

# **Microbenchmarking for architectural exploration (and more)**

**Probing of the memory hierarchy**

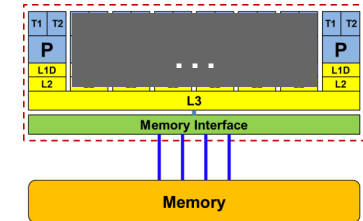
**Saturation effects in cache and memory**

**Typical OpenMP overheads**

# Intel Xeon E5 multicore processors



Microarchitecture	SandyBridge-EP	IvyBridge-EP	Haswell-EP
Shorthand	SNB	IVB	HSW
Xeon Model	E5-2680	E5-2690 v2	E5-2695 v3
Year	03/2012	09/2013	09/2014
Clock speed (fixed)	2.7 GHz	2.2 GHz	2.3 GHz
Cores/Threads	8/16	10/20	14/28
Load/Store throughput per cycle			
AVX(2)	1 LD & 1/2 ST	1 LD & 1/2 ST	2 LD & 1 ST
SSE/scalar	2 LD    1 LD & 1 ST	2 LD    1 LD & 1 ST	2 LD & 1 ST
L1 port width	2×16+1×16 B	2×16+1×16 B	2×32+1×32 B
ADD throughput	1 / cy	1 / cy	1 / cy
MUL throughput	1 / cy	1 / cy	2 / cy
FMA throughput	n/a	n/a	2 / cy
L2-L1 data bus	32 B	32 B	64 B
L3-L2 data bus	32 B	32 B	32 B
LLC size	20 MiB	25 MiB	35 MiB
Main memory	4×DDR3-1600	4×DDR3-1866	4×DDR4-2133
Peak memory BW	51.2 GB/s	51.2 GB/s	68.3 GB/s
Load-only BW	43.6 GB/s (85%)	46.1 GB/s (90%)	60.6 GB/s (89%)
$T_{L3Mem}$ per CL	3.96 cy	3.05 cy	2.43 cy



FP instructions throughput per core

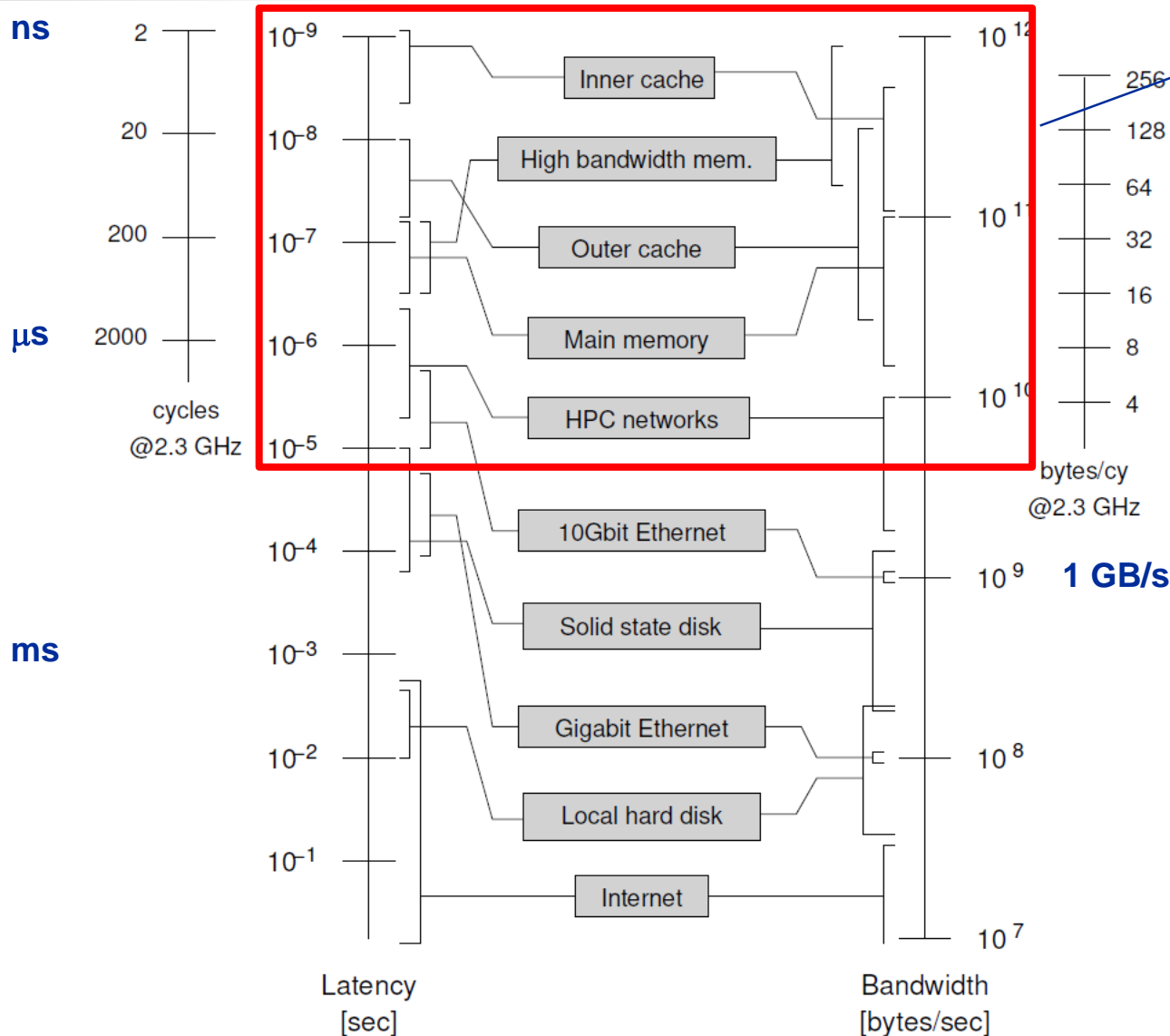
Max. data transfer per cycle between caches

Peak main memory bandwidth



- **Isolate small kernels to:**
  - Separate influences
  - Determine specific machine capabilities (light speed)
  - Gain experience about software/hardware interaction
  - Determine programming model overhead
  - ...
  
- **Possibilities:**
  - Readymade benchmark collections (epcc OpenMP, IMB)
  - STREAM benchmark for memory bandwidth
  - Implement own benchmarks (difficult and error prone)
  - **likwid-bench** tool: Offers collection of benchmarks and framework for rapid development of assembly code kernels

# Latency and bandwidth in modern computer environments



**HPC plays here**

**Avoiding slow data paths is the key to most performance optimizations!**

# The parallel vector triad benchmark

A “swiss army knife” for microbenchmarking

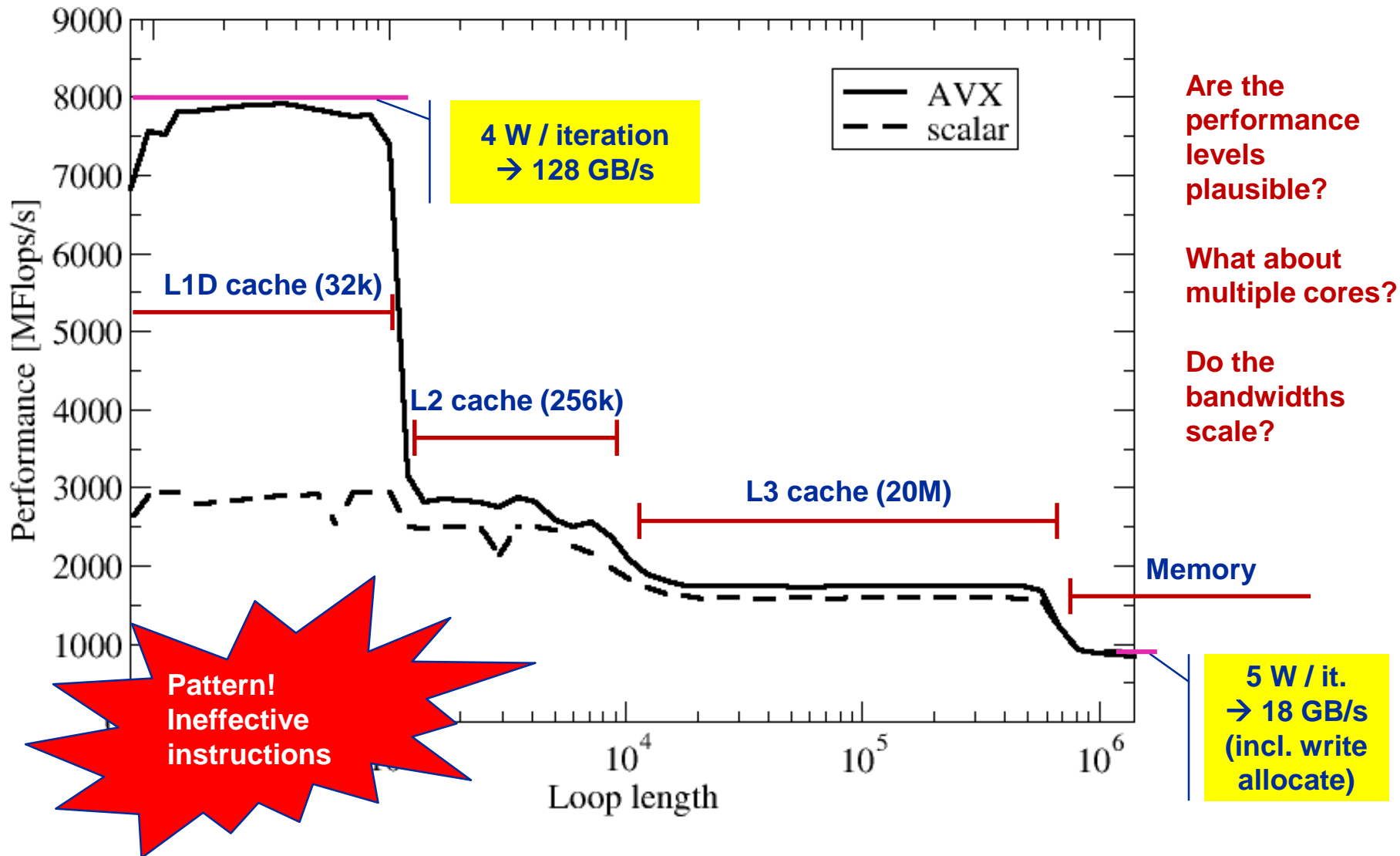


```
double precision, dimension(N) :: A,B,C,D
A=1.d0; B=A; C=A; D=A
stime = timestamp()
do j=1,NITER
  do i=1,N
    A(i) = B(i) + C(i) * D(i)
  enddo
  if(.something.that.is.never.true.) then
    call dummy(A,B,C,D)
  endif
enddo
etime = timestamp()
Mflops = (2.d0*NITER)*N / (etime-stime) / 1.d06
```

Prevents smarty-pants compilers from doing “clever” stuff

- Report performance for different N, choose NITER so that accurate time measurement is possible
- This kernel is limited by data transfer performance for all memory levels on all architectures, ever!

# A (:) = B (:) + C (:) \* D (:) on one Sandy Bridge core (3 GHz)



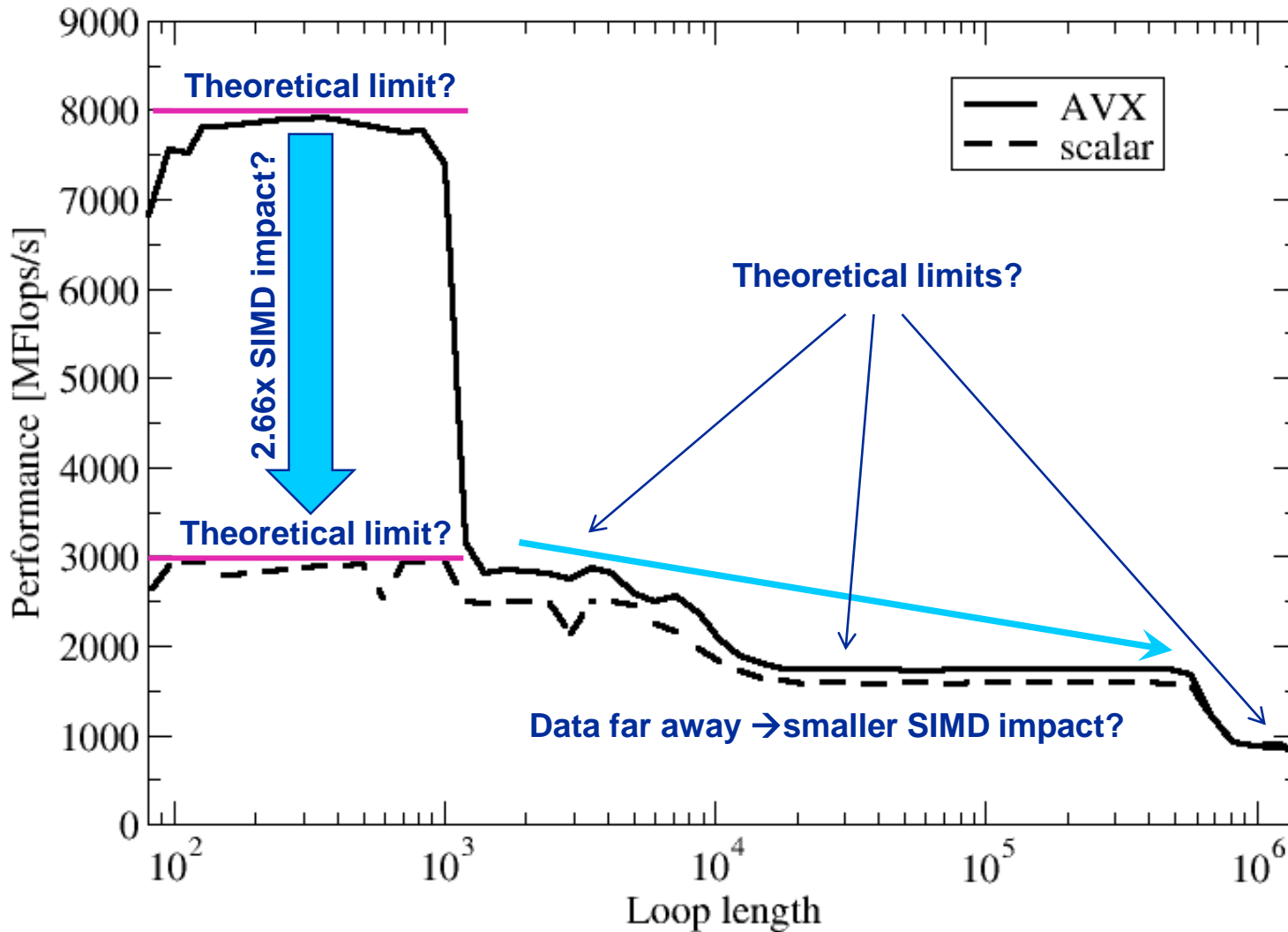
Are the performance levels plausible?

What about multiple cores?

Do the bandwidths scale?

# $A(:) = B(:) + C(:) * D(:)$ on one Sandy Bridge core (3 GHz):

Observations and further questions



See later for answers!



Every core runs its own, independent triad benchmark

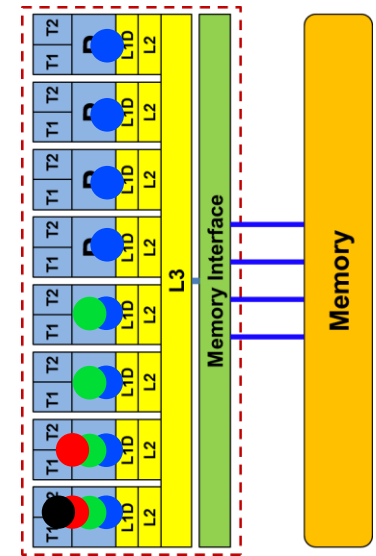
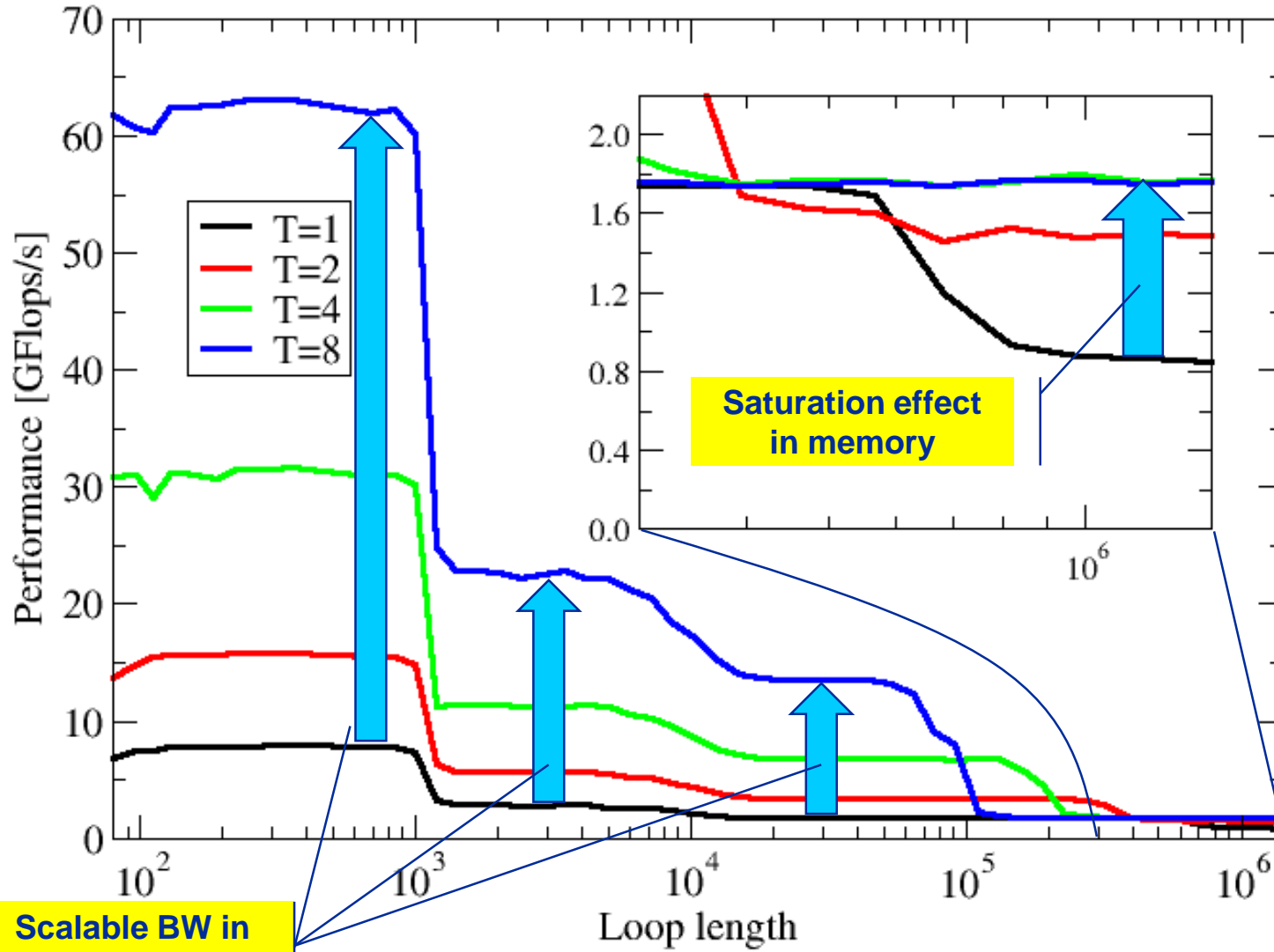
```
double precision, dimension(:), allocatable :: A,B,C,D

!$OMP PARALLEL private(i,j,A,B,C,D)
allocate(A(1:N),B(1:N),C(1:N),D(1:N))
A=1.d0; B=A; C=A; D=A
do j=1,NITER
  do i=1,N
    A(i) = B(i) + C(i) * D(i)
  enddo
  if(.something.that.is.never.true.) then
    call dummy(A,B,C,D)
  endif
enddo
!$OMP END PARALLEL
```

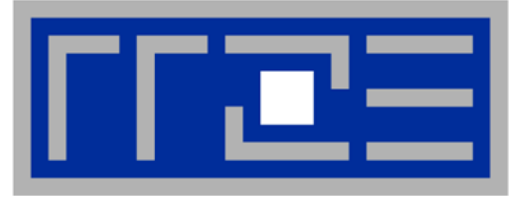
→ pure hardware probing, no impact from OpenMP overhead



# Throughput vector triad on Sandy Bridge socket (3 GHz)



Scalable BW in L1, L2, L3 cache



# **OpenMP performance issues on multicore**

**Synchronization (barrier) overhead**



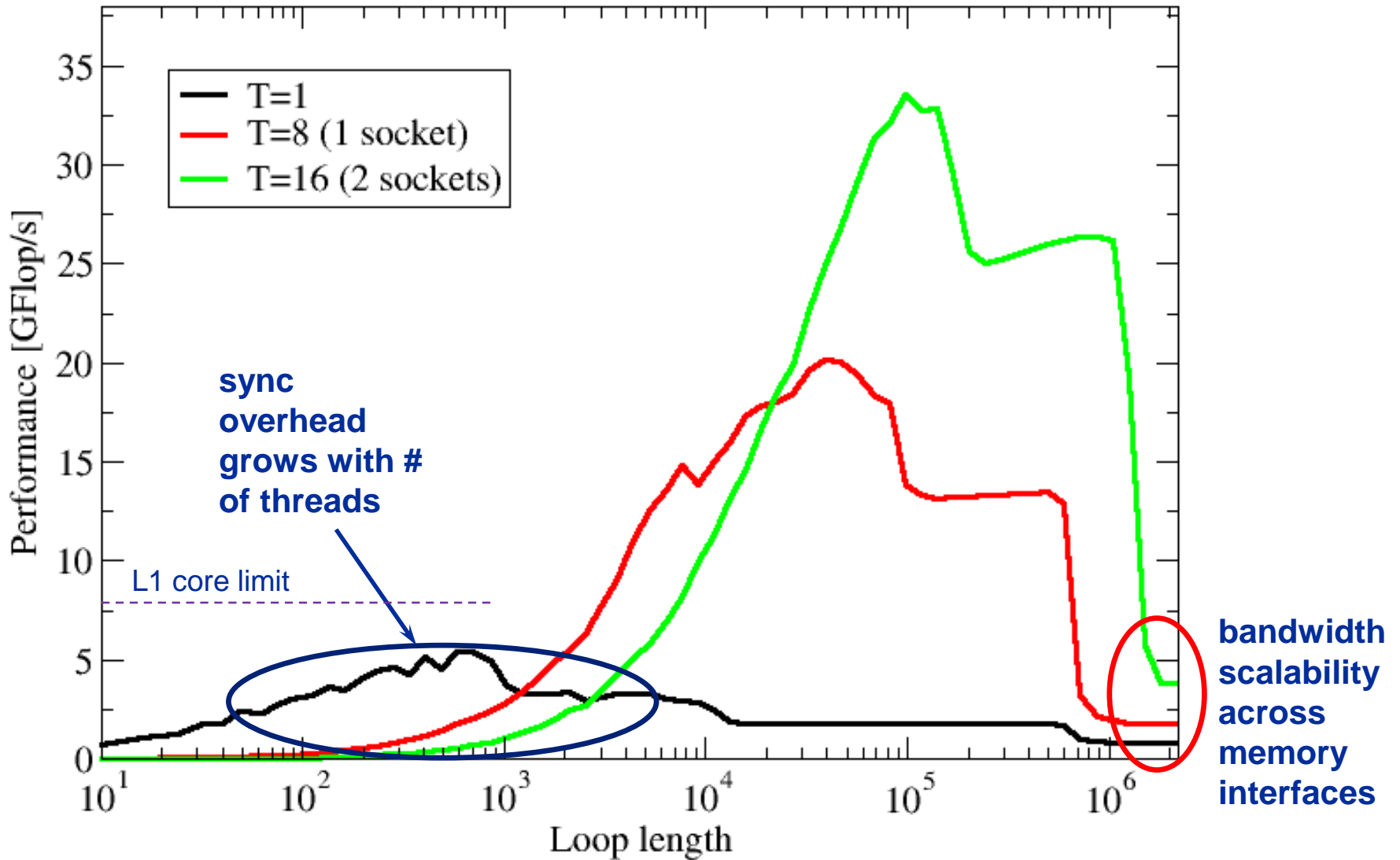
## OpenMP work sharing in the benchmark loop

```
double precision, dimension(:), allocatable :: A,B,C,D

allocate(A(1:N),B(1:N),C(1:N),D(1:N))
A=1.d0; B=A; C=A; D=A
!$OMP PARALLEL private(i,j)
do j=1,NITER
!$OMP DO
  do i=1,N
    A(i) = B(i) + C(i) * D(i)
  enddo
!$OMP END DO
  if(.something.that.is.never.true.) then
    call dummy(A,B,C,D)
  endif
enddo
!$OMP END PARALLEL
```

Implicit barrier

# OpenMP vector triad on Sandy Bridge socket (3 GHz)



# Welcome to the multi-/many-core era

*Synchronization of threads may be expensive!*



!\$OMP PARALLEL ...

...

!\$OMP BARRIER

!\$OMP DO

...

!\$OMP ENDDO

!\$OMP END PARALLEL

Threads are synchronized at **explicit** AND **implicit** barriers. These are a main source of overhead in OpenMP programs.

Determine costs via modified OpenMP  
Microbenchmarks testcase (epcc)

## On x86 systems there is no hardware support for synchronization!

- Next slide: Test **OpenMP** Barrier performance...
- for different compilers
- and different topologies:
  - shared cache
  - shared socket
  - between sockets
- and different thread counts
  - 2 threads
  - full domain (chip, socket, node)

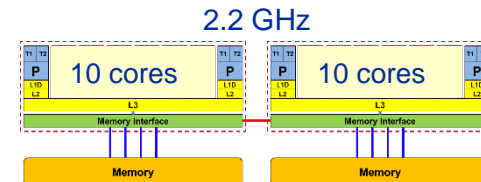
# Thread synchronization overhead on IvyBridge-EP

Barrier overhead in CPU cycles



2 Threads	Intel 16.0	GCC 5.3.0
Shared L3	599	425
SMT threads	612	423
Other socket	1486	1067

Strong topology dependence!



Full domain	Intel 16.0	GCC 5.3.0
Socket (10 cores)	1934	1301
Node (20 cores)	4999	7783
Node +SMT	5981	9897



Overhead grows with thread count

- Strong dependence on compiler, CPU and system environment!
- `OMP_WAIT_POLICY=ACTIVE` can make a big difference



2 threads on  
distinct cores:  
730

	SMT1	SMT2	SMT3	SMT4
One core	n/a	963	1580	2240
Full chip	5720	8100	9900	11400

Still the pain may be much larger, as more work can be done in one cycle on Phi compared to a full Ivy Bridge node

3.2x cores (20 vs 64) on Phi

4x more operations per cycle per core on Phi

→  $4 \cdot 3.2 = 12.8x$  more work done on Xeon Phi per cycle

1.9x more barrier penalty (cycles) on Phi (11400 vs. 6000)

→ One barrier causes  $1.9 \cdot 12.8 \approx 24x$  more pain 😊.



- **Affinity matters!**
  - Almost all performance properties depend on the position of
    - Data
    - Threads/processes
  - Consequences
    - Know where your threads are running
    - Know where your data is
  
- **Bandwidth bottlenecks are ubiquitous**
  
- **Synchronization overhead may be an issue**
  - ... and also depends on affinity!
  - Many-core poses new challenges in terms of synchronization