Are the Killer Micros Still Attacking?

Some Random Thoughts From Users' Perspectives

Georg Hager

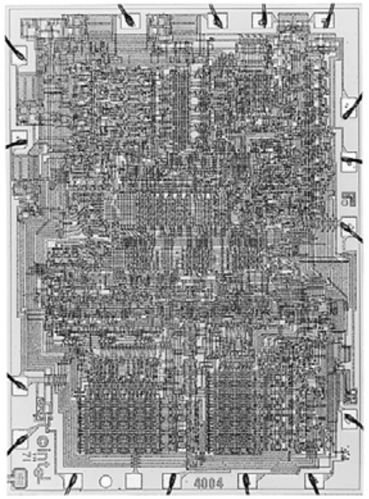
Gerhard Wellein, Thomas Zeiser

Regionales Rechenzentrum Erlangen

University of Erlangen-Nuremberg

Germany

NUG Workshop May 2007



Outline

- “Killer Micros”
- Current Killer Micros
- Complications: Diversity versus Choice
  - Is the single core still good enough?
  - Multi-core nuisances
  - ccNUMA obstacles
  - The Bottleneck Blues
- HPC wishlist
- Are they still attacking?

… and this talk does not contain a single Top500 figure!

The “Killer Micro”

- **Horst D. Simon, ISC 2005 keynote presentation:**
  “*Progress in Supercompting: The Top Three Breakthroughs of the Last 20 and the Top Three Challenges for the Next 20 Years*”
  - **10 reasons why supercomputing dramatically changed from 1985 – 2005:**

    10)The TOP500 list

    9) NAS Parallel Benchmark

    8) The “grid”

    7) Hierarchical algorithms: multigrid and fast multipole

    6) The “Attack of the Killer Micros”

    5) HPCC initiative and Grand Challenge applications

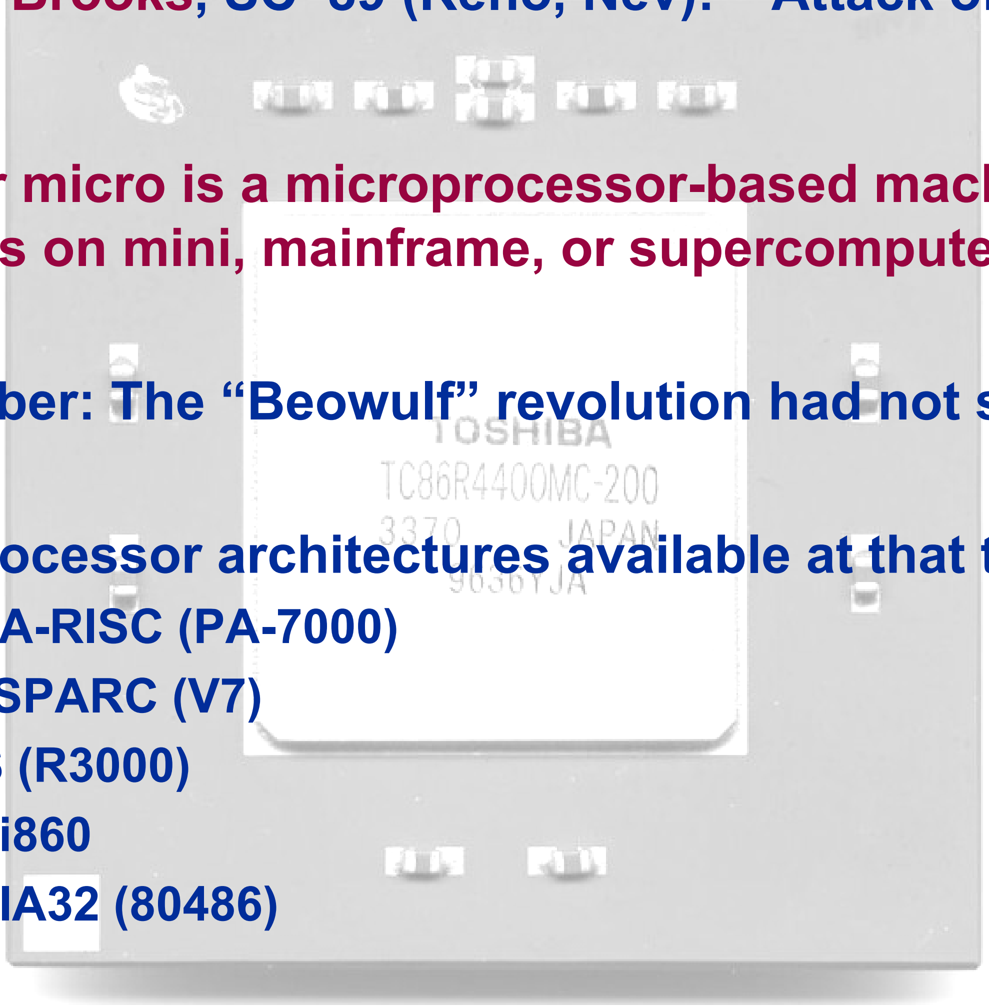
    4) Beowulf clusters

    3) Scientific visualization

    2) MPI

    1) Weak scaling (scaled speedup)

The “Killer Micro”



- Eugene Brooks, SC ’89 (Reno, Nev): “Attack of the Killer Micros”

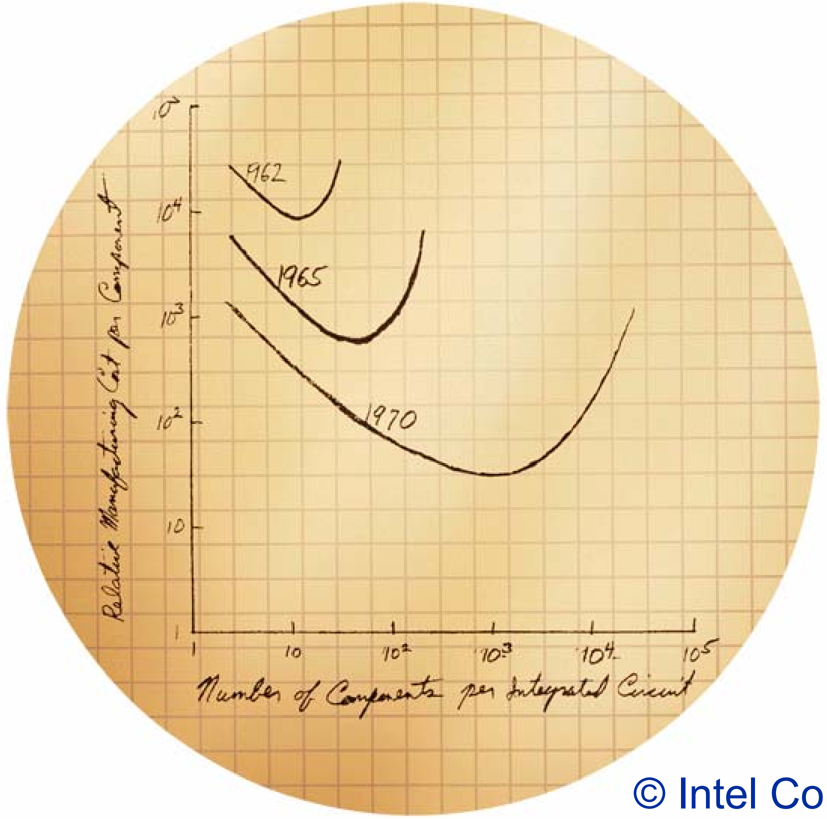
  “A killer micro is a microprocessor-based machine that infringes on mini, mainframe, or supercomputer performance”

- Remember: The “Beowulf” revolution had not started yet!

- Microprocessor architectures available at that time
  - HP PA-RISC (PA-7000)
  - Sun SPARC (V7)
  - MIPS (R3000)
  - Intel i860
  - Intel IA32 (80486)

Killer Micros today:
What are we going to do with all those transistors?

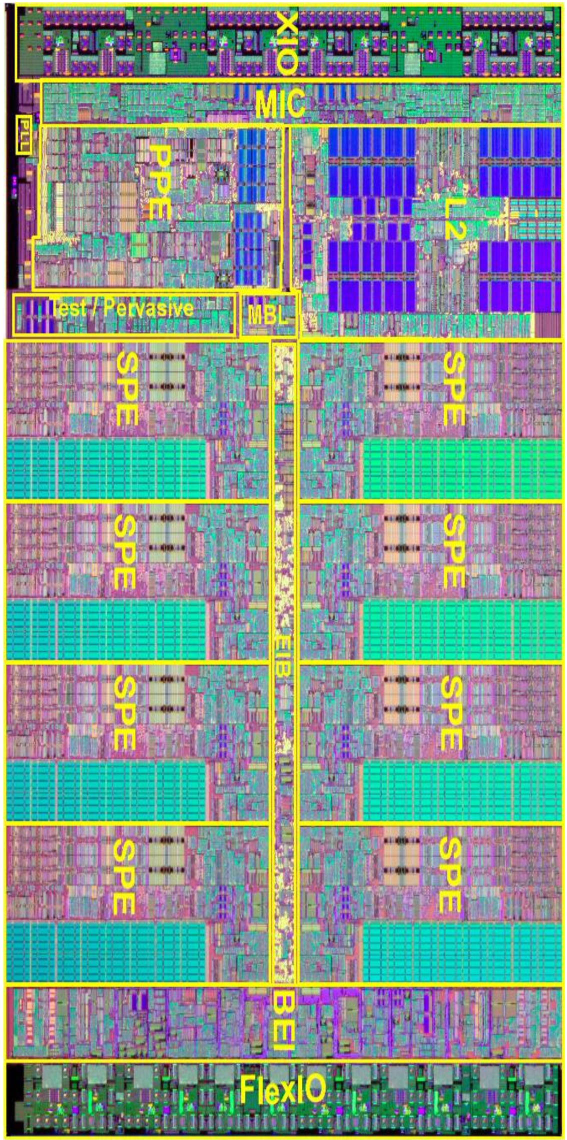
- Intel IA32 (-compatible) designs are rivalling all established microprocessor architectures
  - … or have already swept them off the market
  - Intel Core 2 – “Woodcrest”
- AMD64 – “Opteron”
  - Latency-optimized
  - ccNUMA for the masses – but are users actually aware of that?
- Intel IA64 – “Itanium 2”
  - Good concept
  - Gaining speed (too?) slowly
  - Technological difficulties



© Intel Corp.

Killer Micros today:
What are we going to do with all those transistors?

- IBM Power5
  - Extremely expensive to get in a decent HPC environment
- Sun Niagara I & II
  - On the way to massively multithreaded design
  - Latency hiding by threading (did anyone say "MTA"?)
  - Decent memory BW at a moderate price
- PowerPC 440 @ BlueGene/L
  - Not really a killer micro…
  - BG/L as blueprint for the future?
- Special purpose hardware
  - IBM Cell Broadband Engine (CBE) →
  - FPGA
  - ClearSpeed



Diversity vs. choice

- Challenges for HPC software developers and users

ccNUMA

Pseudo-UMA

Multi-core

“Bottleneck computing”

SIMD

Performance tools

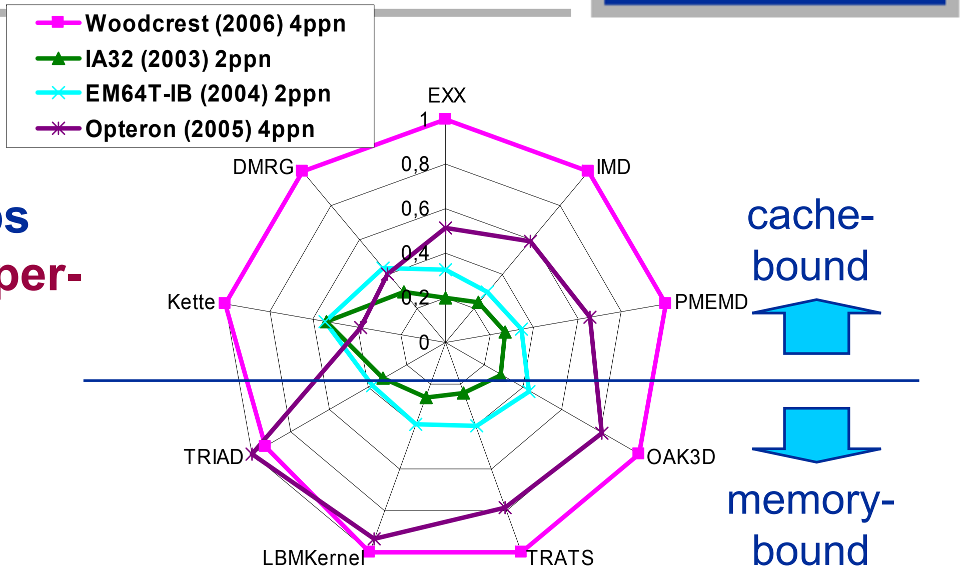
The “BlueGene effect”

Accelerator HW

OS issues

The BlueGene effect: Is Moore's party over?

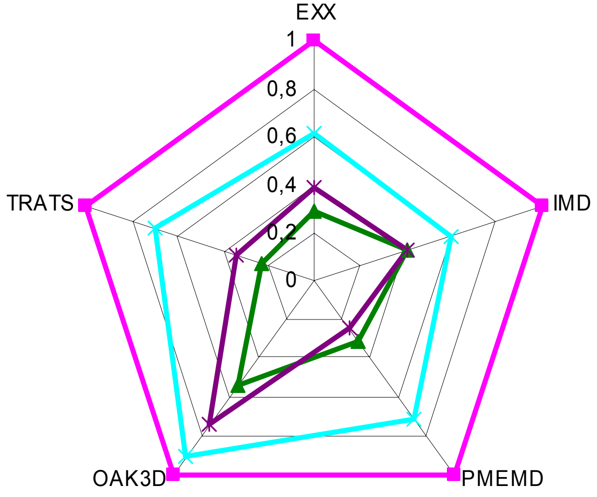
- **Hypothesis: 1-core performance will stagnate; only multi-core chips will prevent the performance-per-transistor ratio to hit the floor**
- **1-node shootout with RRZE benchmark suite (MPI/OpenMP/serial):**



cache-bound

memory-bound

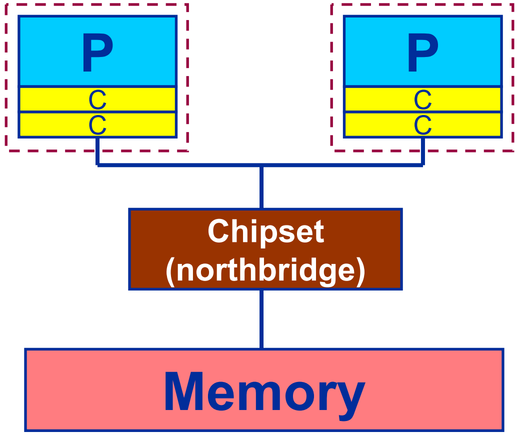
- **Parallel shootout (MPI):**



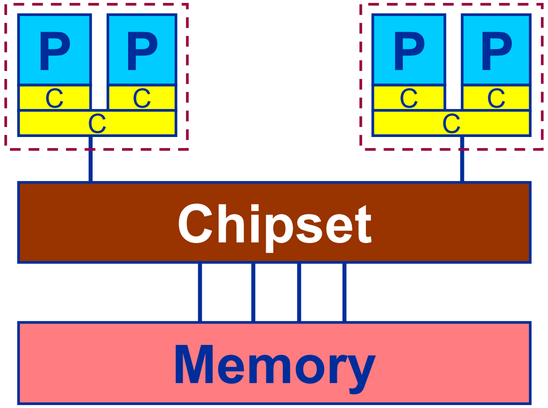
- **The current chips still seem "good enough"**

UMA vs. pseudo-UMA vs. ccNUMA

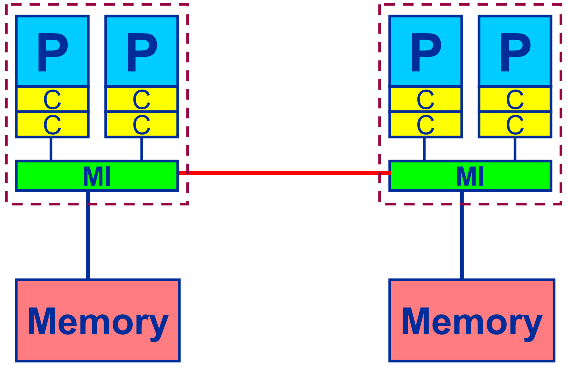
- Dual CPU Intel Xeon node



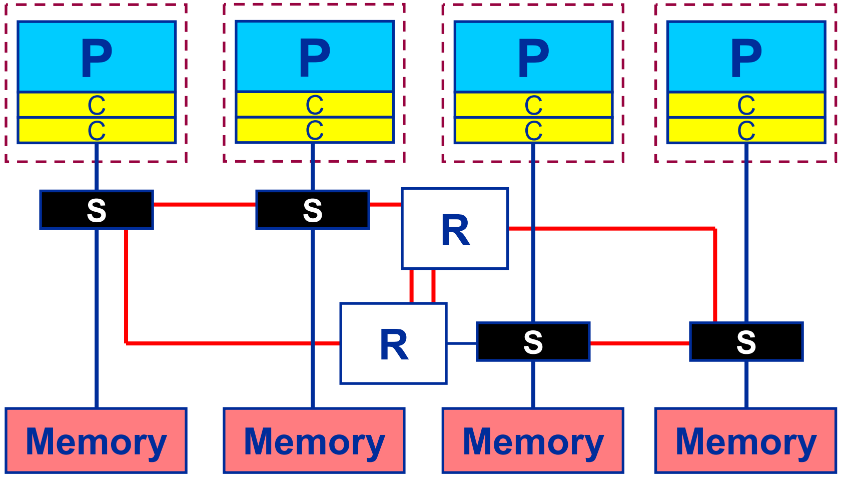
- Dual Intel "Core" node



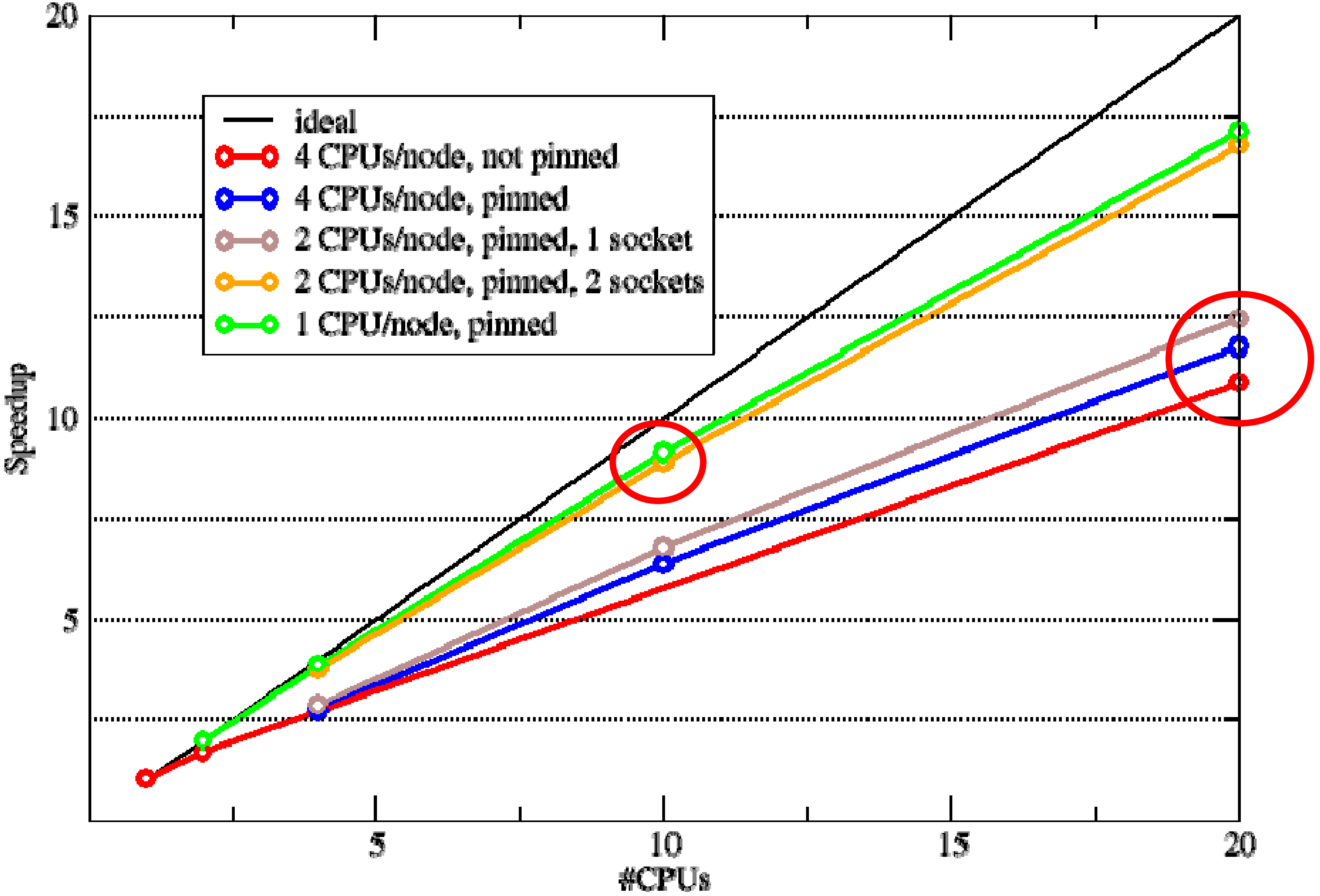
- Dual AMD Opteron node



- SGI Altix

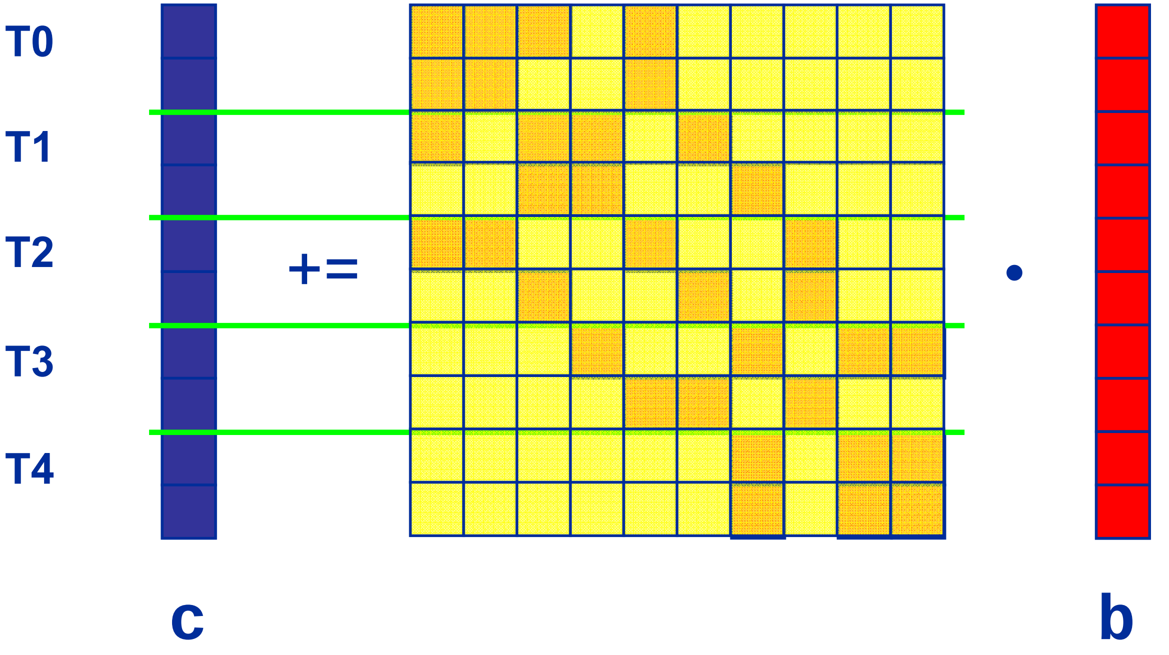


Pseudo-UMA example: ABINIT strong scalability on Woodcrest cluster



ccNUMA Example:
Data Parallelism for Sparse MVM

- Parallelize the loop that treats consecutive elements of result vector (or consecutive matrix rows)
- General idea:



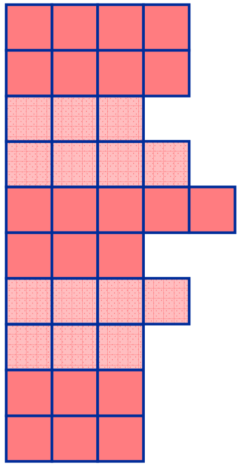
- RHS vector is accessed by all threads
  - … but this is shared memory, so it does not have to be stored multiple times!

OpenMP Parallelization of CRS MVM

- Parallelized loop is outer loop

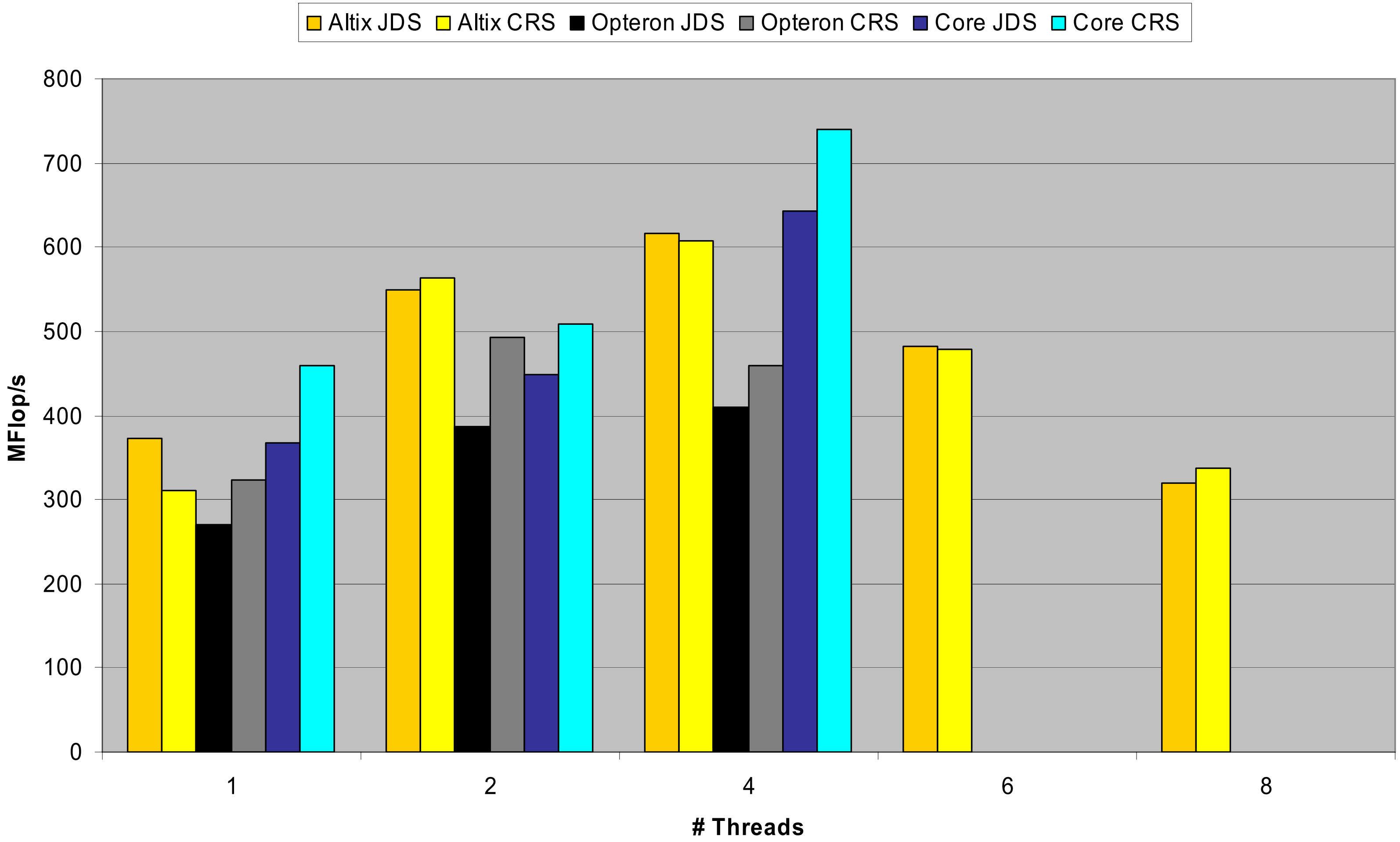
```
!$OMP parallel do
do i = 1,N_r
 do j = row_ptr(i), row_ptr(i+1) - 1
  c(i) = c(i) + val(j) * b(col_idx(j))
 enddo
enddo
!$OMP end parallel do
```

- Features
  - Long outer loop
    - small OpenMP overhead
  - Variable length of inner loop
    - possible load imbalance



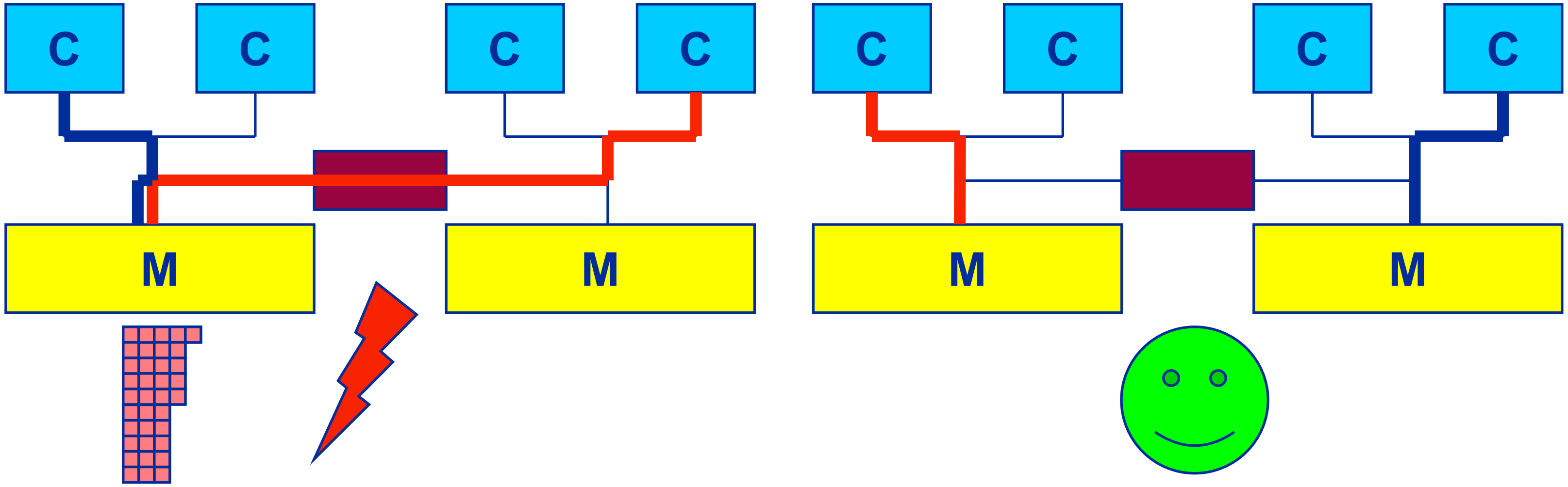
The Pitfalls of ccNUMA

Naïve approach for OpenMP parallelization fails on ccNUMA



Memory Locality Problems

- ccNUMA:
  - whole memory is transparently accessible by all processors
  - but physically distributed
  - with varying bandwidth and latency
  - and potential congestion ("overcommitted" memory paths)
- → Make sure that memory access is always as "local" and "distributed" as possible!



Data Locality in Parallel Sparse MVM

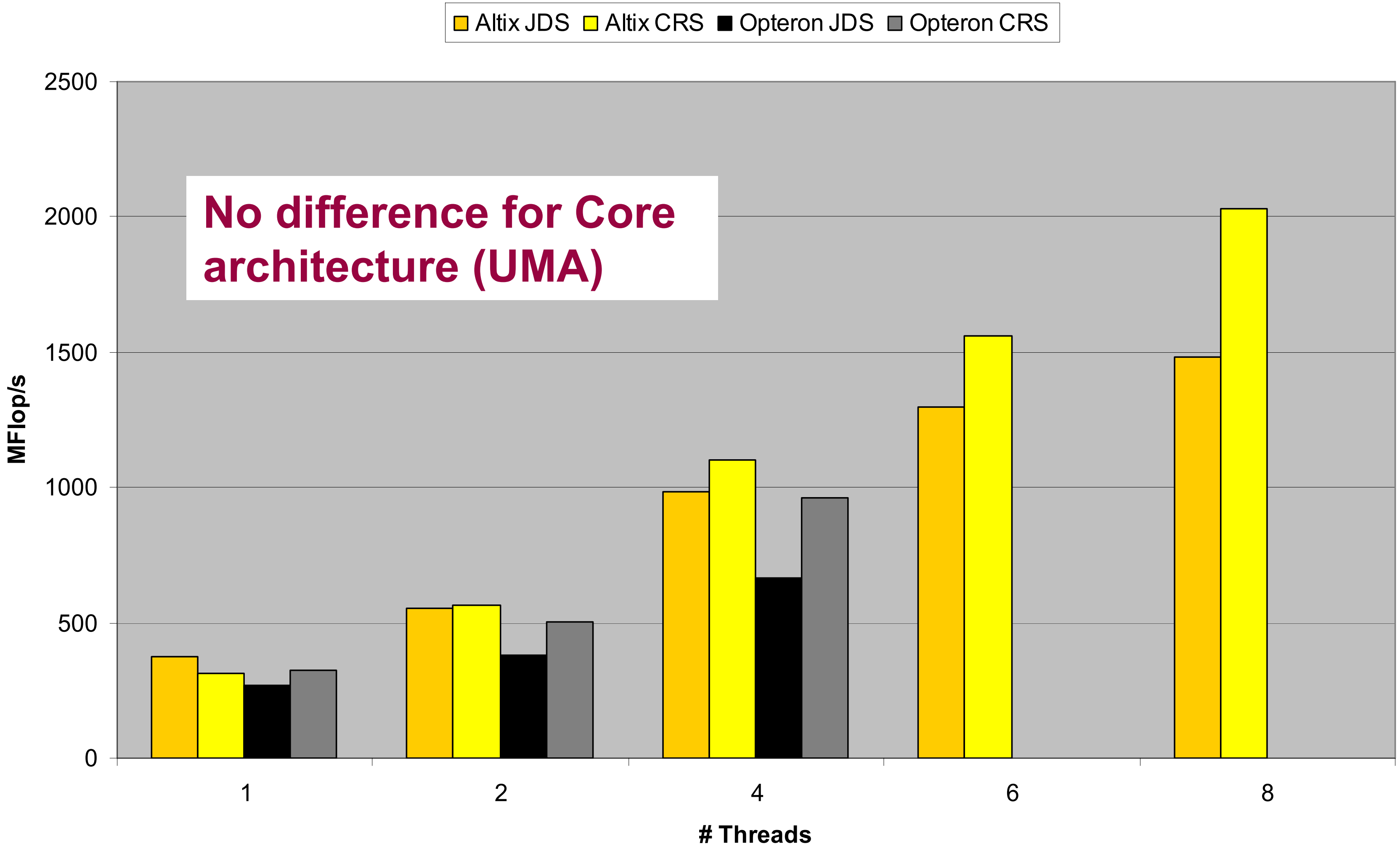
- No code change in MVM loop required (apart from static schedule)
- CRS
  - Initialization of arrays val[], c[], b[], row_ptr[] and col_idx[] must be done in parallel

```
!$OMP parallel do private(start,end,j)
!$OMP& schedule(static)
do i=1,N_r
  start = row_ptr(i)
  end = row_ptr(i+1)
  do j=start,end-1
    val(j) = 0.d0
    col_idx(j)= 0
  enddo
enddo
```

- Similar for JDS

Parallel Sparse MVM
Doing It Right^{TM} on ccNUMA

- Correct placement leads to acceptable scalability



No difference for Core architecture (UMA)

Doing It Right^{TM} with C++ on ccNUMA

- Don't forget that constructors tend to touch the data members of an object. Example:

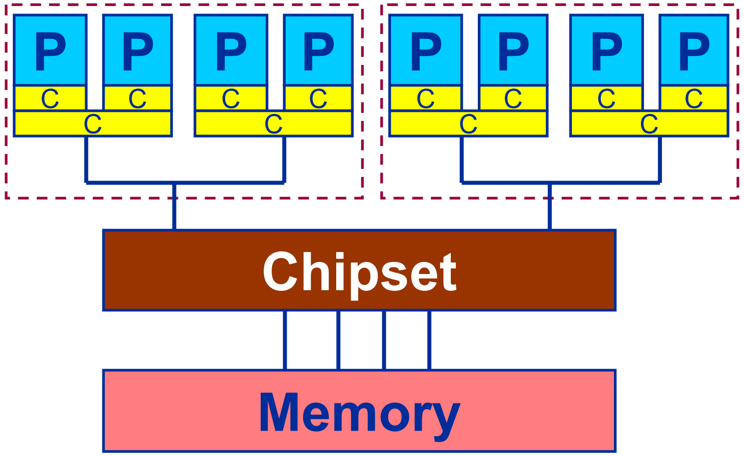
```
class D {
  double d;
public:
  D(double _d=0.0) throw() : d(_d) {}
  inline D operator+(const D& o) throw() {
    return D(d+o.d);
  }
  inline D operator*(const D& o) throw() {
    return D(d*o.d);
  }
...
};
```

→ placement problem with `D* array = new D[1000000];`

- Solutions
  - Overload `operator new` for parallel first touch
  - Use STL container with custom NUMA-aware allocator class

Bottleneck Computing on Quad-Core

- Multi-core obviously establishes
  - Bandwidth bottlenecks of increasing severity
  - More “where-to-run-what” diversity



- Maybe we shouldn’t rant about architectures but go back to the drawing board and redesign our algorithms…
  - Data access patterns, flop-to-load ratio, time blocking

Optimal data access patterns: Lattice Boltzmann Method & Discretization Scheme

- **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)**

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

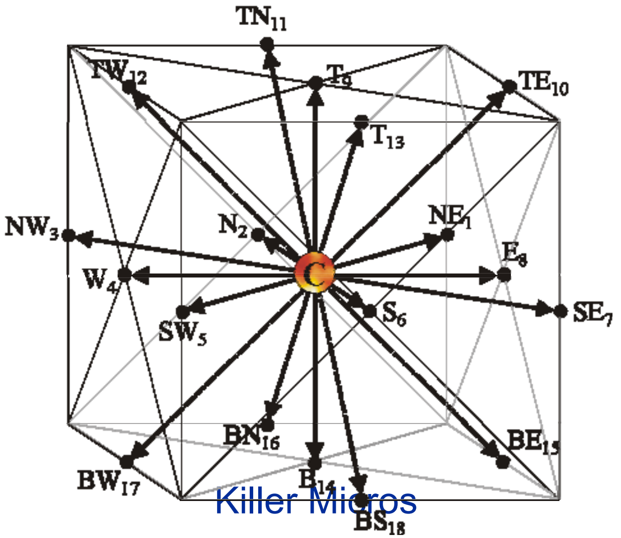
$$f_\alpha(\vec{x},t) = f(\vec{x},\xi_\alpha,\mathrm{t})$$

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

ξ_α – determined by

discretization scheme:

Choose D3Q19 in 3 dimensions



Optimal data access patterns:
Basic implementation strategy

```
double precision F(0:18,0:xMax+1,0:yMax+1,0:zMax+1,0:1)
do z=1,zMax
   do y=1,yMax
      do x=1,xMax
          if( fluidcell(x,y,z) ) then
             LOAD F(0:18,x,y,z,t)
             Relaxation (complex computations)
             SAVE F( 0,x  ,y  ,z  ,t+1)
             SAVE F( 1,x+1,y+1,z  ,t+1)
             SAVE F( 2,x  ,y+1,z  ,t+1)
             SAVE F( 3,x-1,y+1,z  ,t+1)
             …
             SAVE F(18,x  ,y-1,z-1,t+1)
          endif
      enddo
   enddo
enddo
```

LD 1-2 Cachelines (cont. access)

Collide Step

LD & ST
19 Cachelines

Stream Step

Cache line read for ownership (RFO)!

#load operations: 19*xMax*yMax*zMax + 19*xMax*yMax*zMax

#store operations: 19*xMax*yMax*zMax

Optimal data access patterns:
Setting up a simple performance model for LBM

- **Crossover between cache and memory bound computations:**

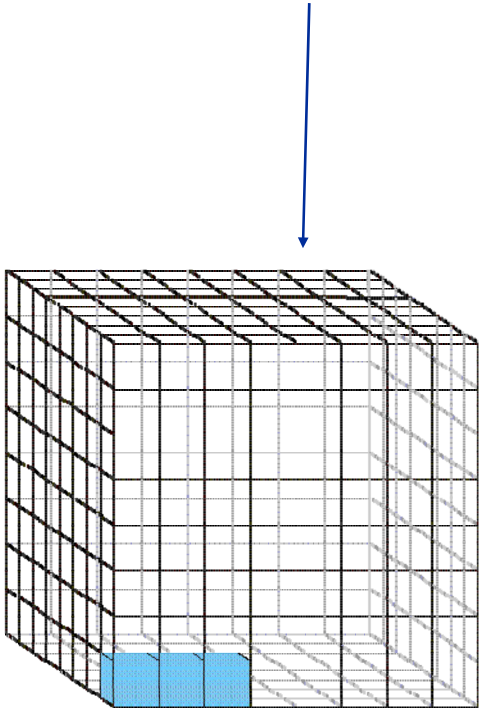
  2 *19 *$xMax^3$ * 8Byte ~ L2/L3 cache size → xMax ~ 14-15 (for 1 MB cache)

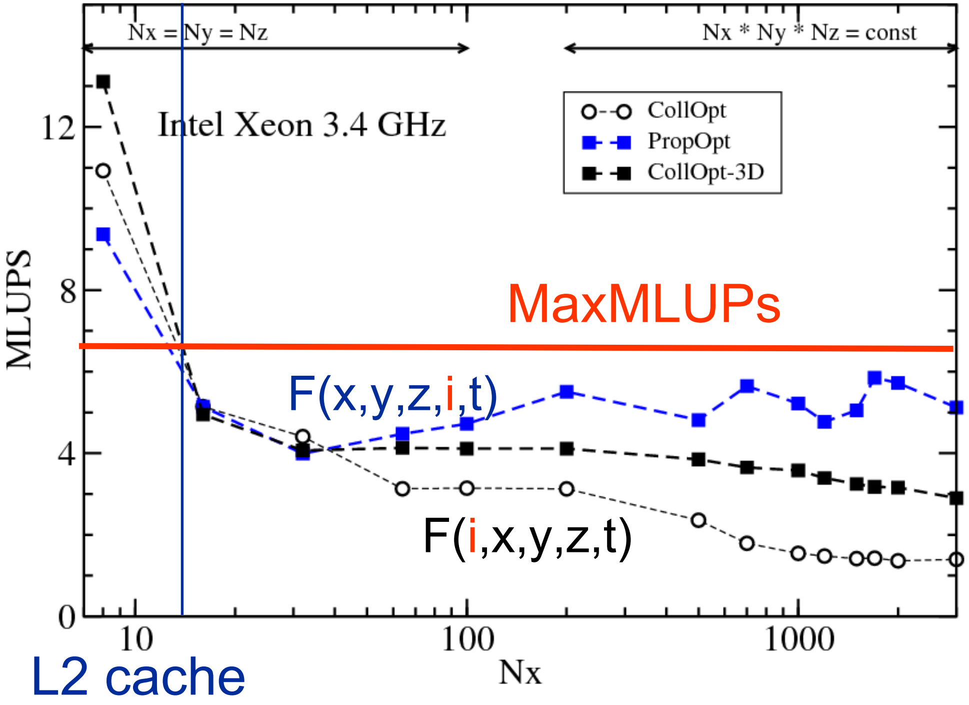
- **Data must be transferred from and to main memory in each time step – “Full time step LBM” (FTS-LBM):**
  - Assumption: full use of each cache line loaded
  - Data to be transferred for a single fluid cell update:
    (2+1)*19*8 Byte → 456 Bytes/(cell update)
  - Max. number of lattice site updates per second MLUPs:
    MaxMLUPs=MemoryBandwidth / (456 Bytes/cell update)
  - *MemoryBandwidth = 3-4 GByte/s → MaxMLUPs~ 6-8 MLUPs*
  - Check efficiency of data access: Minimum number of bus accesses:
    BusAccesses = N^3 * N_t * (456/64) (for Intel Xeon)

    (N=xMax=yMax=zMax; N_t: time steps)

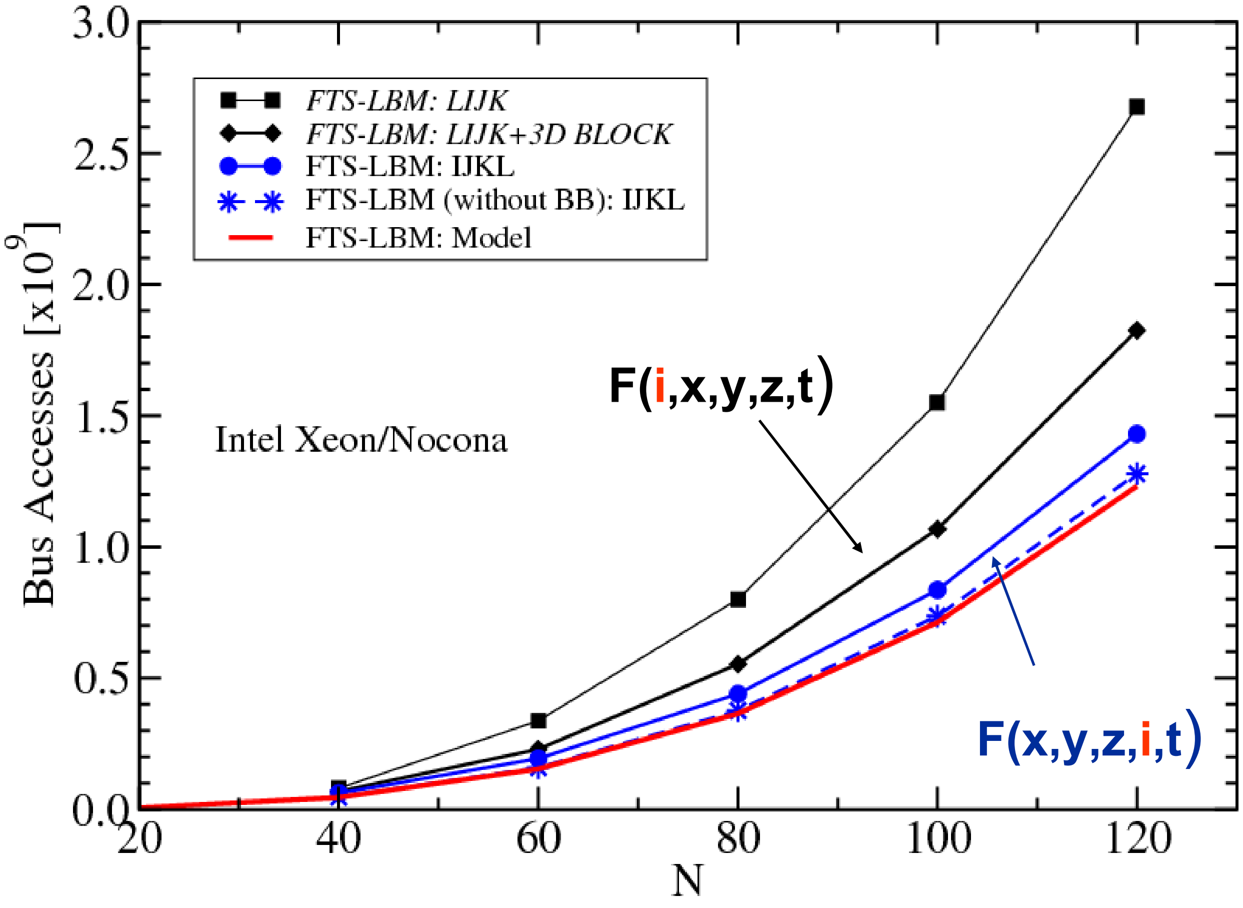
Optimal data access patterns: Data layout vs. spatial blocking

- **Spatial blocking & data layout for FTS-LBM:**
  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)` (Fortran ordering)
  - Implement spatial blocking for `F(i,x,y,z,t)`





Optimal data access patterns: Data layout vs. spatial blocking



All very good, but LBM is still limited by bus bandwidth even on a single core!

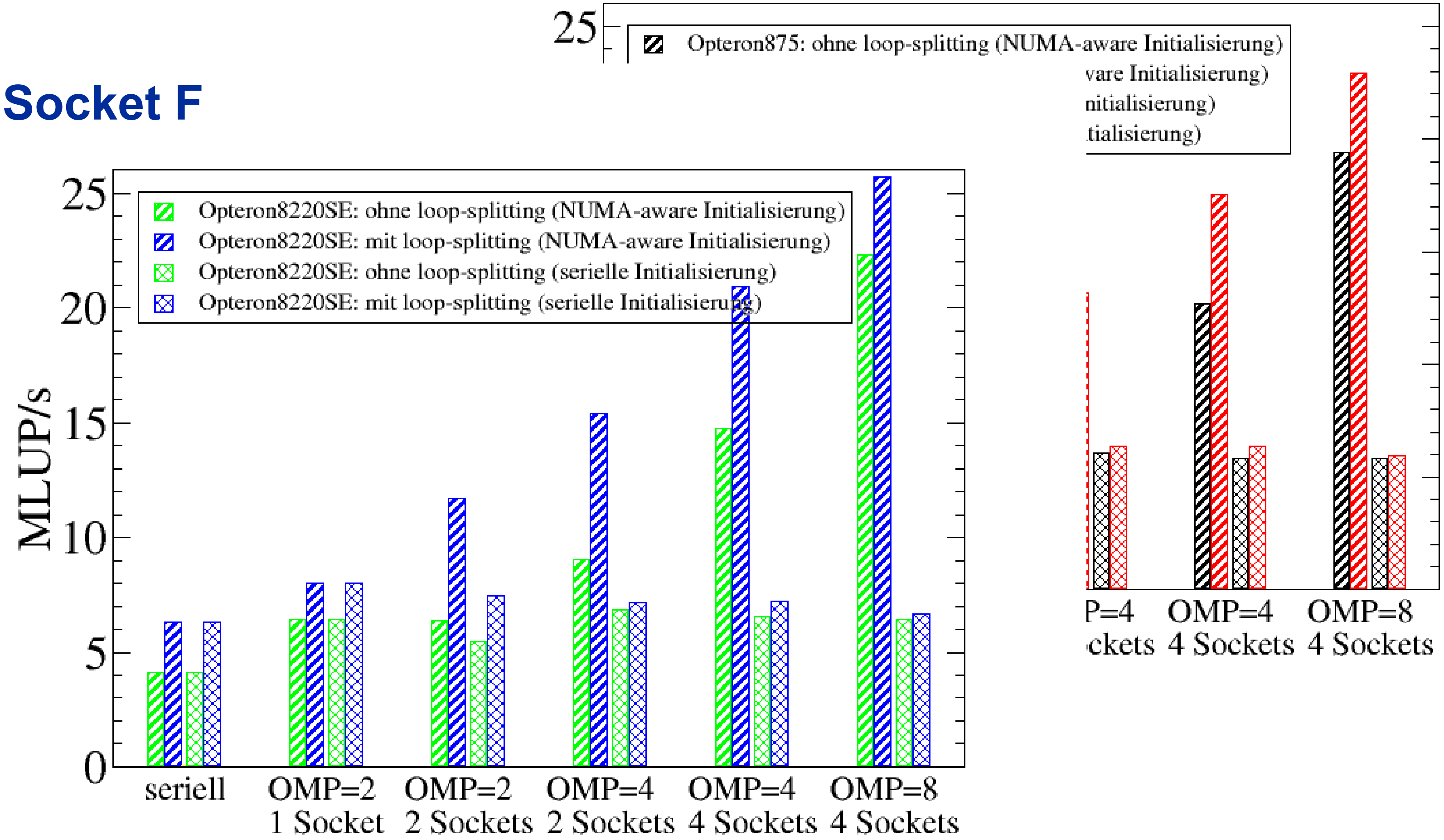
F(x,y,z,i,t) layout implies minimum data transfer!

Bounce back (BB) implementation introduces some overhead not included in the model

FTS-LBM on ccNUMA: Opteron 4-socket machines

Socket 940

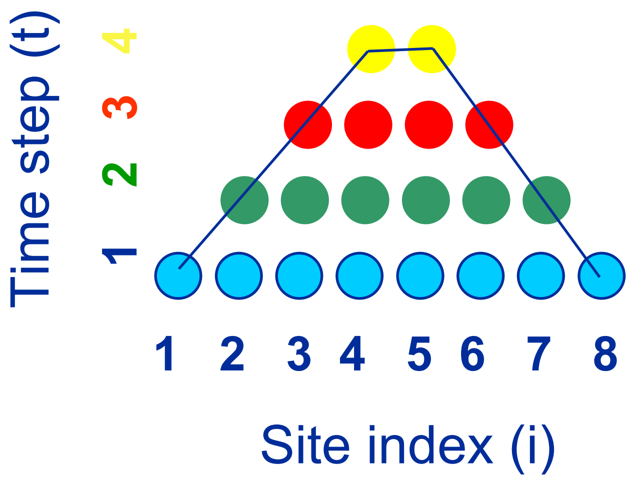
Socket F



Cache oblivious blocking approach
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 whole domain

Block of 8 sites of a long 1D chain

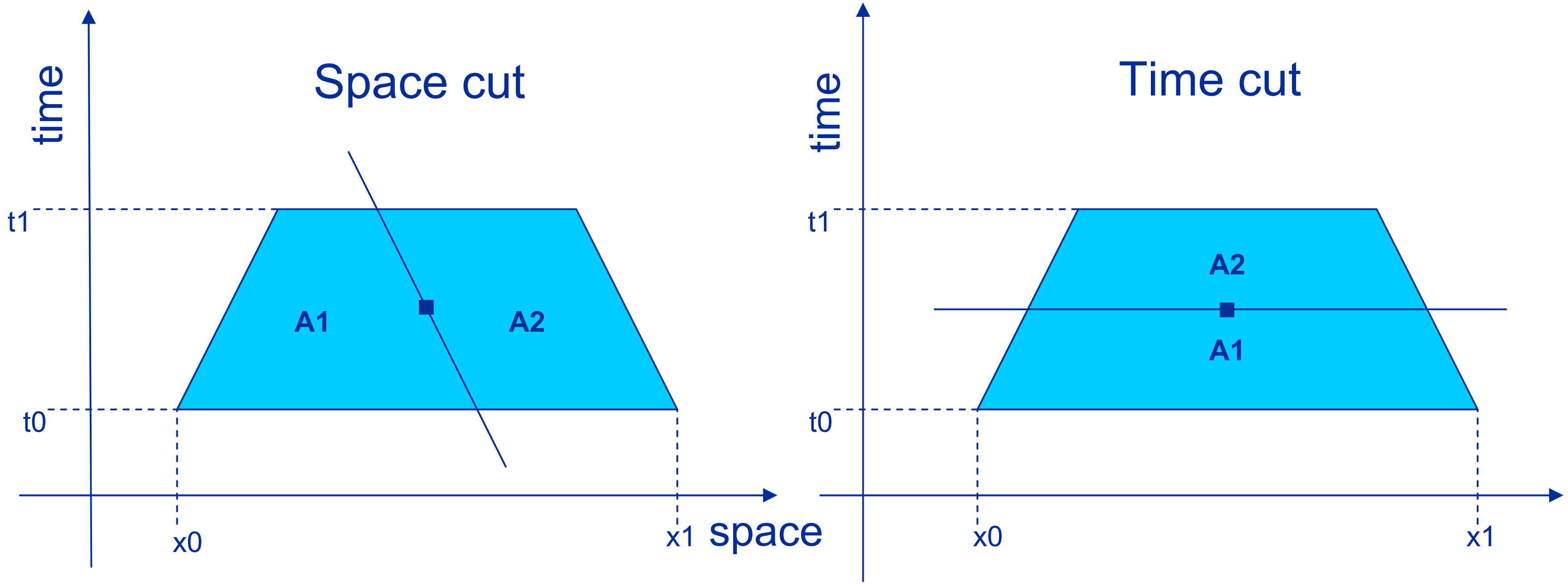


Cache-Oblivious Blocking Approach introduced by Frigo et al. for matrix transpose / FFT / sorting: M. Frigo, C.E. Leiserson, H. Prokop, and S. Ramachandran. *Cache-oblivious algorithms*. In *Proc. of the 40th IEEE Symp. on Foundations of Computer Science* (FOCS 99), p.285-297. 1999

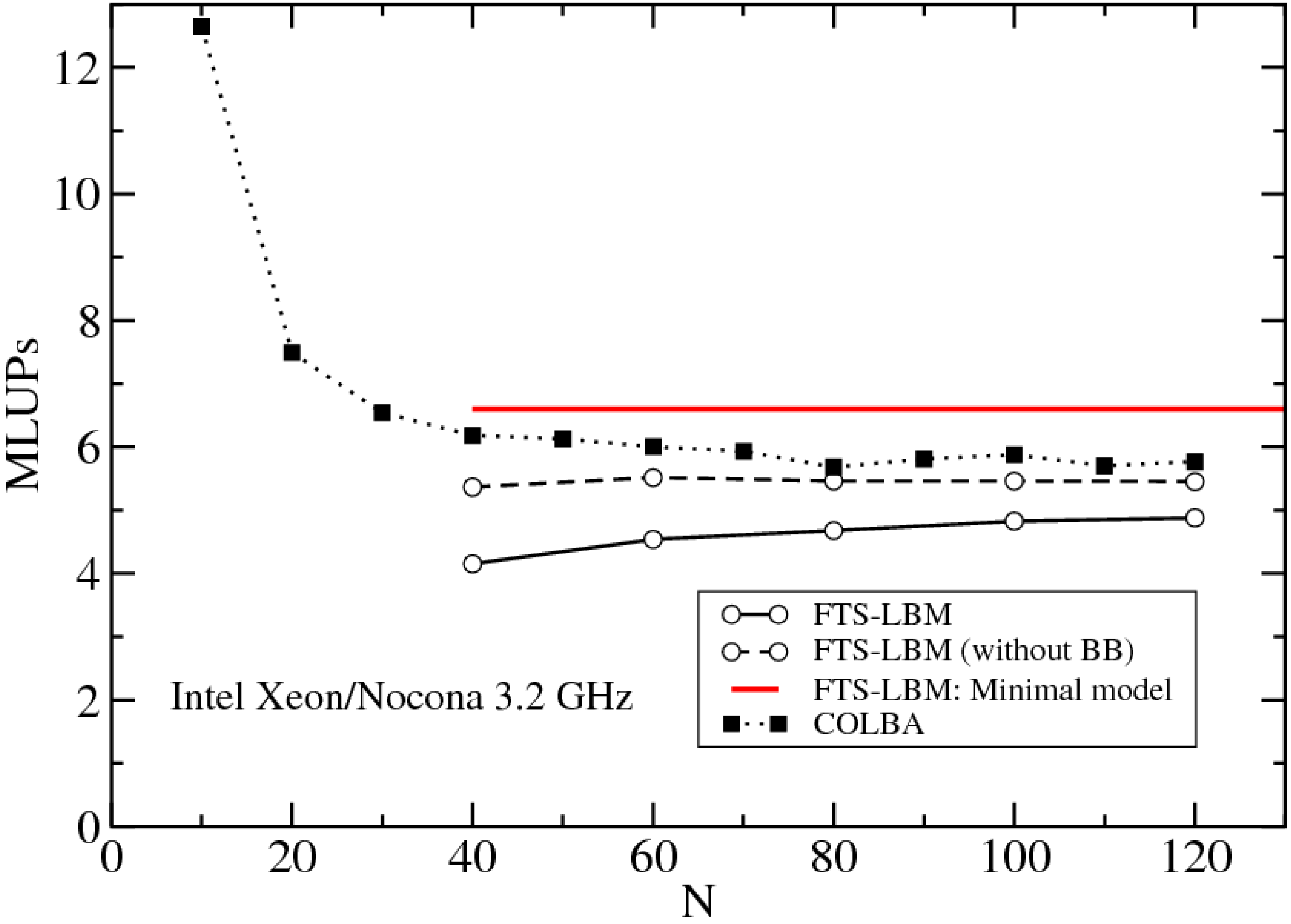
Cache oblivious blocking approach
Basic Idea

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

Example: 1 spatial dimension (1D)

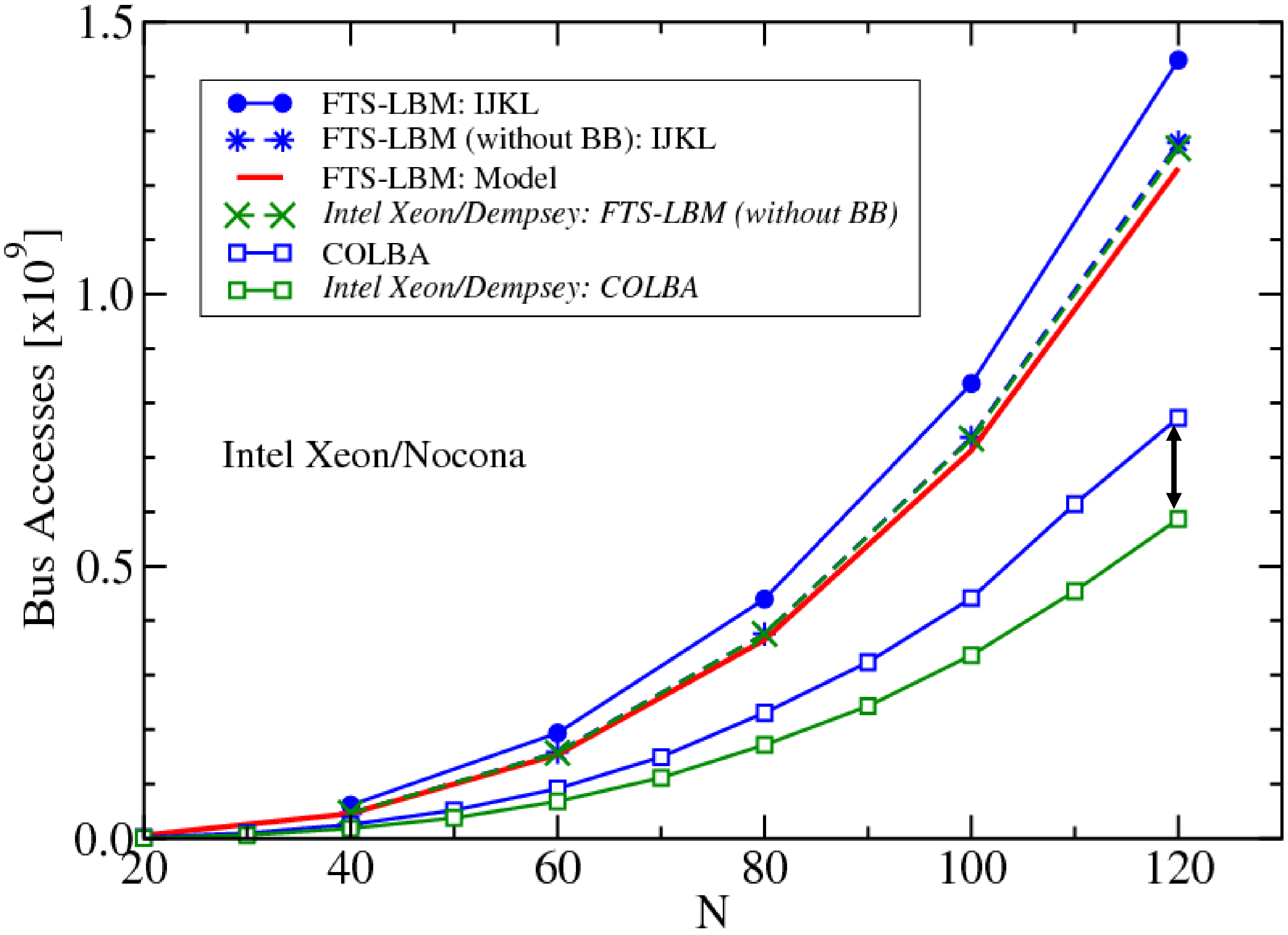


Cache oblivious blocking approach
Sequential performance



COLBA improves 1-thread performance only slightly…

Cache oblivious blocking approach
Memory bandwidth requirements



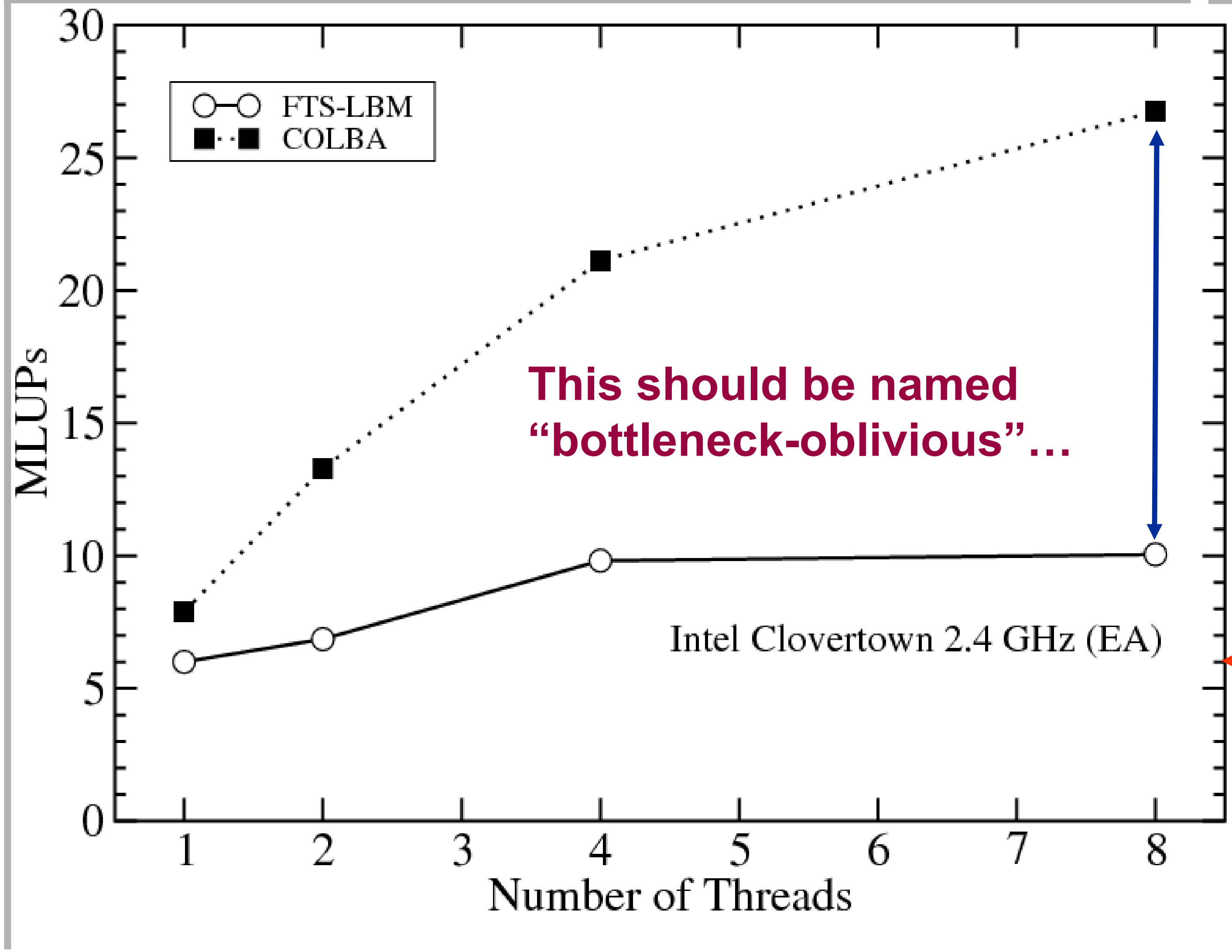
L2 size:

1 MB (Nocona)

2 MB (Dempsey)

Perfect cache model: BusAccesses (COLBA) ~ (Cache Size) $^{-1/3}$

Cache oblivious blocking approach
Cutting the cheese on quad core



This should be named "bottleneck-oblivious"…

Intel Xeon/Nocona

Conclusions
How to revive the Killer Micros

Where we are

- ccNUMA is here and getting stronger
- Architecture diversity will increase
- Placement & pinning are vital
- Multi-core is here
- “BlueGene effect” hard to see (yet)
  - but inevitable
- Special-purpose computing on the rise

What we need

- ccNUMA-aware compilers, users and OSs
- Well-adapted system environments, easy-to-use tools
- “Bottleneck-oblivious” algorithms
- Sharpen UPA (see next slide)
- Hiding complexities with libraries & compilers

Are they still attacking?

Unpopular Parallelism?

- Current micros are still too fast…
- User Parallelization Awareness must be boosted!
- Harbinger of doom: "Pleeeeease give me a queue with 8 cores and 10 days of runtime!"

HLRB2 (federal system) queue snapshot:

```
Queue            Run
--------------------
N1020              1
N2040              1
np_S64             1
special            1
N510               3
N256               4
N128               5
N64L               5
N64               90
               -----
                 111
```