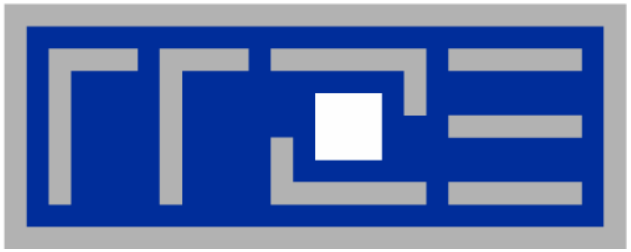


Sun UltraSPARC T2
First Tests

Georg Hager
Gerhard Wellein
Thomas Zeiser
Michael Meier

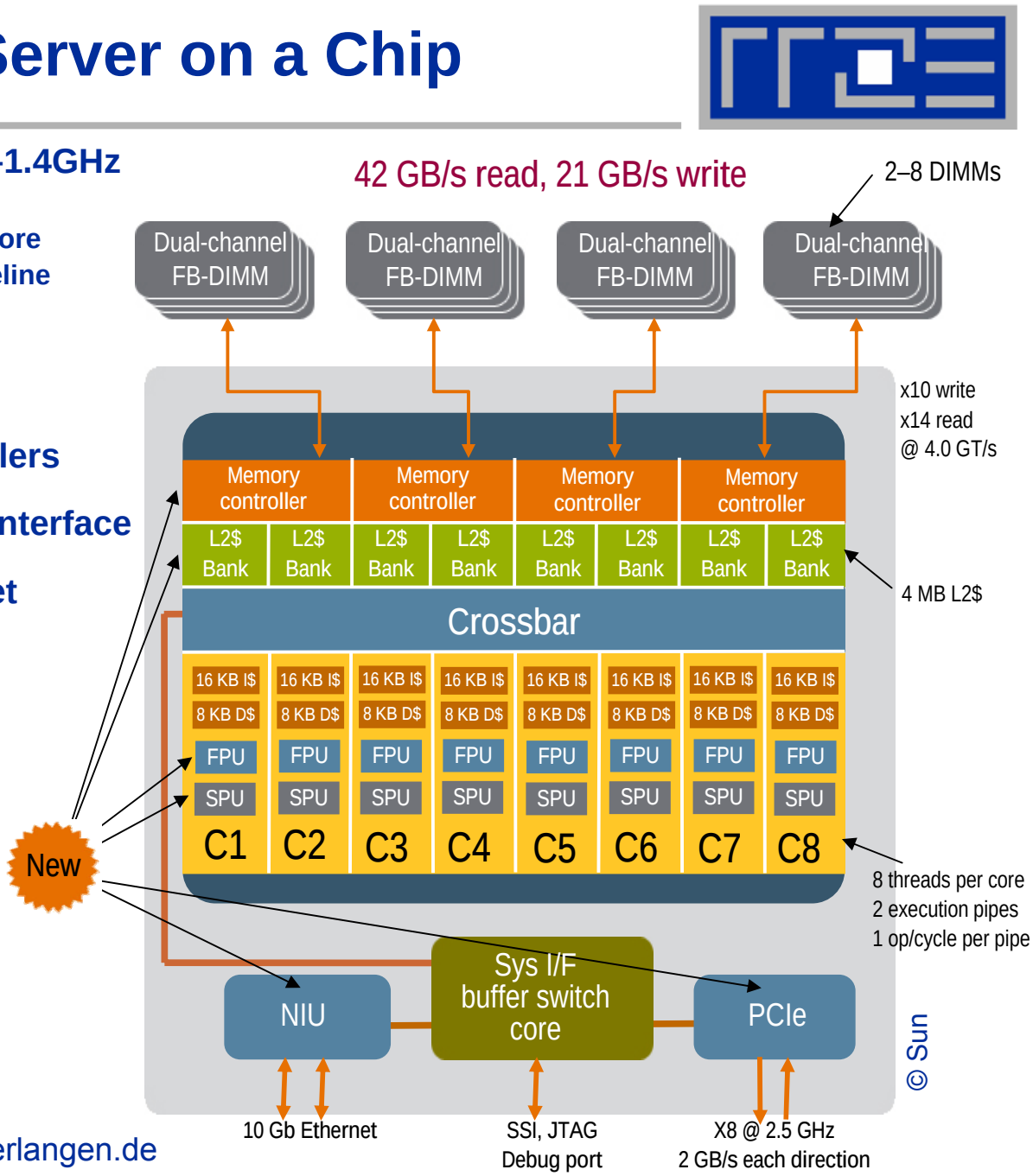
HPC Services, RRZE

SunDay RRZE, 2007/11/06

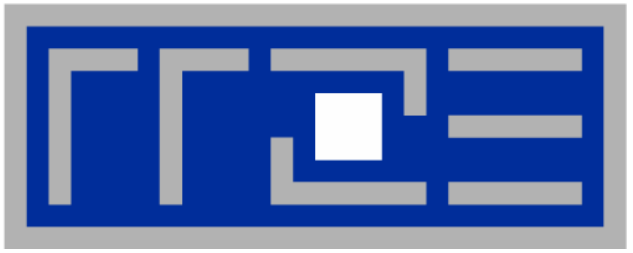
Outline

- Sun UltraSPARC T2 basics
- Lattice Boltzmann Method (LBM) benchmarks
- RRZE benchmark suite (very preliminary)
- STREAM performance numbers
  - How to get good data streaming
- Thanks to Sun and RWTH Aachen for granting access to EA T2 system

UltraSPARC T2: Server on a Chip



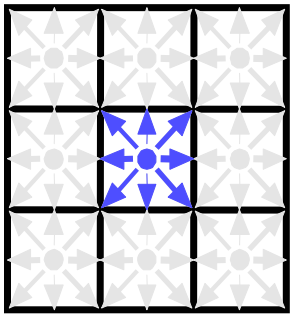
- 8 SPARC V9 cores @ 1.2–1.4GHz
  - 8 threads per core
  - 2 execution pipelines per core
  - 1 instruction/cycle per pipeline
  - 1 FPU per core
  - 1 SPU (crypto) per core
  - 4 MB, 16-way, 8-bank L2$
- 4 FB-DIMM DRAM controllers
- 2.5 GHz x 8 PCI-Express interface
- 2 x 10 Gb on-chip Ethernet
- Technology: TI 65nm
- Die size: 342mm^2
- Power: < 95 W (nominal)
- On-Chip Encryption:
  DES, 3DES, AES, RC4, SHA1, SHA256, MD5, RSA 2048, ECC, CRC32



Lattice Boltzmann Method

Evaluating single node performance with an OpenMP Kernel

The Lattice Boltzmann Method (LBM)



- Evolved from "cellular automata models"
- Physical basis: the Boltzmann equation

$$\frac{\partial f}{\partial t} + \boldsymbol{\xi}\frac{\partial f}{\partial \boldsymbol{x}} + \boldsymbol{K}\frac{\partial f}{\partial \boldsymbol{\xi}} = \mathcal{Q}(f,f)$$

1) approximation of the collision process by the BGK relaxation
2) physical discretisation → „velocity discrete Boltzmann equation"
3) numerical discretisation of spatial and temporal derivatives

- ⇒ explicit lattice Boltzmann equation

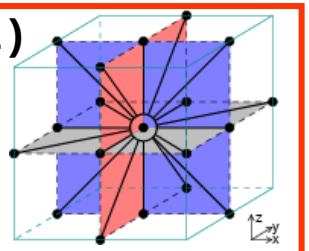
$$f_i(\boldsymbol{x}+\boldsymbol{c}_i,\ t+1) - f_i(\boldsymbol{x},\ t) = -\frac{1}{\tau}\left(f_i - f_i^{eq}\right)$$

- satisfies the incompressible Navier Stokes equations with 2nd order accuracy
- **numerical and computational advantages**

Optimal data access patterns:

Basic implementation strategy: LIJK layout


```
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

Stream Step

LD & ST
19 Cachelines

If cache line of store operation is not in cache it must be loaded first (RFO)!

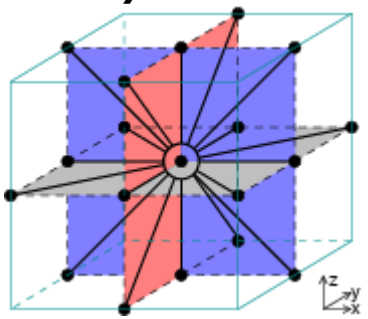
#load operations: **19***xMax*yMax*zMax + **19***xMax*yMax*zMax

#store operations: **19***xMax*yMax*zMax

Assuming full use of each cache line!

Optimal data access patterns:

Basic implementation strategy: IJKL layout & OpenMP



```
double precision f(0:xMax+1,0:yMax+1,0:zMax+1,0:18,0:1)
!$OMP PARALLEL DO PRIVATE(Y,X,…) SCHEDULE(RUNTIME)
do z=1,zMax
   do y=1,yMax
      do x=1,xMax
         if( fluidcell(x,y,z) ) then
```

Collide

```
            LOAD f(x,y,z,         0:18,t)
            Relaxation (complex computations)
```

Stream

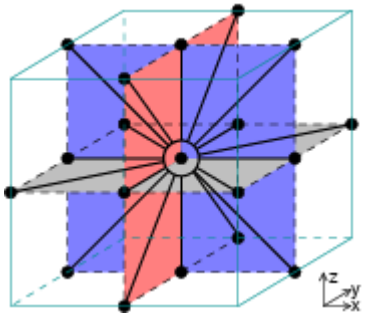
```
            SAVE f(x  ,y  ,z  ,  0,t+1)
            SAVE f(x+1,y+1,z  ,  1,t+1)

            …
            SAVE f(x  ,y-1,z-1, 18,t+1)
         endif
      enddo
   enddo
enddo
```

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

#store operations: **19***xMax*yMax*zMax

Optimal data access patterns:
Basic implementation strategy: VECTOR & OpenMP

```
double precision f(0:((xMax+1)*(yMax+1)*(0:zMax+1)),0:18,0:1)
!$OMP PARALLEL DO PRIVATE(…) SCHEDULE(RUNTIME)
do m=1,(zMax*yMax*xMax)
    if( fluidcell(m) ) then
       LOAD f(m, 0:18,t)
       Relaxation (complex computations)
       SAVE f(m  ,                    0,t+1)
       SAVE f(m-(xMax+2)-(xMax+2)*(yMax+2), 18,t+1)
    endif
enddo
```



Pitfall on ccNUMA systems: Initialization is still of IJKL style…

```
!$OMP PARALLEL DO PRIVATE(Y,X,…) SCHEDULE(RUNTIME)
do z=1,zMax
  do y=1,yMax; do x=1,xMax
              f(x  ,y  ,z  ,        0:18,t)=…
  enddo; enddo
enddo
```

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:
  - 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. performance: Million Lattice Site Updates per second

    MaxMLUPs=
    MemoryBandwidth [MByte/s] / (456 Bytes/cell update)
  - Good approximation: **MemoryBandwidth** = “STREAM TRIADS”
- *#Floating Point Operations varies: 150 – 200 per fluid cell update*
  5 MLUPs ⟷ 1 GFlop/s

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

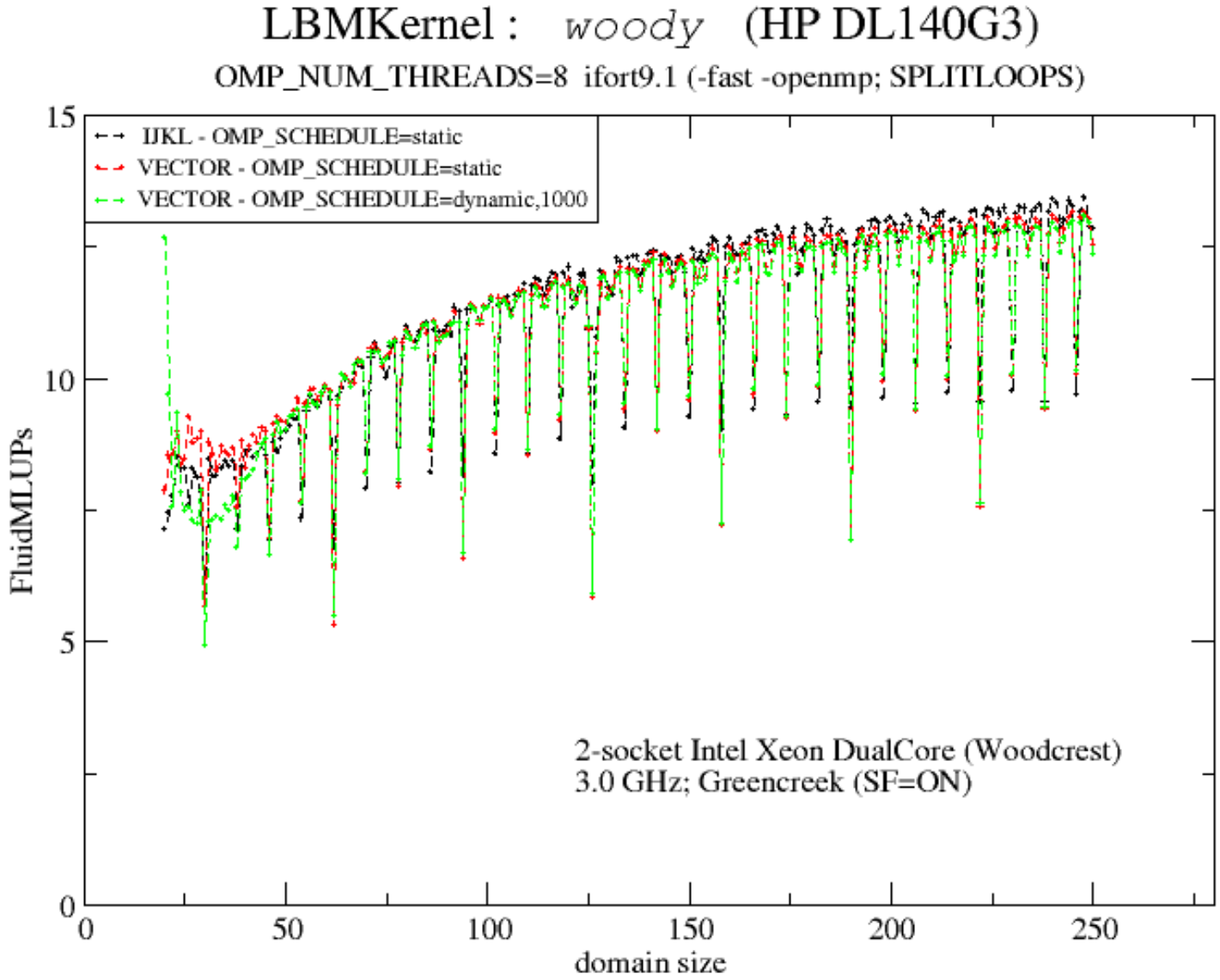
- So far we could not convince the Intel compiler to use "Non-Temporal stores"
- Single Core estimates

	Peak Perf. [GFlop/s]	Max. MLUPS	STREAM [MByte/s]	Max. MLUPS	**Max. Measure**
Intel Xeon/Nocona	6.4	32	3,000	**6.6**	**4.5-5.0**
Intel Xeon/Woodcrest	12.0	60	4,200	**9.2**	**7.0-8.0**
AMD Opteron875	4.4	22	3,500	**7.7**	**4.5-5.0**
Intel Montecito	6.4	32	6,100	**13.4**	**9.0-10.0**
NEC SX8	16	**80**	~60,000	**~200**	**~65**

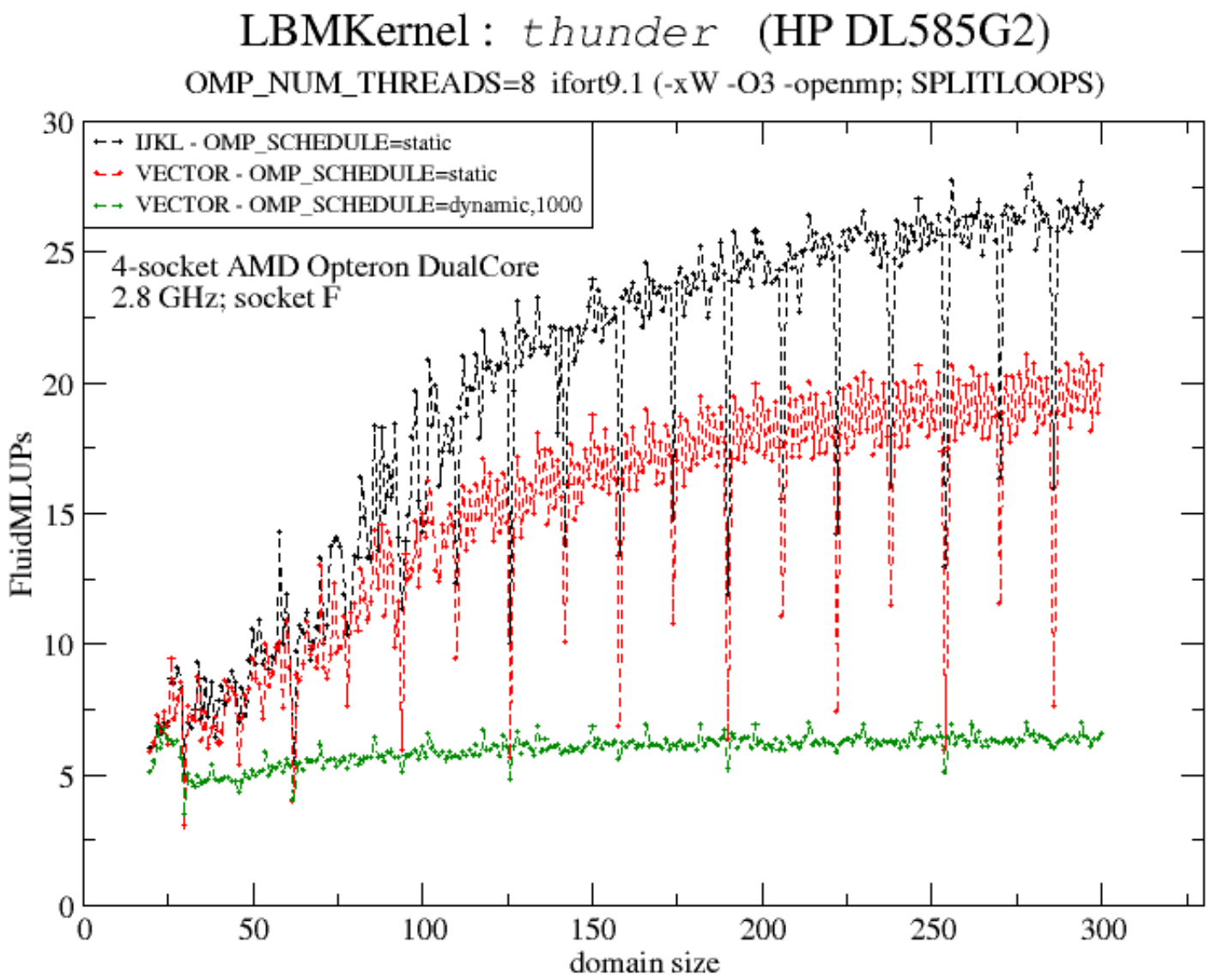
Simple model provides a good approximation

Vector-performance not limited by memory bandwidth!

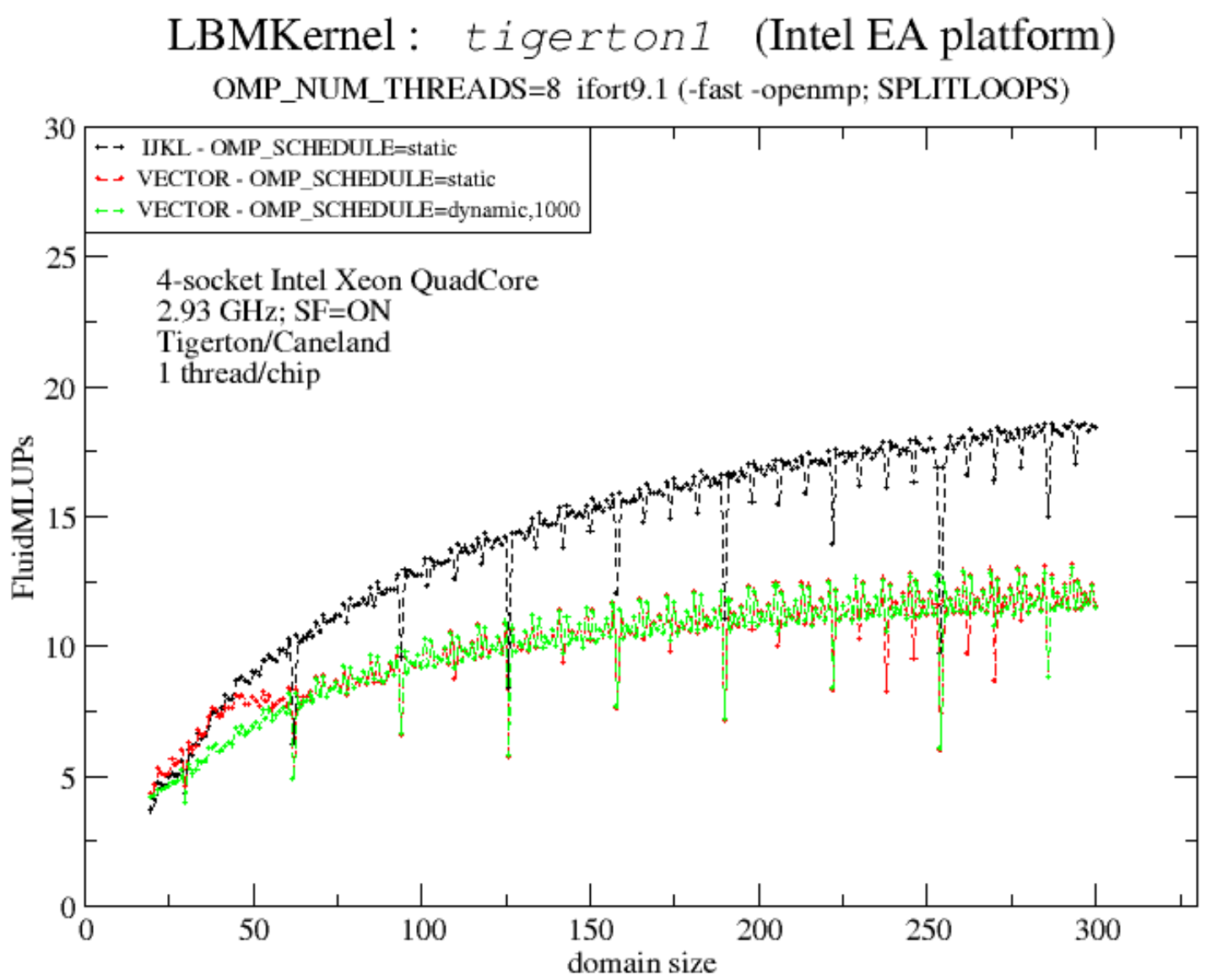
LBM
Performance – Single node: 2-socket Woodcrest



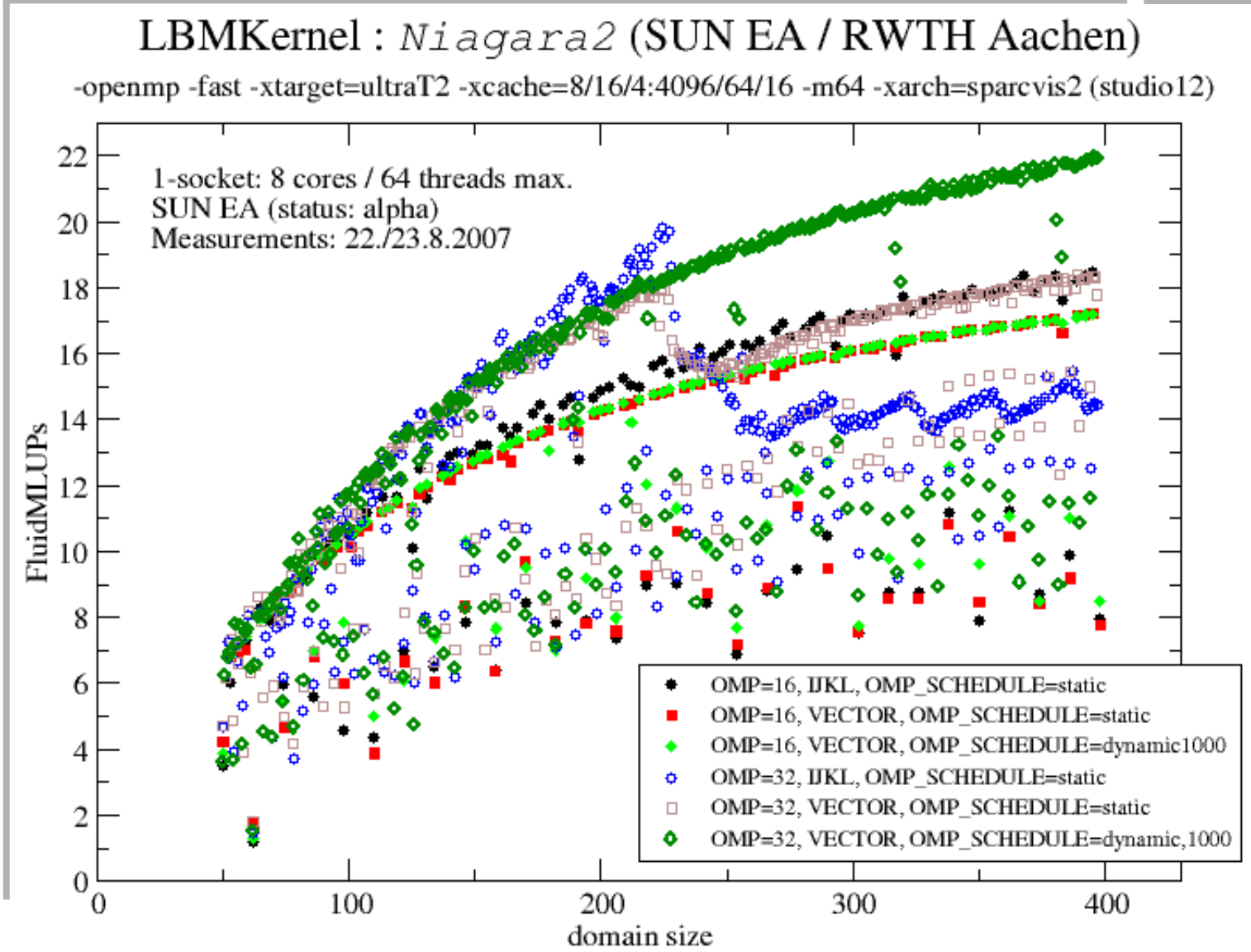
LBM
Performance – Single node: 4-socket F Opteron DC



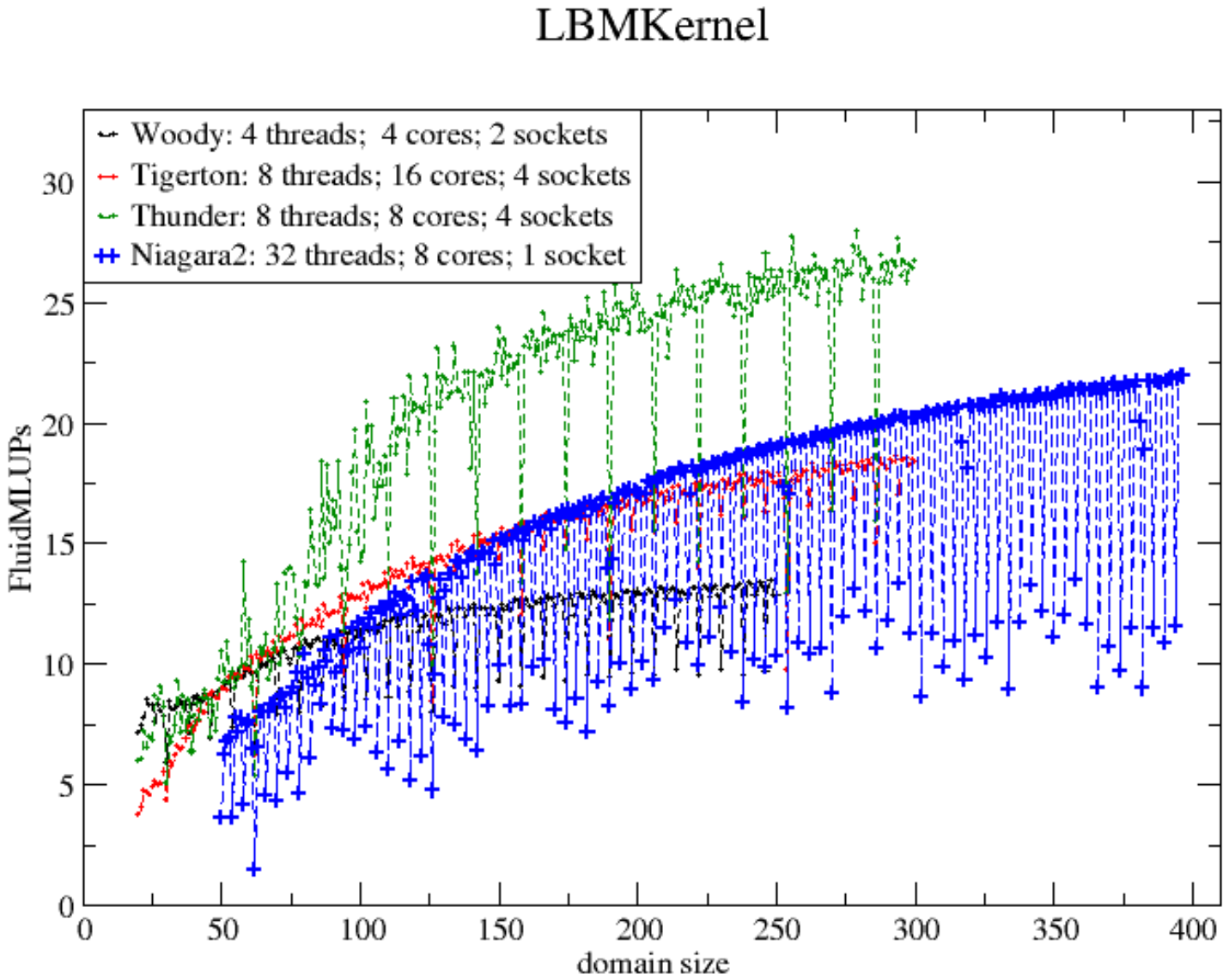
LBM
Performance – Single node: 4-socket Tigerton



LBM
Performance – Single node: 1-socket Niagara2



LBM
Performance – Compare them all



RRZE Benchmark Suite

- **Standardized, easy-to-use benchmark suite for use in evaluations, procurements etc.**
- **Several benchmark codes (applications and low-level) under a common Makefile construct**
- **Customizing one single include file and typing “make” does the trick**
- **Performance data collected as simple numbers (higher is better) with timestamps for easy reference**

- **Benchmarks used in previous procurement:**

  AMBER, EXX, IMD, OAK3D, TRATS, PIO, TRIAD

- **Additional benchmarks in the suite: Kette, DMRG**

Benchmark results
RRZE Benchmark Suite: EXX & AMBER8

- **EXX**
  - Quantum Chemistry package developed at Theoretical Chemistry, U Erlangen
  - Calculation of structural and electronic properties of periodic systems (solids, slabs, wires)
  - Using (time-dependent) Density Functional Theory (DFT)
  - Performance dominated by FFT operations (FFTW)
  - Largely cache-bound
  - Fortran 90 and MPI
- **AMBER8 (pmemd)**
  - Widely used commercial MD package
  - Distributed-memory FFT and force field calculations
  - Benchmark case (HCD): HPr:CCpa Tetramer dynamics
  - Largely cache-bound
  - Fortran 77 and MPI

Benchmark results
RRZE Benchmark Suite

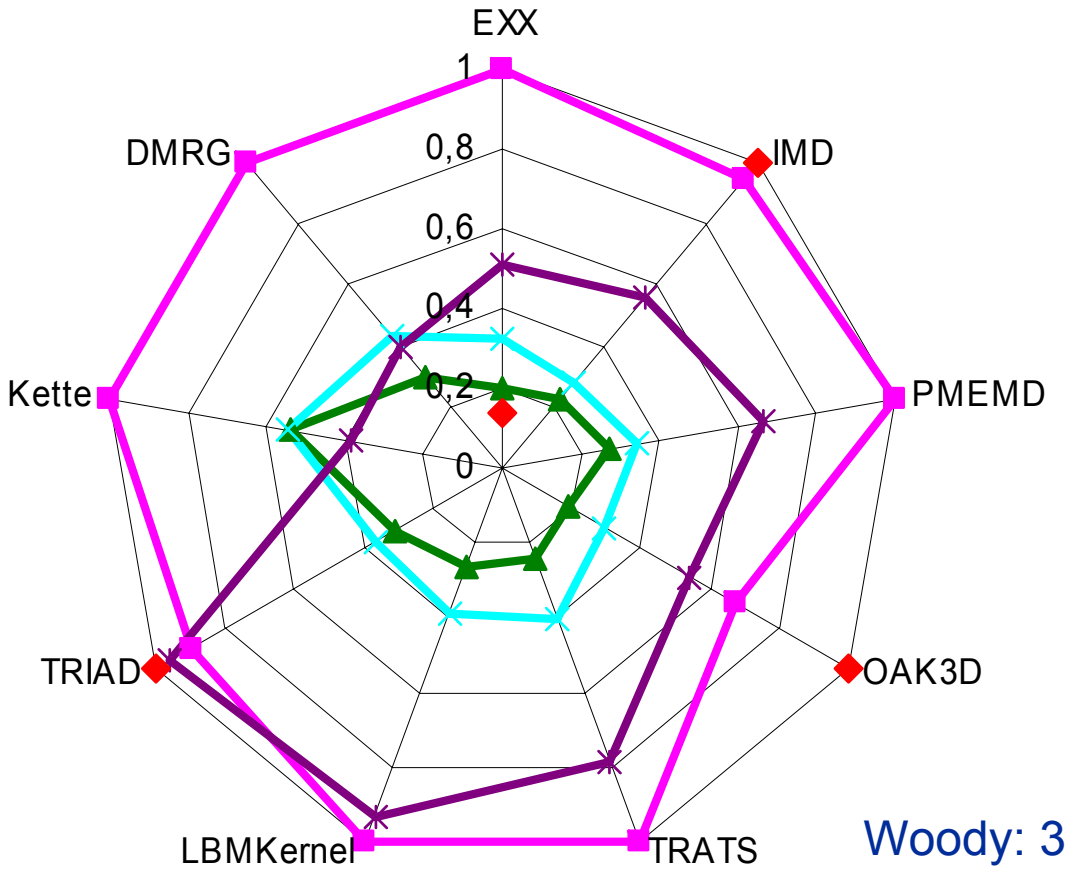
- **IMD**
  - Molecular dynamics package developed at U Stuttgart
  - Used at FAU for calculation of defect dynamics in solids
  - Weakly dependent on memory bandwidth
  - C and MPI

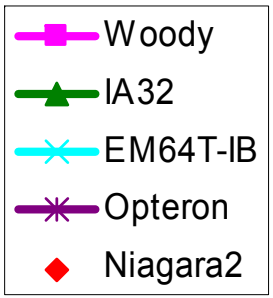
- **TRATS**
  - Production Lattice-Boltzmann CFD code developed at Institute for Fluid Dynamics, U Erlangen
  - Memory-bound on standard microcomputers, compute-bound on NEC SX (code balance: ≈ 0.4 Words/Flop)
  - Fortran 90 and MPI
  - Also known as BEST
  - RRZE built a small kernel to evaluate optimization approaches & single CPU/node characteristics (LBMkernel)

RRZE Benchmark Suite

- **OAK3D**
  - Physics code developed at Theoretical Physics, U Erlangen
  - Simulates the dynamics of exotic (superheavy) nuclei via time-dependet Hartree-Fock (TDHF)
  - Photoabsorption, e^- capture, fusion, fission
  - Taylor expansion of time evolution and predictor-corrector step
  - Uses FFT for calculating derivatives
  - Performance dominated by small-size FFTs and dense matrix-vector operations
  - Some memory bandwidth required, benefits from large caches
  - Fortran 90 and MPI

1-node shootout RRZE Benchmark suite





Woody: 3GHz HP DL140G3 (4C)

For Reference: Memory bandwidth
Optimized version of STREAM: Tigerton (Intel EA)

4 sockets (SF=ON) QuadCore@2.93 GHz	COPY [MB/s]	SCALE [MB/s]	ADD [MB/s]	TRIAD [MB/s]
1 thread (1 Chip; 1 FSB)	2753	2798	2650	2641
4 threads (2 Chips; 1 FSB)	2855	2855	2680	2680
4 threads (4 Chips; 2 FSB)	5656	5678	5360	5350
4 threads (4 Chips; 4 FSB)	8345	8442	10260	10264
8 threads (8 chips; 4 FSB)	8686	8601	10735	10715
AMD Opteron DC 2.8 GHz 4 sockets / DDR667 / 8 threads				
Best out of 10	17035	17154	17591	17625
Worst out of 10	14982	15088	15363	15379

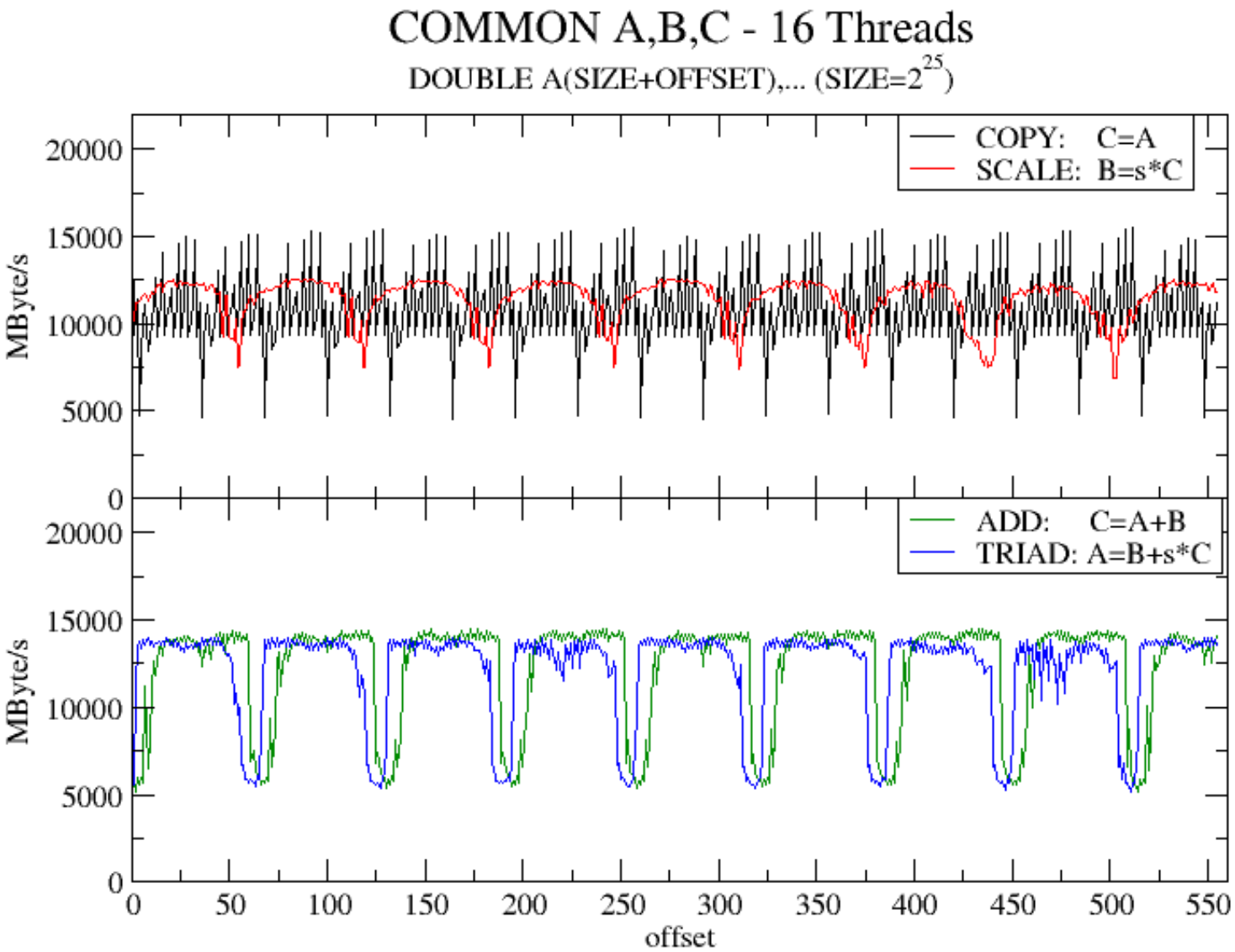
Memory bandwidth
Stream: Niagara2 (SUN EA – status: alpha)

8 cores (ARRAY=32*1024*1024) COMMON=ON	COPY [MB/s]	SCALE [MB/s]	ADD [MB/s]	TRIAD [MB/s]
16 threads (pinned) (offset=0)	13818	9681	5001	5158
16 threads (pinned) (offset=1)	9302	8612	4784	5134
16 threads (pinned) (offset=512)	13124	9629	5267	5237
16 threads (pinned) (offset=512+16)	12375	10383	12400	12129

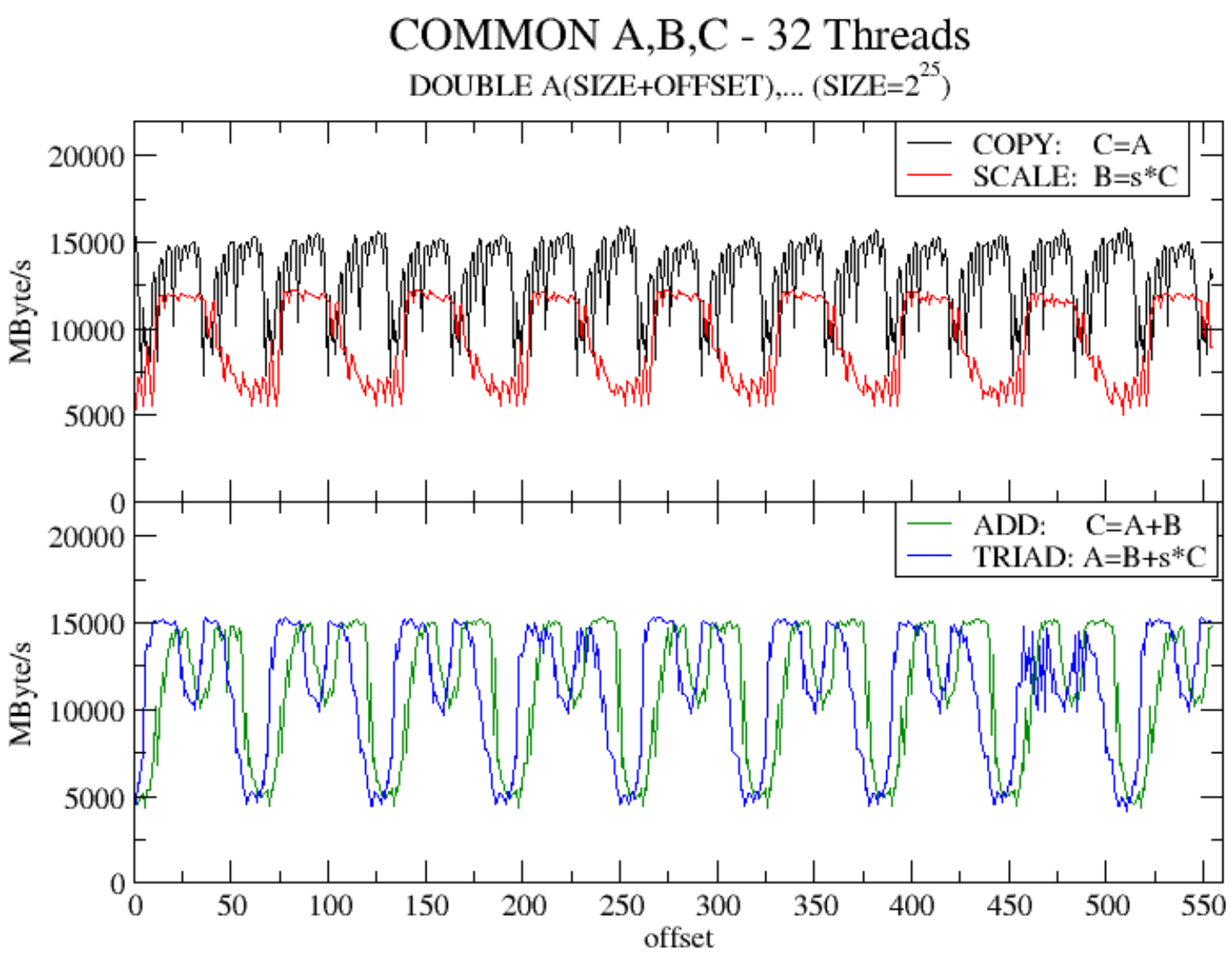
Observations

- Physical addresses mapping to memory controller uncontrollable?
- RFO on write misses → "effective bandwidth" increases by 3/2 (4/3) for COPY & SCALE (ADD & TRIAD)

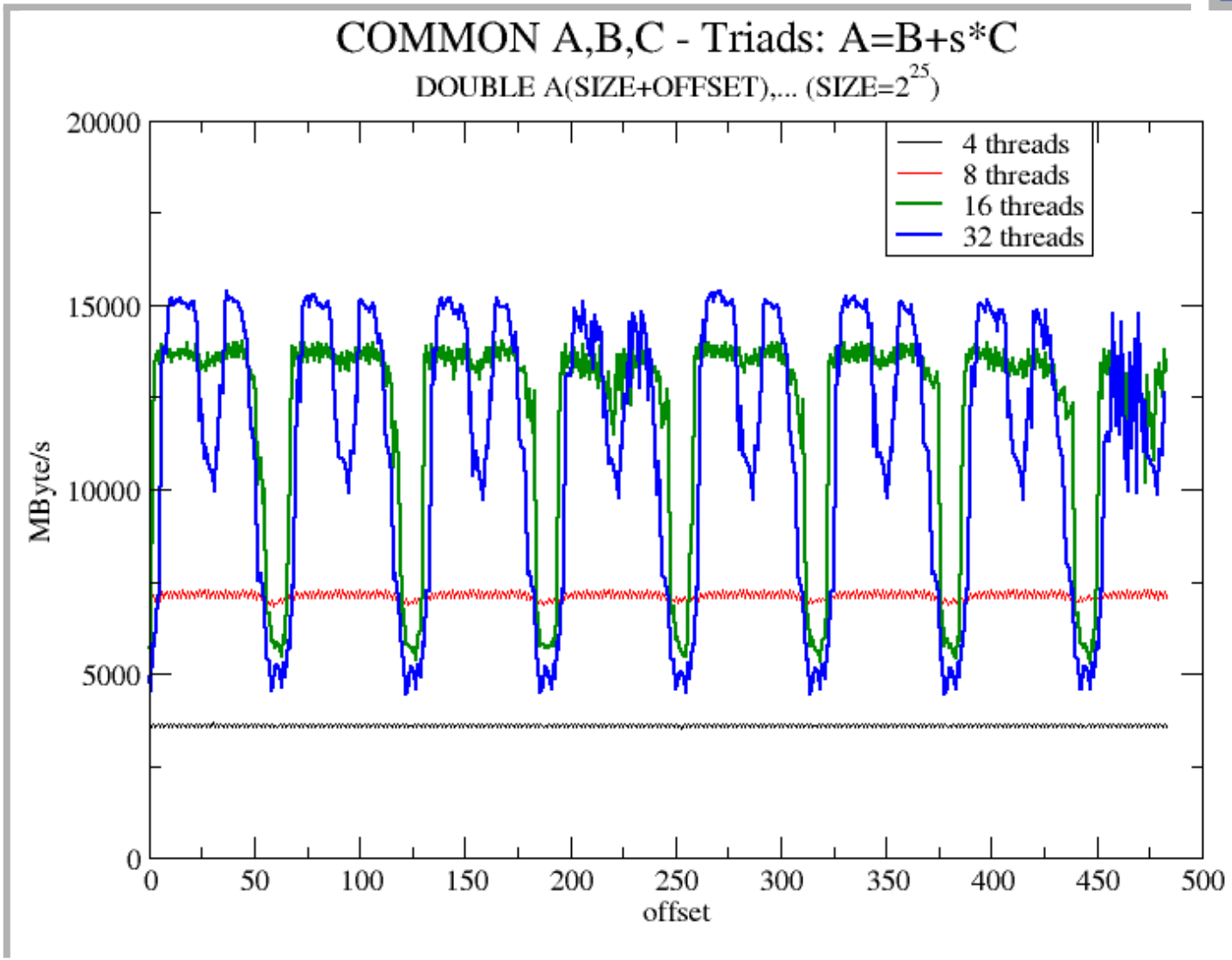
Memory bandwidth – stream
Niagara2 (SUN EA – status: alpha; SW upgrade)



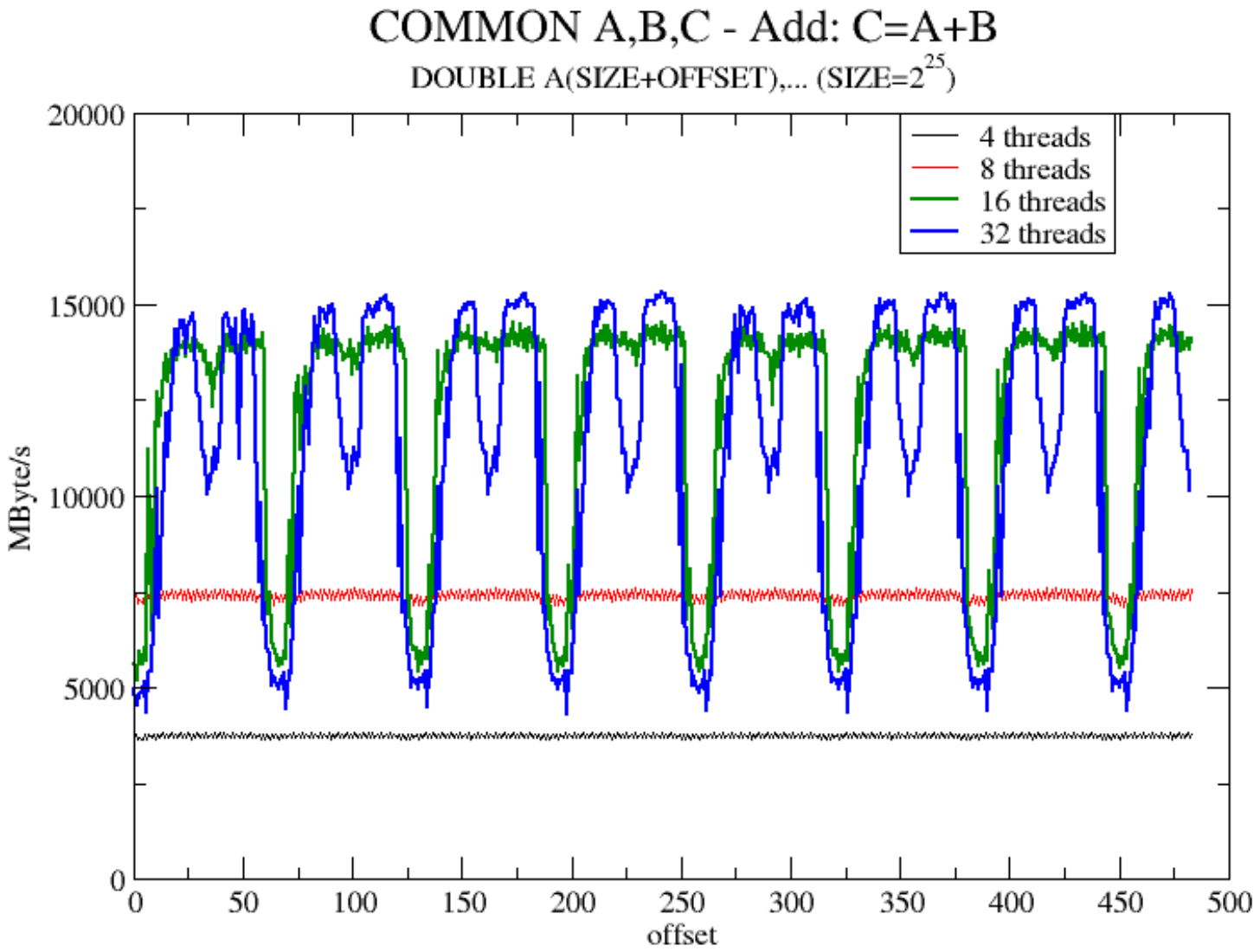
Memory bandwidth – stream
Niagara2 (SUN EA – status: alpha; SW upgrade)



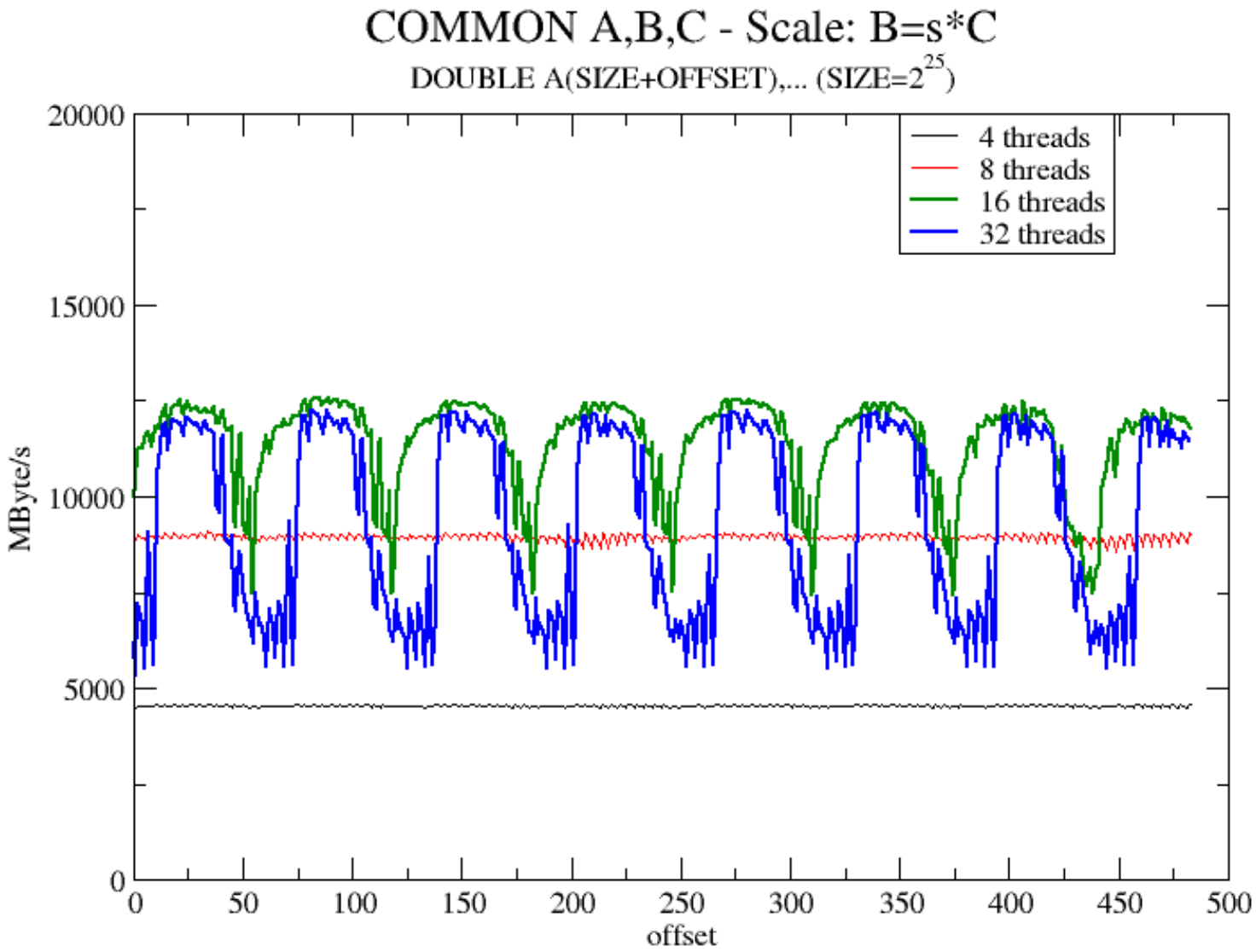
Memory bandwidth – stream
Niagara2 (SUN EA – status: alpha; SW upgrade)



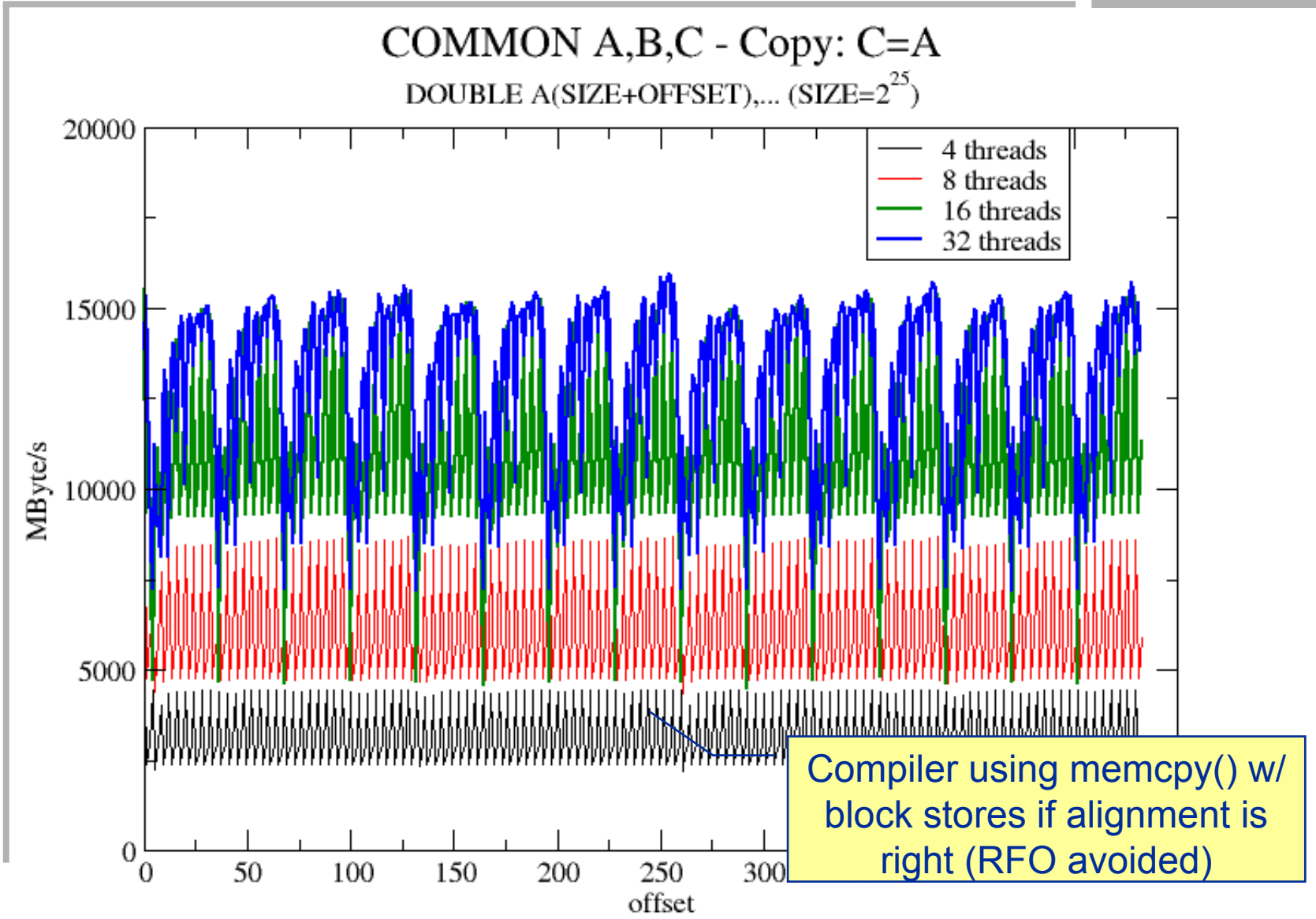
Memory bandwidth – stream
Niagara2 (SUN EA – status: alpha; SW upgrade)



Memory bandwidth – stream
Niagara2 (SUN EA – status: alpha; SW upgrade)



Memory bandwidth – stream
Niagara2 (SUN EA – status: alpha; SW upgrade)



Compiler using memcpy() w/ block stores if alignment is right (RFO avoided)

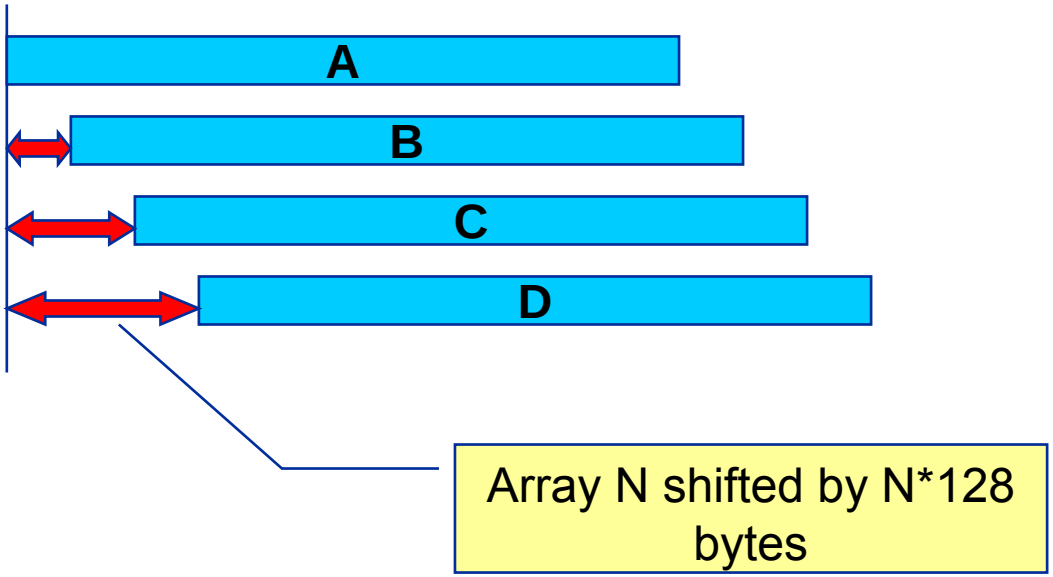
Vector triad A(1:N)=B(:) +C(:)*D(:)

- Observation
  - Mutual alignment of arrays has high impact on performance
  - Slight change in N can have big effect

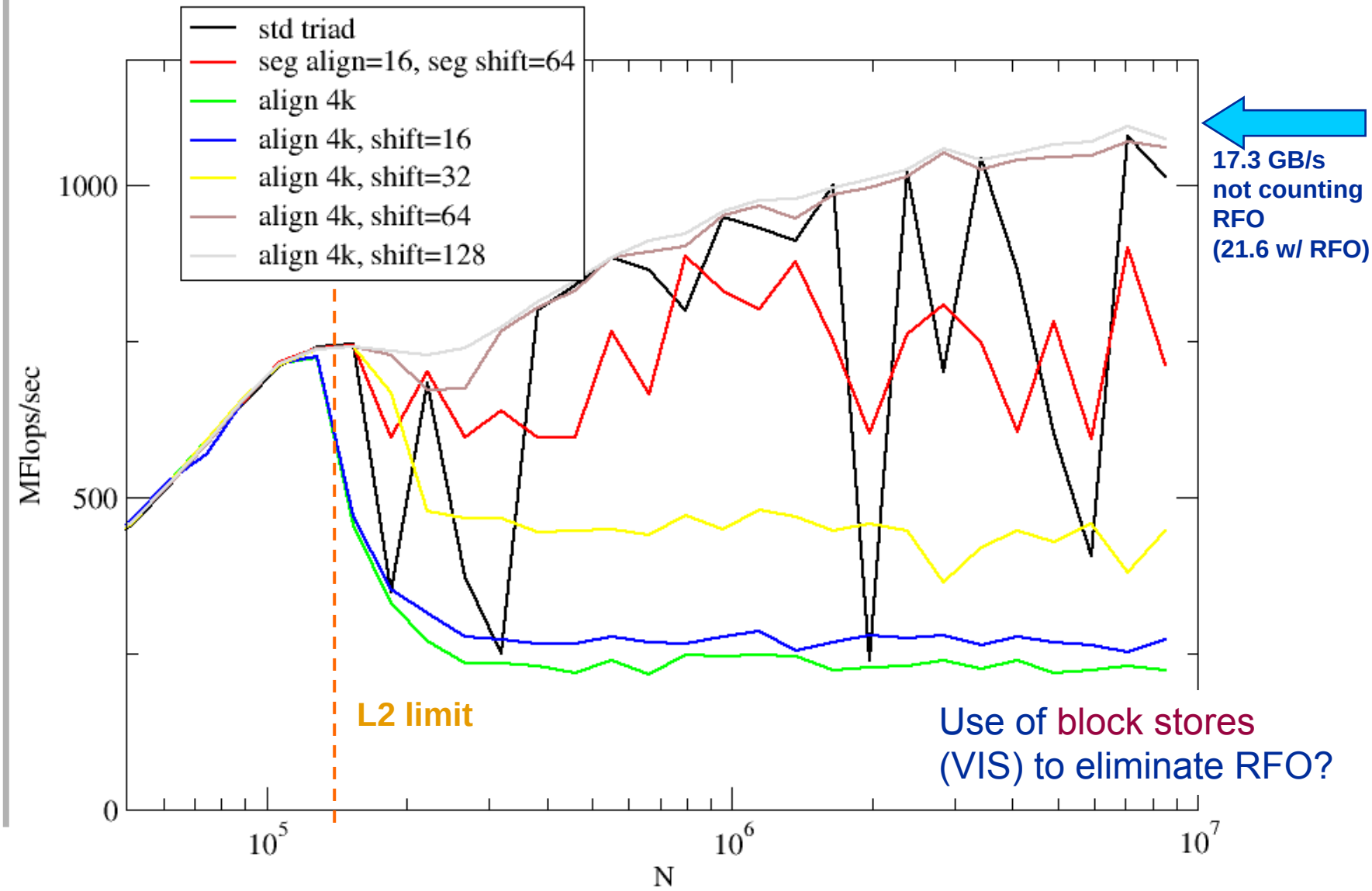
- How can access patterns be optimized independent of N?
  - Alignment of array start addresses to page boundaries is bad
    - Severe conflicts
  - Different OpenMP chunks (“segments”) can be aligned and shifted vs. alignment boundary
    - ameliorates bottlenecks, but still erratic numbers

Vector triad A(1:N)=B(:) +C(:)*D(:)

- Arrays A, B, C, D can be aligned to page boundaries and shifted w.r.t each other
  - Address bits 8:7 determine mapping to memory controller
  - Bit 6 chooses L2 bank per controller
  - Best result for shift = 128 (see next slide)



Schönauer vector triad A(1:N)=B(:) +C(:)*D(:)
32 threads



Conclusions

- Interesting massively threaded architecture
- Watch data alignment and aliasing issues
- Sensible measure for HPC: Performance per box per $