D**
  - **Numerical accuracy and stability**
  - **Computational speed and simplicity**



D3Q15



D3Q19



D3Q27

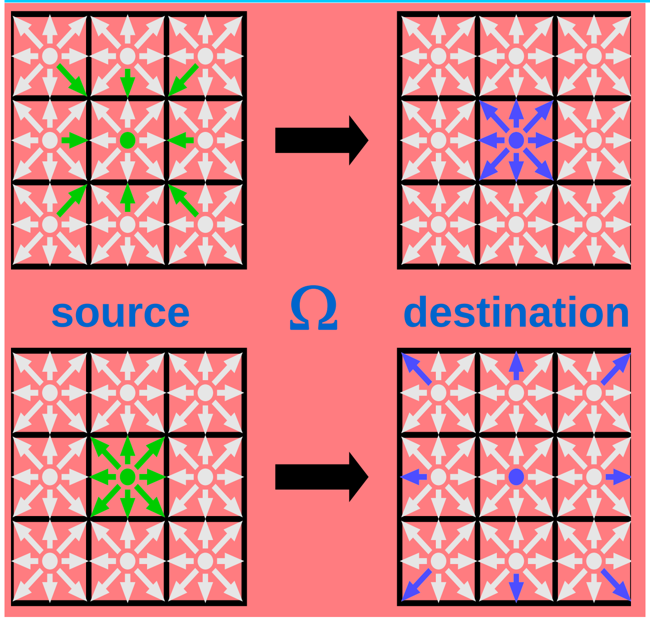
We choose D3Q19 because of good balance between stability and computational efficiency

Cache-Oblivious Blocking Approach
Stream and collide steps

- **Discretization in space $\vec{x}$ and time t:**

collision step: $\tilde{f}_\alpha(x_i,t) = f_\alpha(x_i,t) - \omega\,[\,f_\alpha(x_i,t) - f_\alpha^{(eq)}(x_i,t)]$

streaming step: $f_\alpha(x_i + \vec{e}_\alpha \delta t, t+\delta t) = \tilde{f}_\alpha(x_i,t)$



Stream-Collide (Pull-Method)

Get the distributions from the neighboring cells in the source array and store the relaxated values to one cell in the destination array

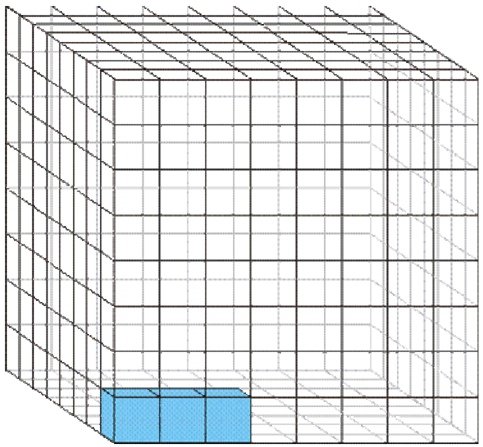
Collide-Stream (Push-Method)

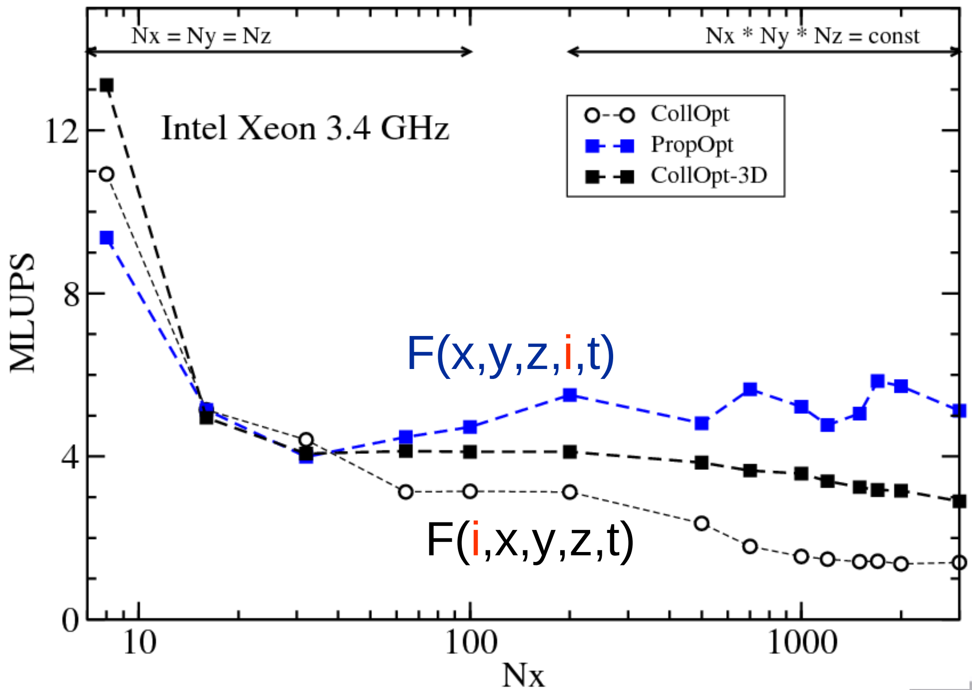
Take the distributions from one cell in the source array and store the relaxed values to the neighboring cells in the destination array

We choose Collide-Stream in what follows and do both steps in a single loop

Cache-Oblivious Blocking Approach
Spatial and temporal blocking – Basics (*Full matrix*)

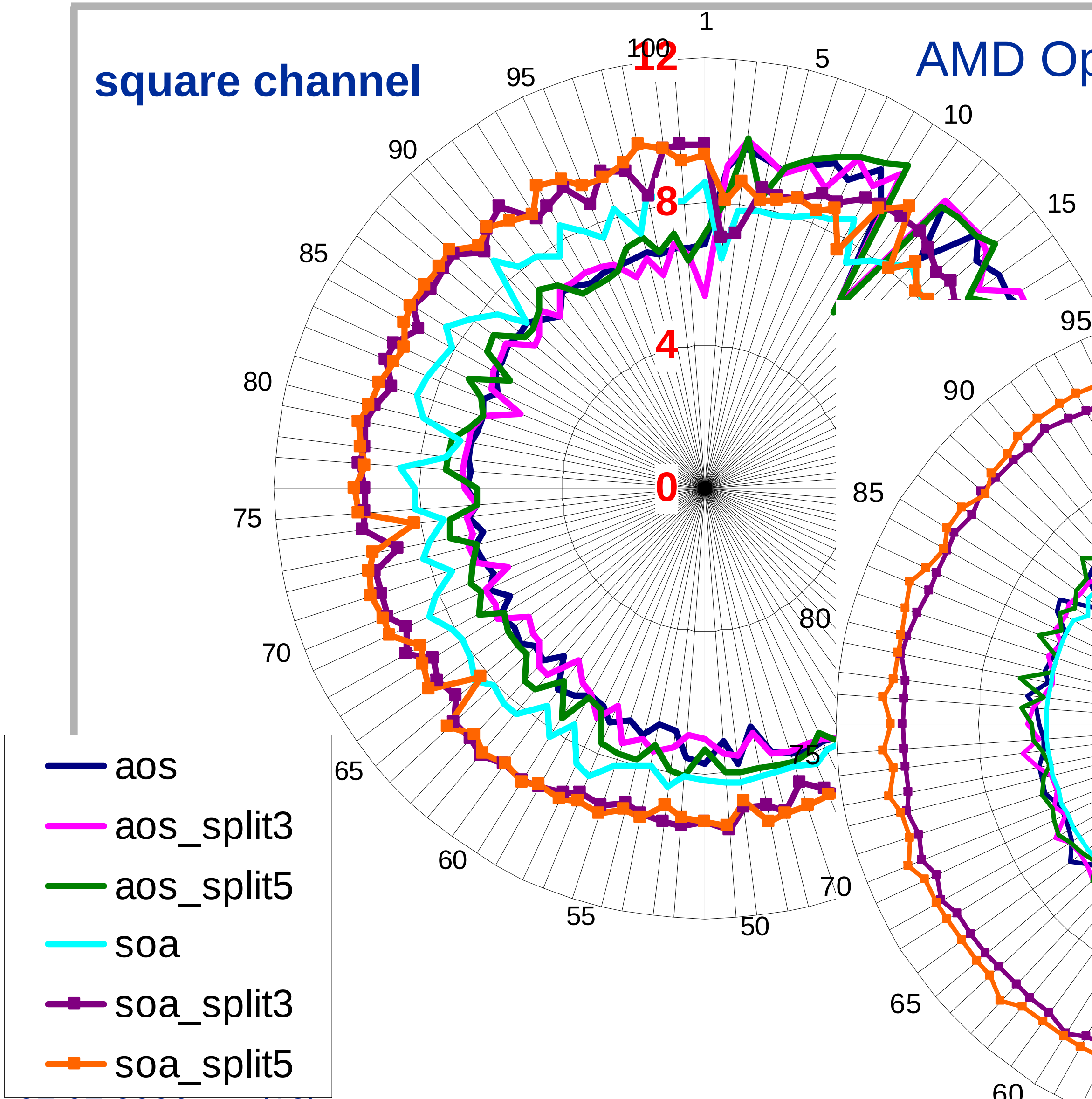
- Spatial blocking:
  Once a cache line is in the cache all entries should be used!
  - Investigate data-layout: **`F(i,x,y,z,t)`** vs. **`F(x,y,z,i,t)`**
  - Implement spatial blocking
  - Iterative LBM: Each time step is performed on all cells of the full domain
  - [1] G. Wellein, T. Zeiser, G. Hager, and S. Donath, Comp. & Fluids, Vol. 35 (2006)

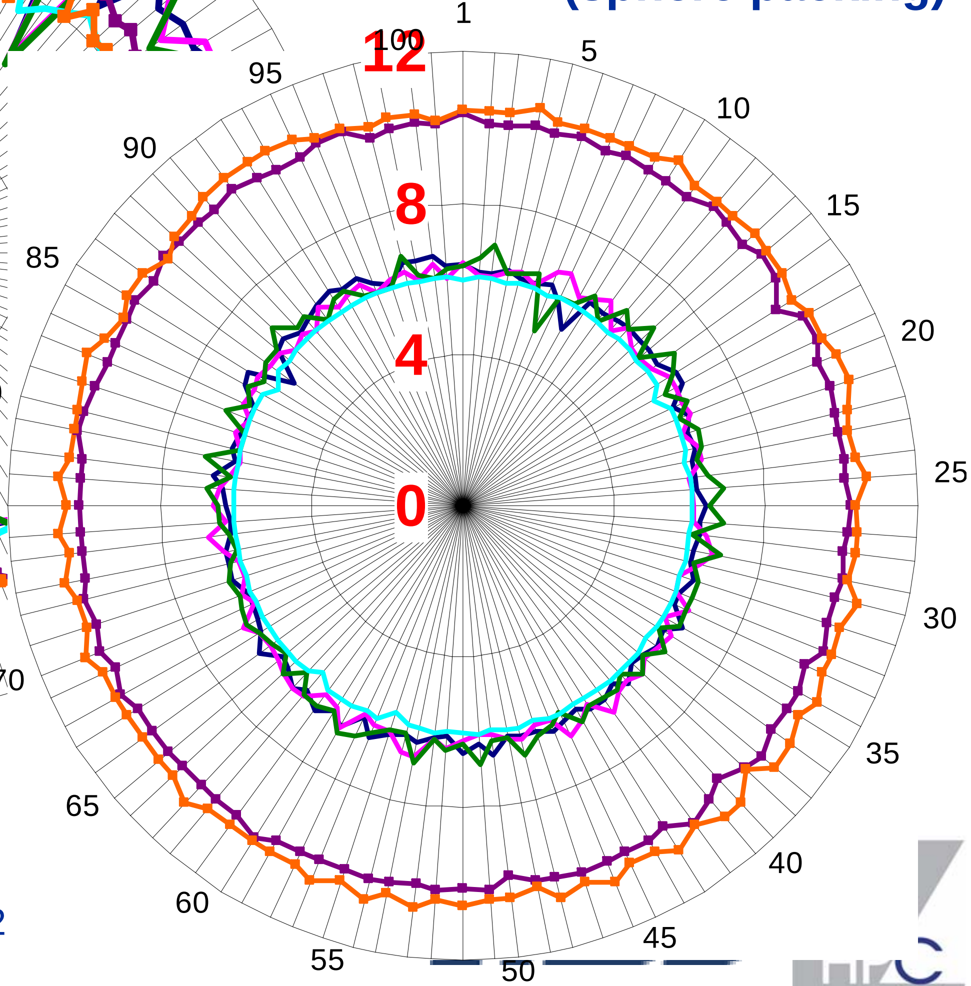




Cache-Oblivious Blocking Approach
Spatial and temporal blocking – Basics (*Sparse LBM*)

AMD Opteron 270 (4 cores 2GHz)





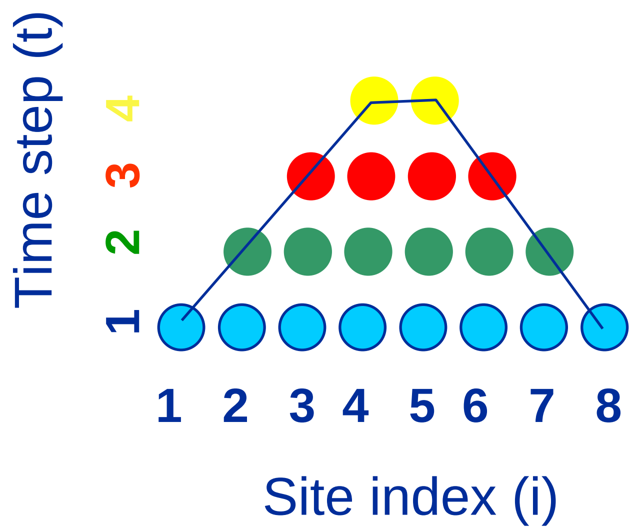
Cache-Oblivious Blocking Approach for the LBM – improving temporal locality

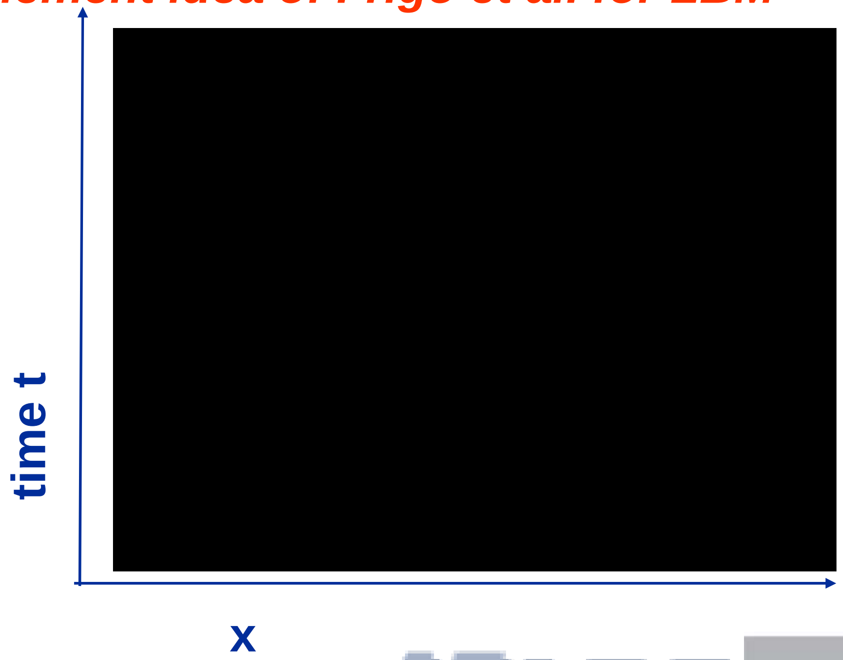
Cache-Oblivious Blocking Approach
Spatial and temporal blocking - Basics

- Temporal blocking: Load small blocks to cache and perform several time steps before loading next block
  - Choose appropriate block sizes
  - Optimize kernel for cache performance
  - Time-Blocked LBM applications:
    A fixed number of time steps will be performed on the domain

Implement idea of Frigo et al. for LBM

Block of 8 sites of a long 1D chain





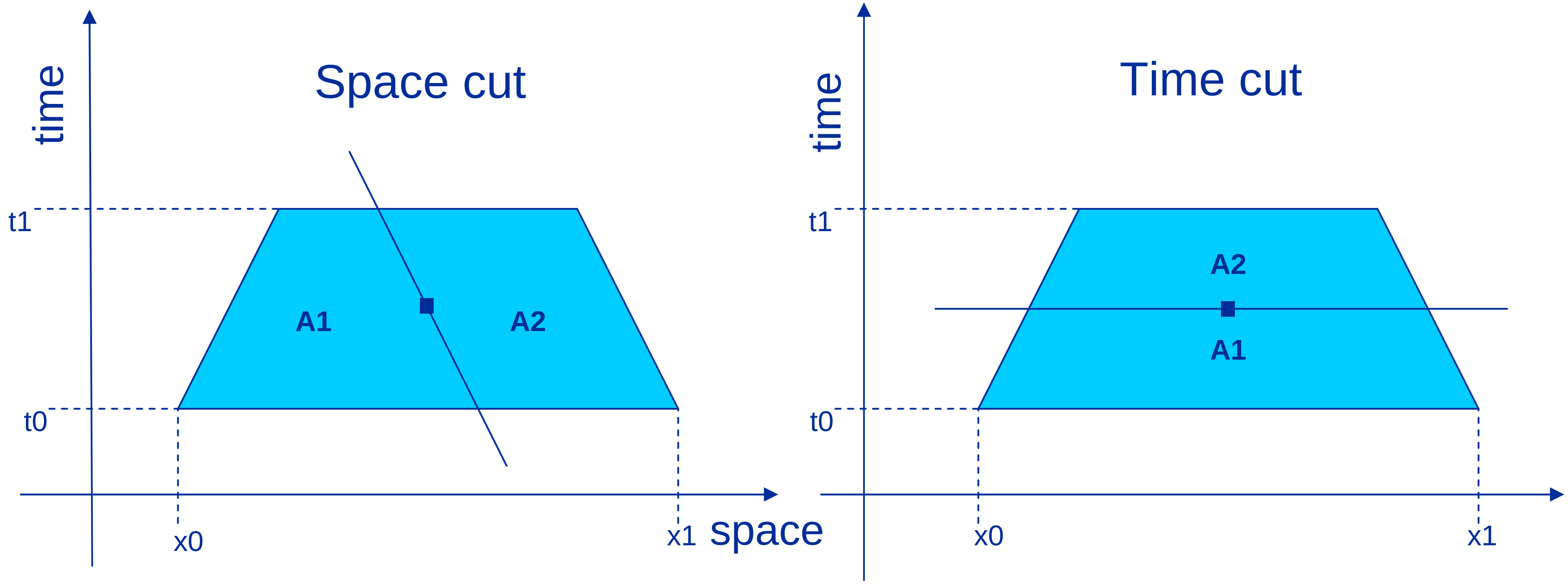
Cache-Oblivious Blocking Approach
Temporal blocking of LBM using Frigo's method

- Standard temporal blocking approaches:
  - Appropriate blocking sizes and time steps must be determined for each cache size & cache hierarchy
- Cache-Oblivious Blocking Approach introduced by Frigo, Prokop, et al. for matrix transpose / FFT / sorting
  - [2] Harald Prokop. Cache-Oblivious Algorithms. Masters thesis, MIT. 1999.
  - [3] M. Frigo, C.E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science* (FOCS 99), p.285-297. 1999
- Cache-oblivious blocking (recursive) approach (COBRA):
  - Independent of hardware parameters, e.g. cache size / cache-line length
  - Algorithms should choose an optimal amount of work and move data optimally among multiple levels of cache
  - Can easily be extended to stencils of higher order

Cache-Oblivious Blocking Approach
Basic Idea

- Recursive algorithm with space and time cuts to define domains which
  - fits into cache
  - allow several time steps to be performed

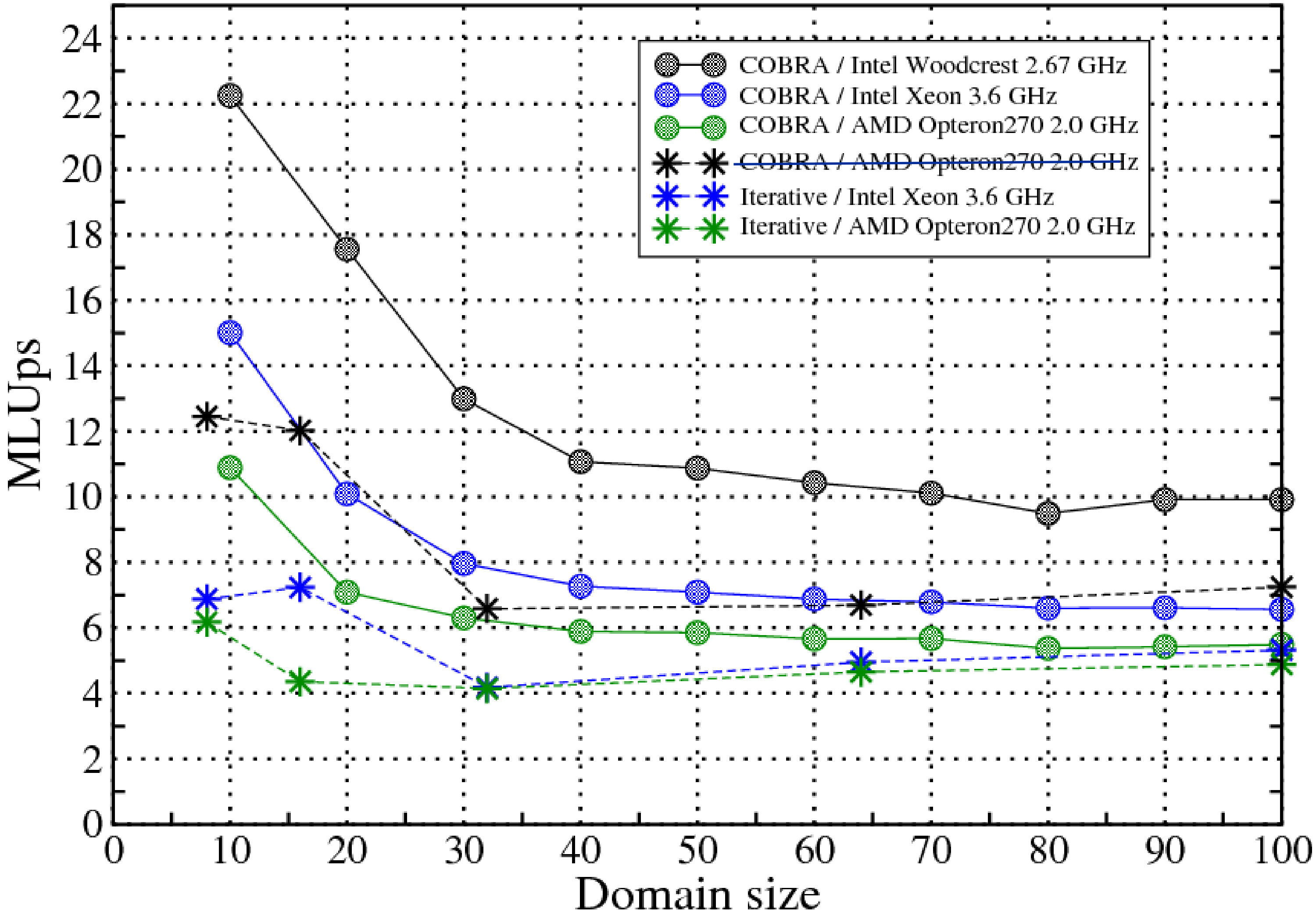
Example: 1 spatial dimension (1D)



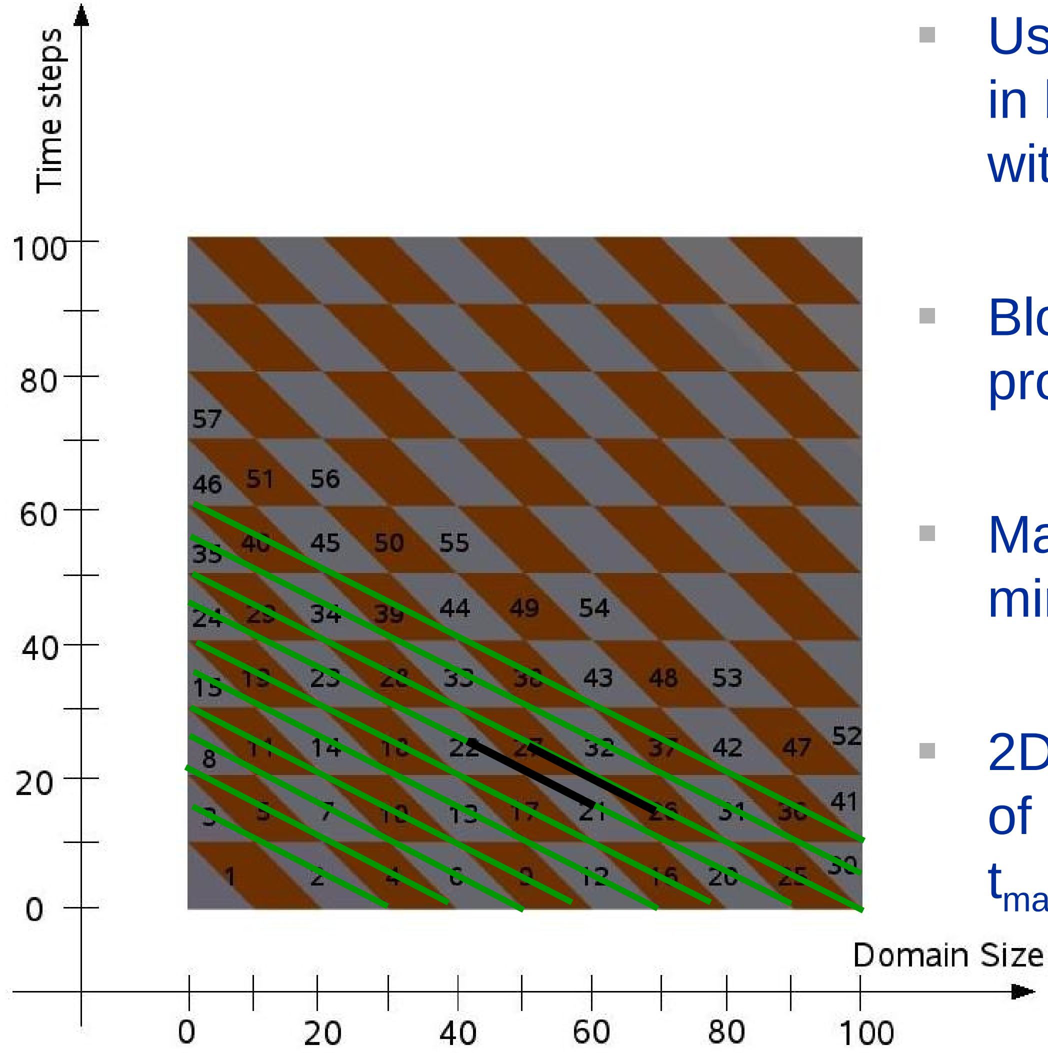
Cache-Oblivious Blocking Approach (COBRA)
Structure of recursive algorithm

```
void walk1(int t0, int t1, int x0, int ix0, int x1, int ix1)
{
  int dt = t1 - t0;
  if (dt == 1) {
      /* base case */
      Solve Kernel()
      }
      else if (dt > 1) {
      if (2 * (x1 - x0) + (ix1 - ix0) * dt >= 4 * dt) {
      /* space cut */
          int xm = (2 * (x0 + x1) + (2 + ix0 + ix1) * dt) / 4;
          walk1(t0, t1, x0, ix0, xm, -1);
          walk1(t0, t1, xm, -1, x1, ix1);
      } else {
      /* time cut */
          int s = dt / 2;
          walk1(t0, t0 + s, x0, ix0, x1, ix1);
          walk1(t0 + s, t1, x0 + ix0 * s, ix0, x1 + ix1 * s, ix1);
      }
    }
  }
```

Cache-Oblivious Blocking Approach (COBRA)
Single processor performance – COBRA vs. Iterative

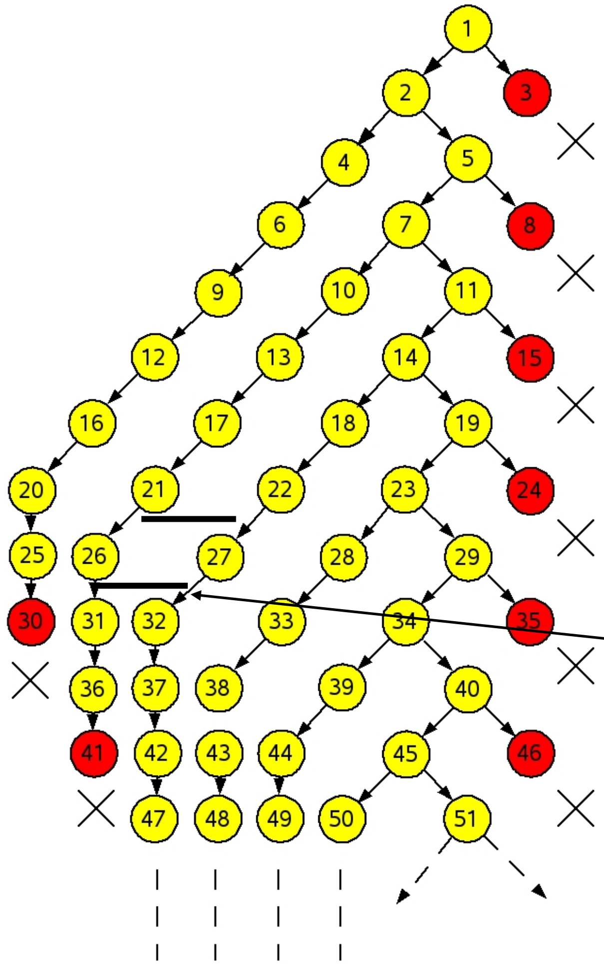


Cache-Oblivious Blocking Approach (COBRA)
Parallelization – Simple wavefront in spacetime

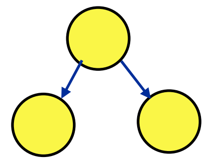


- Use COBRA to cut spacetime in blocks of size t_Bx t_B with t_B<< t_{max}
- Blocks on diagonals can be processes in parallel
- Max. length of diagonal: min(t_{max} / t_B ; D / (2* t_B))
- 2D spacetime projection of 3D domain (100^3) with t_{max}=100 & t_B=10

Cache-Oblivious Blocking Approach
Parallelization – Implementation

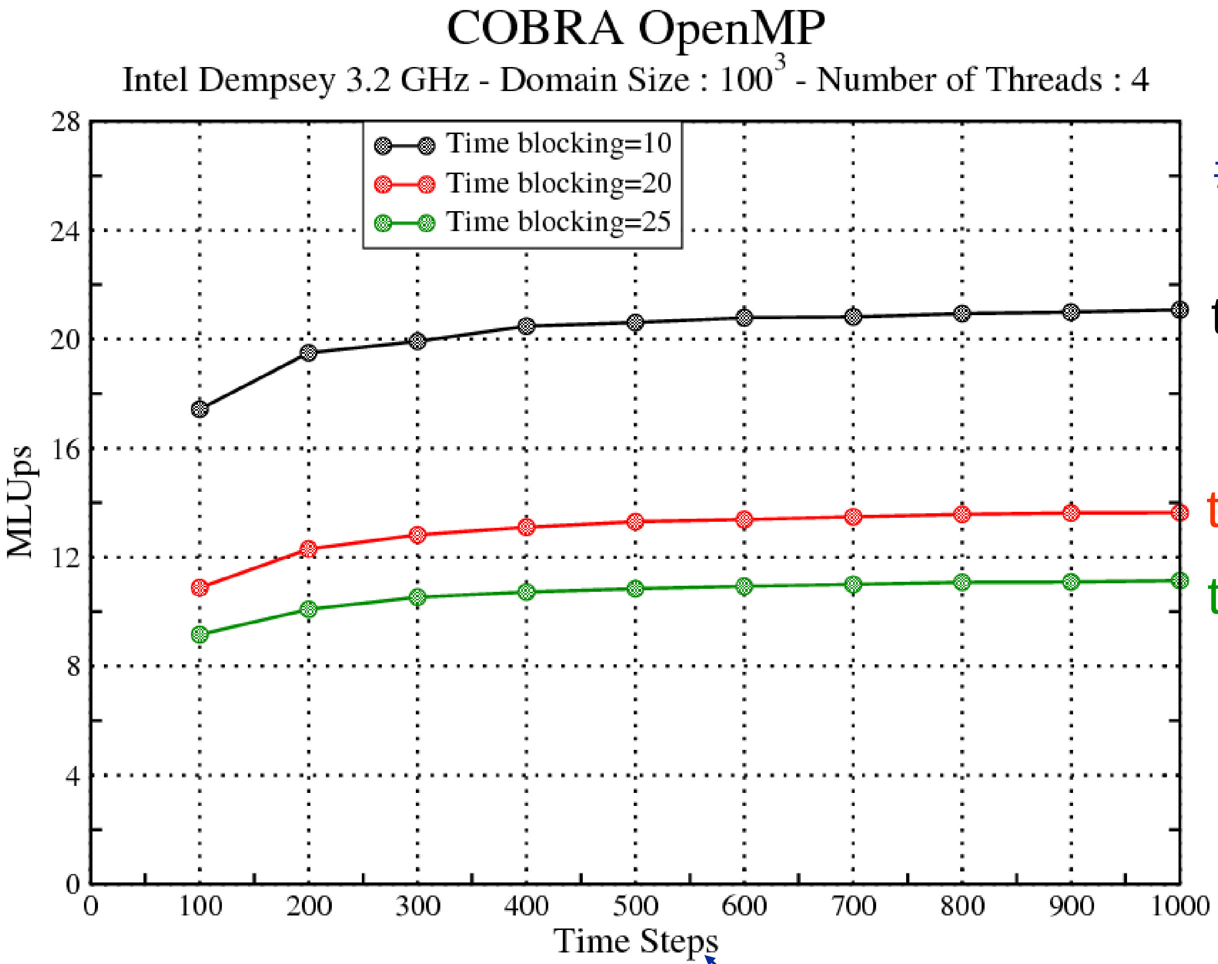


- Use shared memory approach
- Standard OpenMP does not yet fully support nested parallelism



- Use Intel`s `#pragma intel omp taskq`
- Use OpenMP locking functions to synchronize threads before starting the next wavefront

Parallel performance & scalability ⟺ t_B, t_{max}, #threads



D=100
#threads=4

t_B=10 -> 5 diags.

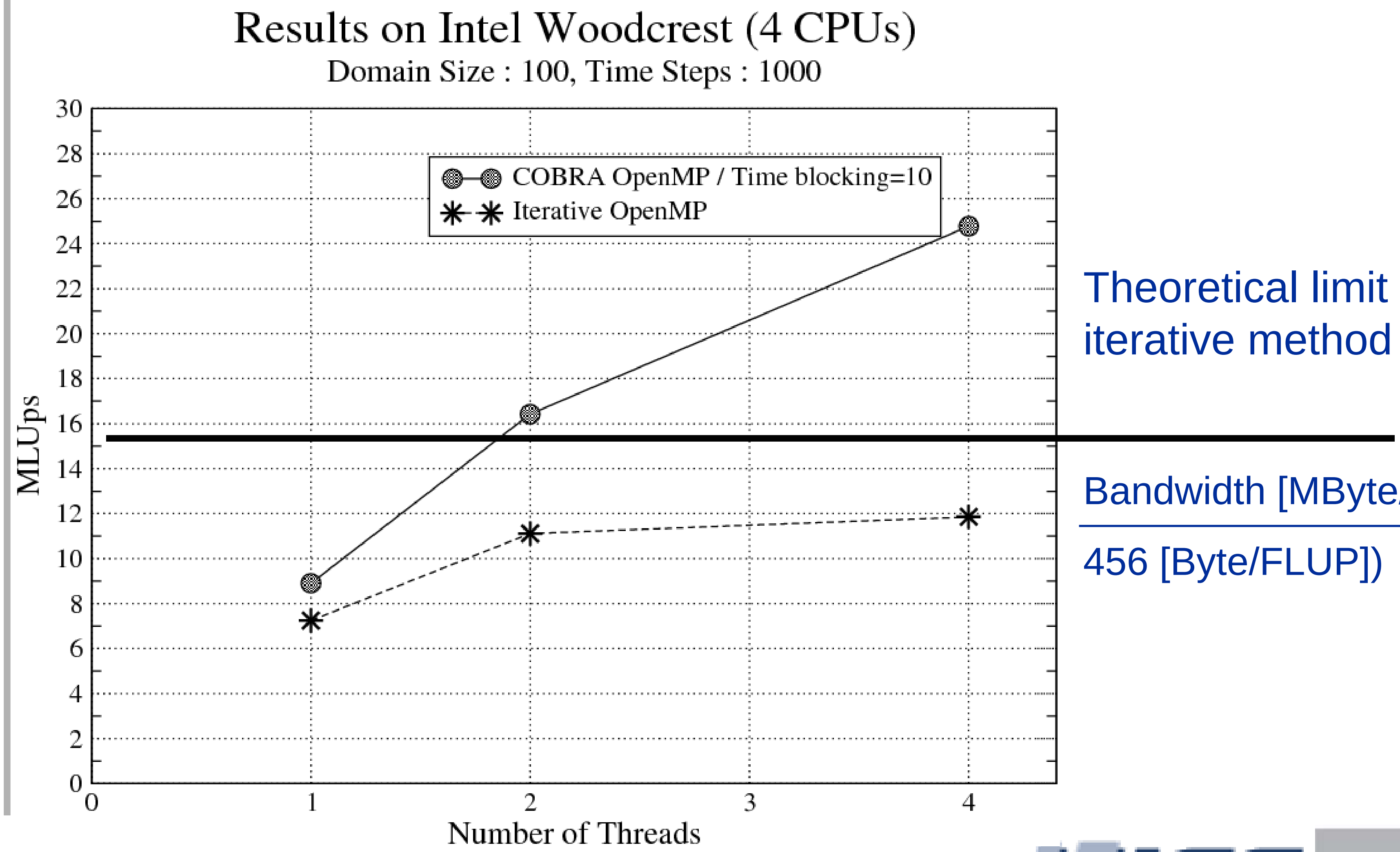
t_B=20 -> 2,5 diags.

t_B=25 -> 2 diags.

Max. length of diagonal: min(t_{max} / t_B ; D/(2* t_B))

Cache-Oblivious Blocking Approach (COBRA)
Parallelization – Parallel performance & scalability

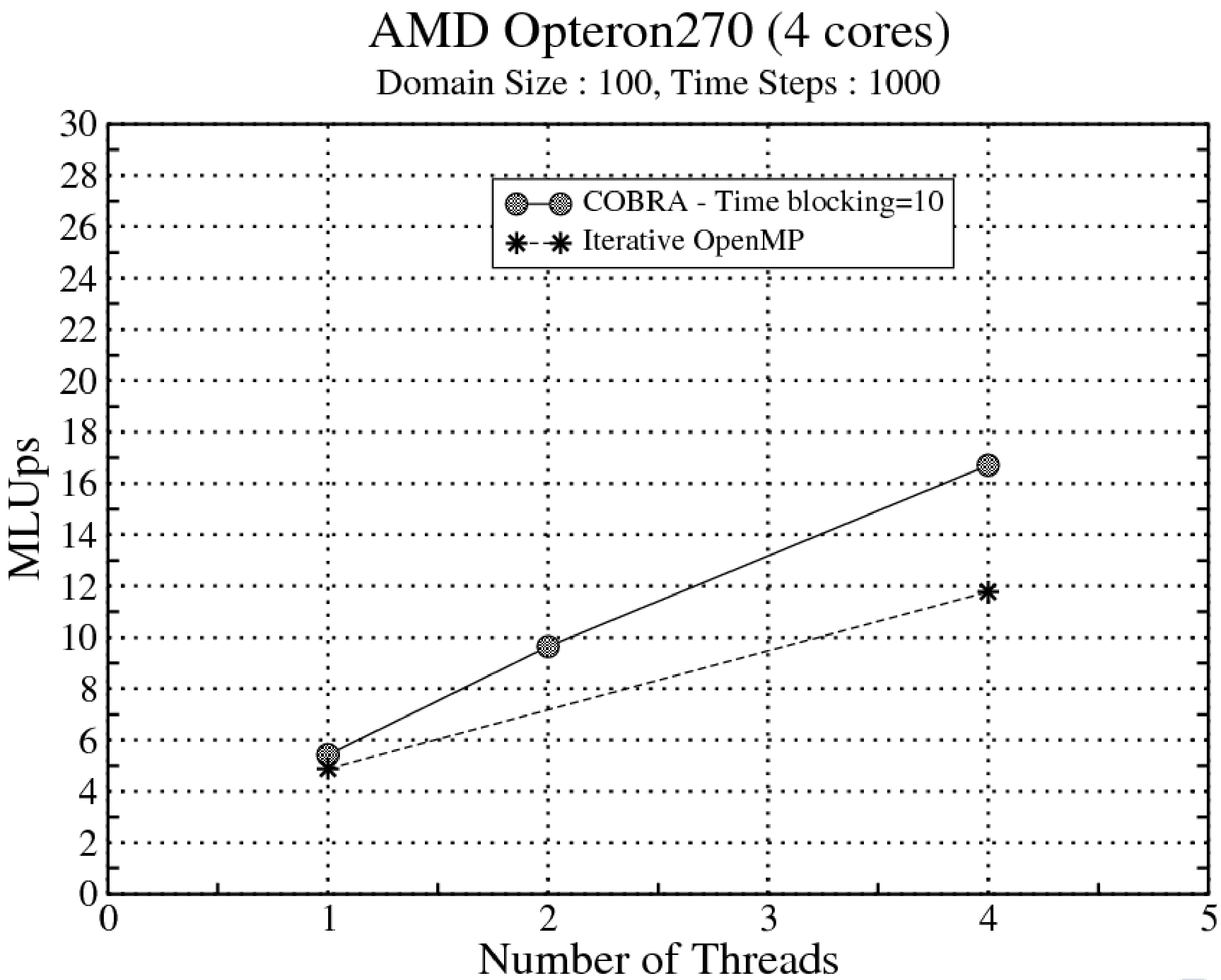
- Parallel performance & scalability ⟺ Block size t_B, t_{max}, #threads



Theoretical limit of iterative method = Bandwidth [MByte/s] / 456 [Byte/FLUP])

Cache-Oblivious Blocking Approach (COBRA)
Parallelization – Parallel performance & scalability

- Parallel performance & scalability ⟺ Block size t_B, t_{max}, #threads

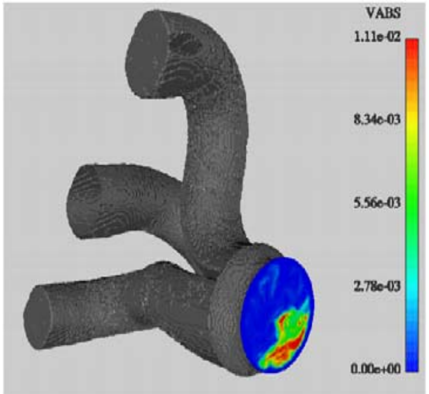


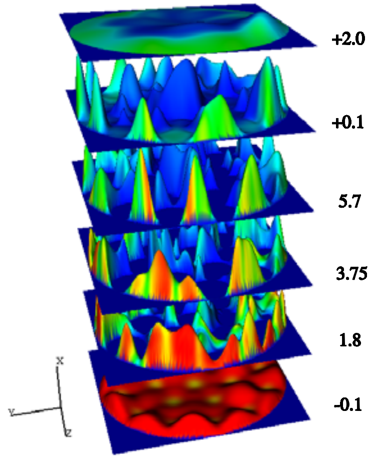
Summary & Outlook

- **Iterative LBM:**
  - **Efficient implementation strategies for simple (full matrix) and complex (sparse LBM) geometries are available**
  - **Easy parallelization through domain decomposition (pref. MPI)**

- **Time blocked LBM:**
  - **Cache-Oblivious Blocking Approach (COBRA) has high potential to overcome the bandwidth limitations for simple geometries**
  - **Shared memory parallelization through task-queue model**

  - **Use on complex geometries (sparse LBM) ?**
  - **Pure MPI parallelization – large overhead through multiple ghost layers!**
  - **Hybrid parallelization approach (OpenMP within node + MPI between nodes)?**





Thank you!

Bayerisches Staatsministerium für
Wissenschaft, Forschung und Kunst

KONWIHR

http://www.rrze.uni-erlangen.de/hpc/

Acknowledgement:

R. Vogelsang, R. Wolff (SGI)

W. Oed (CRAY)

Th. Schoenemeyer (NEC)

Acknowledgement:

M. Brehm et al. (LRZ)

U. Küster, P. Lammers (HLRS)

H. Cornelius, H. Bast, A. Semin (Intel)