

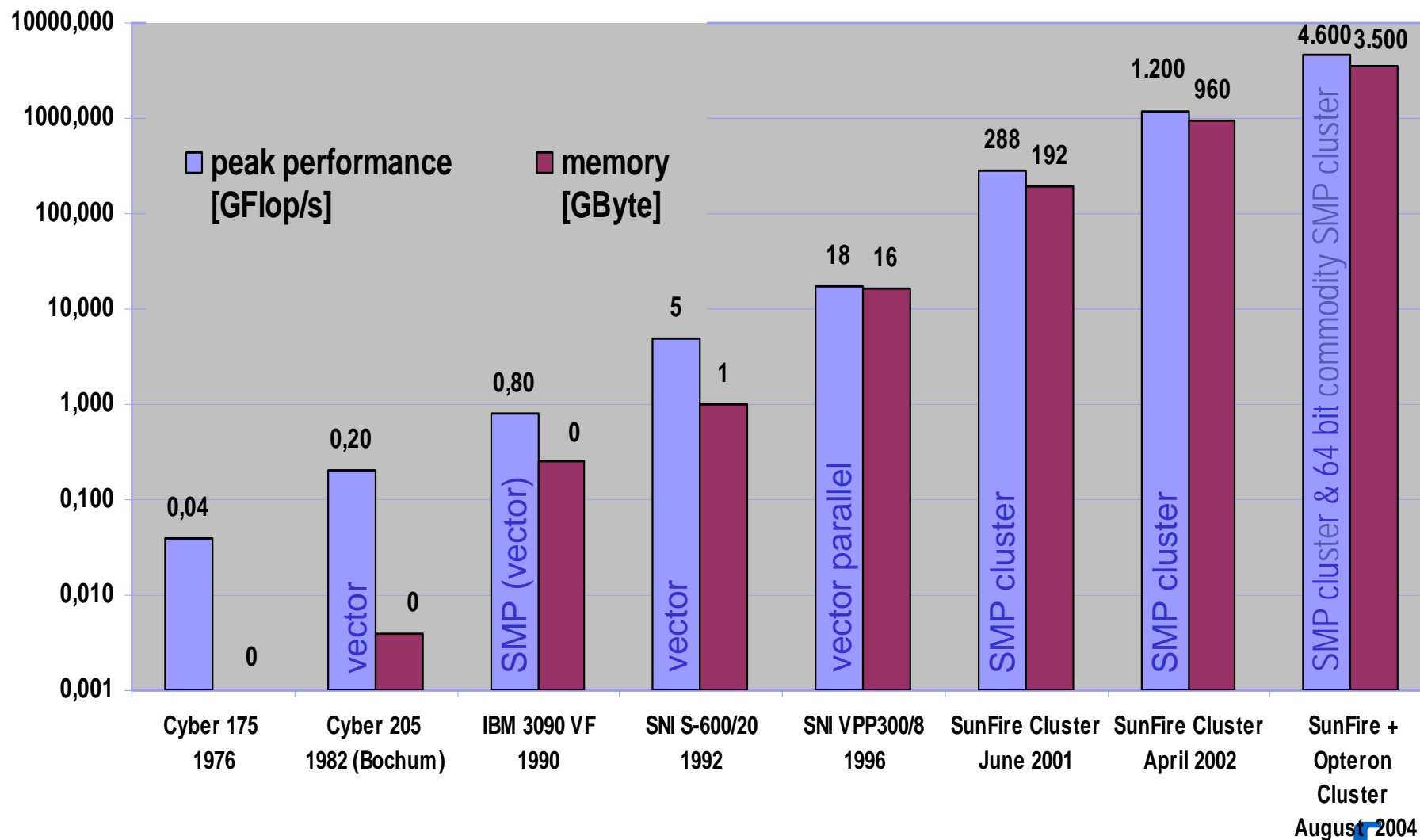
High Performance Computing State of the Art and Limitations

Christian Bischof

**Center for Computing and Communication
Institute for Scientific Computing
RWTH Aachen University
www.rz.rwth-aachen.de, www.sc.rwth-aachen.de**

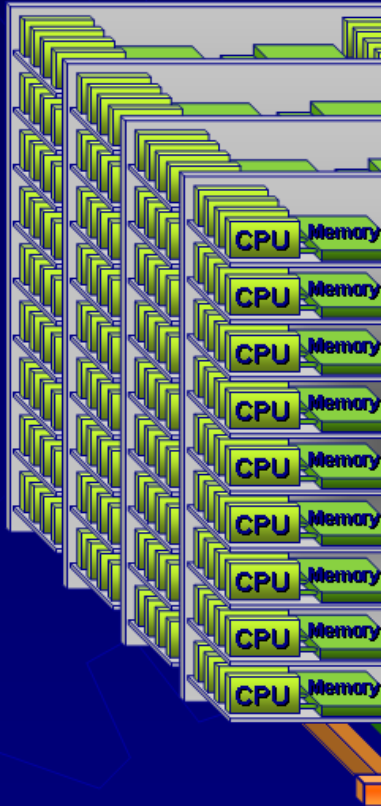
- Ingredients
 - Mathematical models,
 - Numerical methods,
 - Computer science algorithms, and
 - Software engineering approaches
- Infrastructure
 - High Performance Computing
 - Large-Scale Data Handling (1 PB archive @ RWTH)
 - Visualization Capabilities (Virtual Reality Center Aachen)
 - Broadband Network Connectivity (10 GB backbone @ RWTH)
- The importance of simulation is reflected by its growing role in applied engineering (e.g. CEM2006) and structural changes at universities, e.g. Center for Computational Engineering Sciences (CCES) at RWTH Aachen University.

HPC Ressources at Aachen University



HPC System at Aachen University

4 x E25k Cluster
(SunFire Link)



4 x E25k systems with:

→ 72 UltraSPARC IV, 288 GByte memory

16 x E6900 systems with

→ 24 UltraSPARC IV, 96 GByte memory

8 x E2900 systems with

→ 12 UltraSPARC IV, 48 GByte memory

64 quad-Opteron systems

→ 2,2 GHz Opterons, 8 GByte memory

4 quad-Opteron systems (dual core)

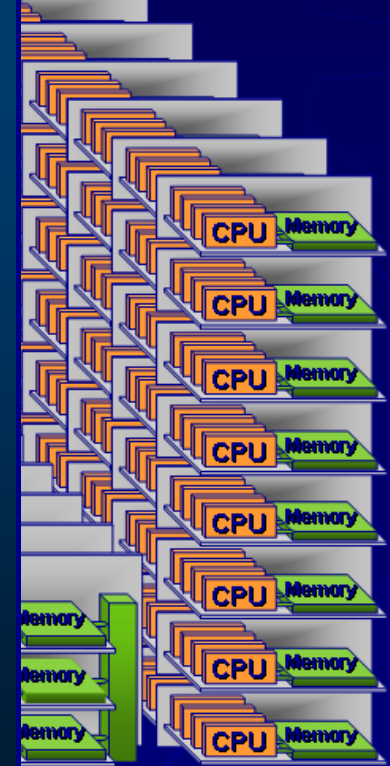
→ 2,2 GHz Opterons, 16 GByte memory

Total:

→ over 4.6 TFlops

→ over 3.5 TByte memory

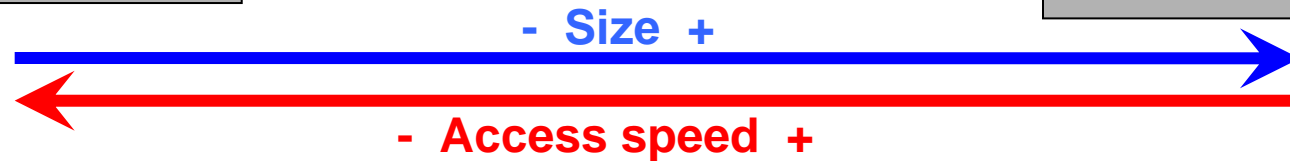
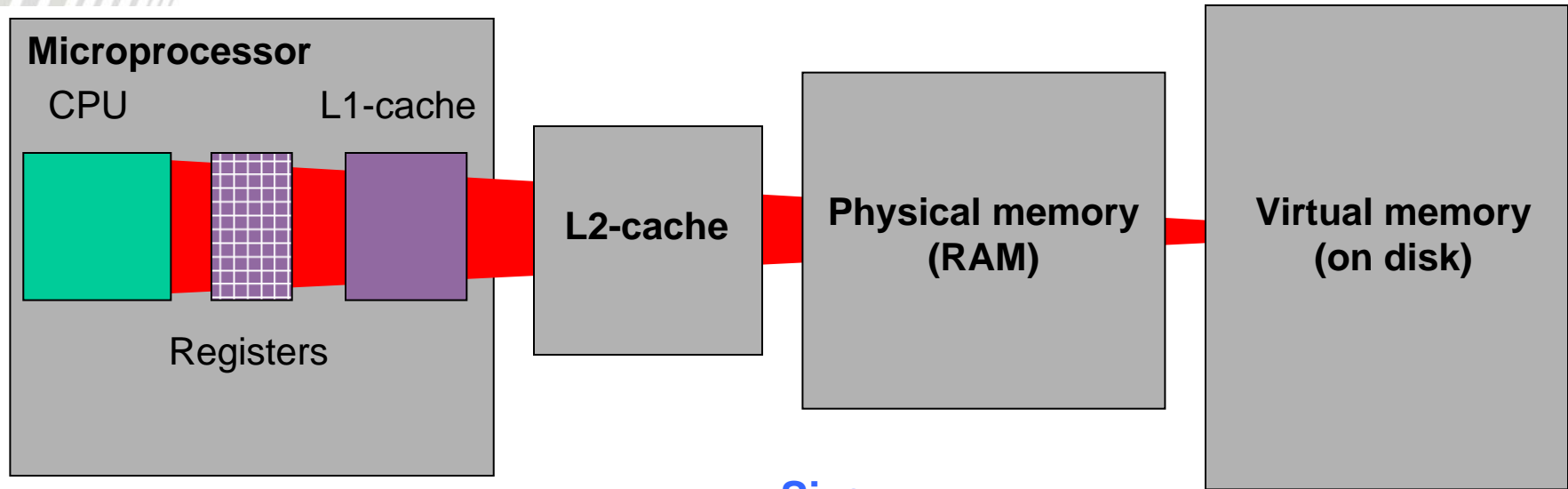
Quad-Opteron
Cluster



Storage Area Network (SAN)



Memory Hierarchy



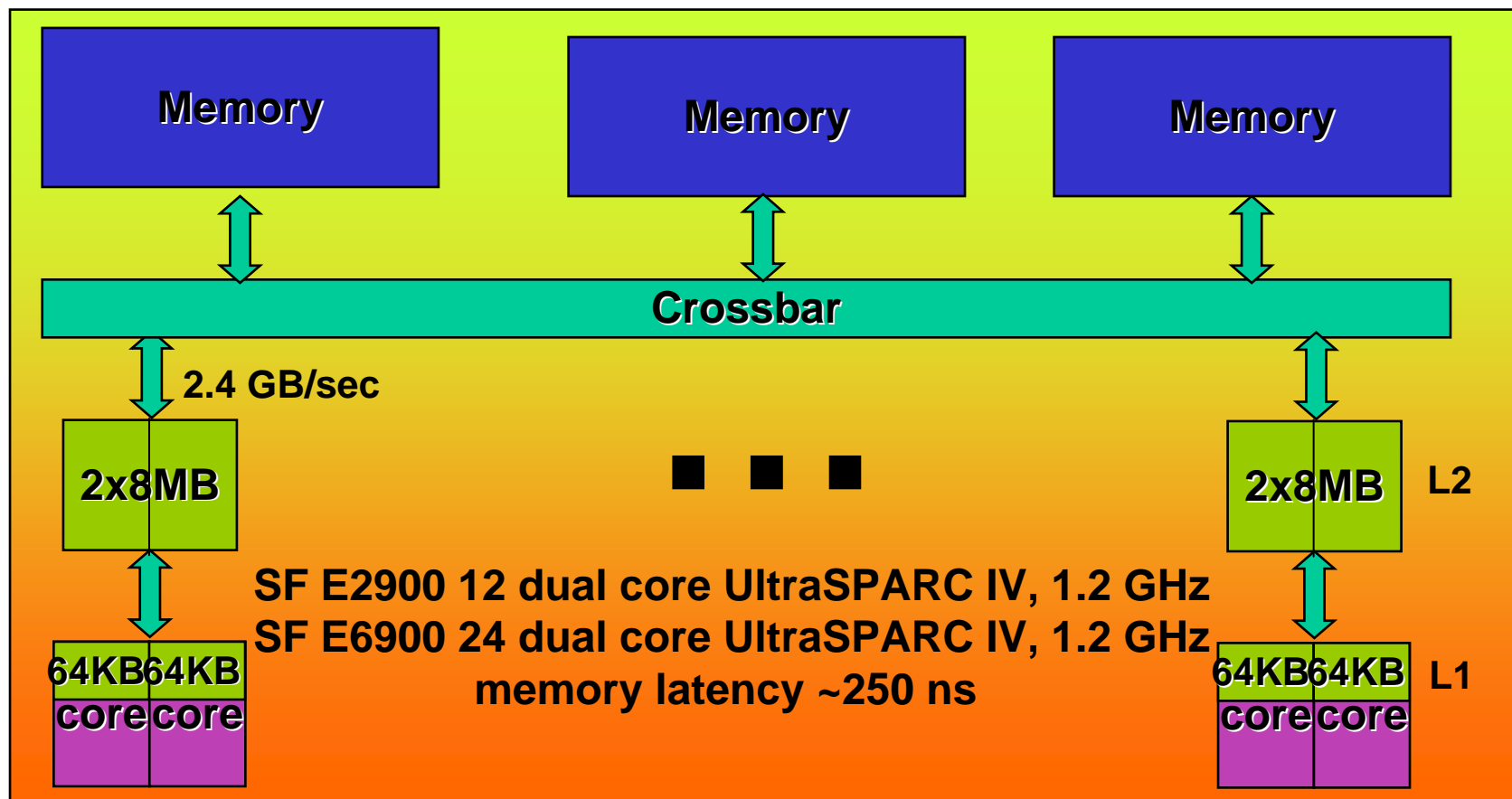
Memory access time	<1ns	~5ns	~50ns	much slower
Processor speed (Flops)	3 Gflops	600 Mflops	60 Mflops	

Caches only “work” when data is reused (**data locality**). Random access to memory (e.g. pointer chasing) will slow down a process to the speed of memory.

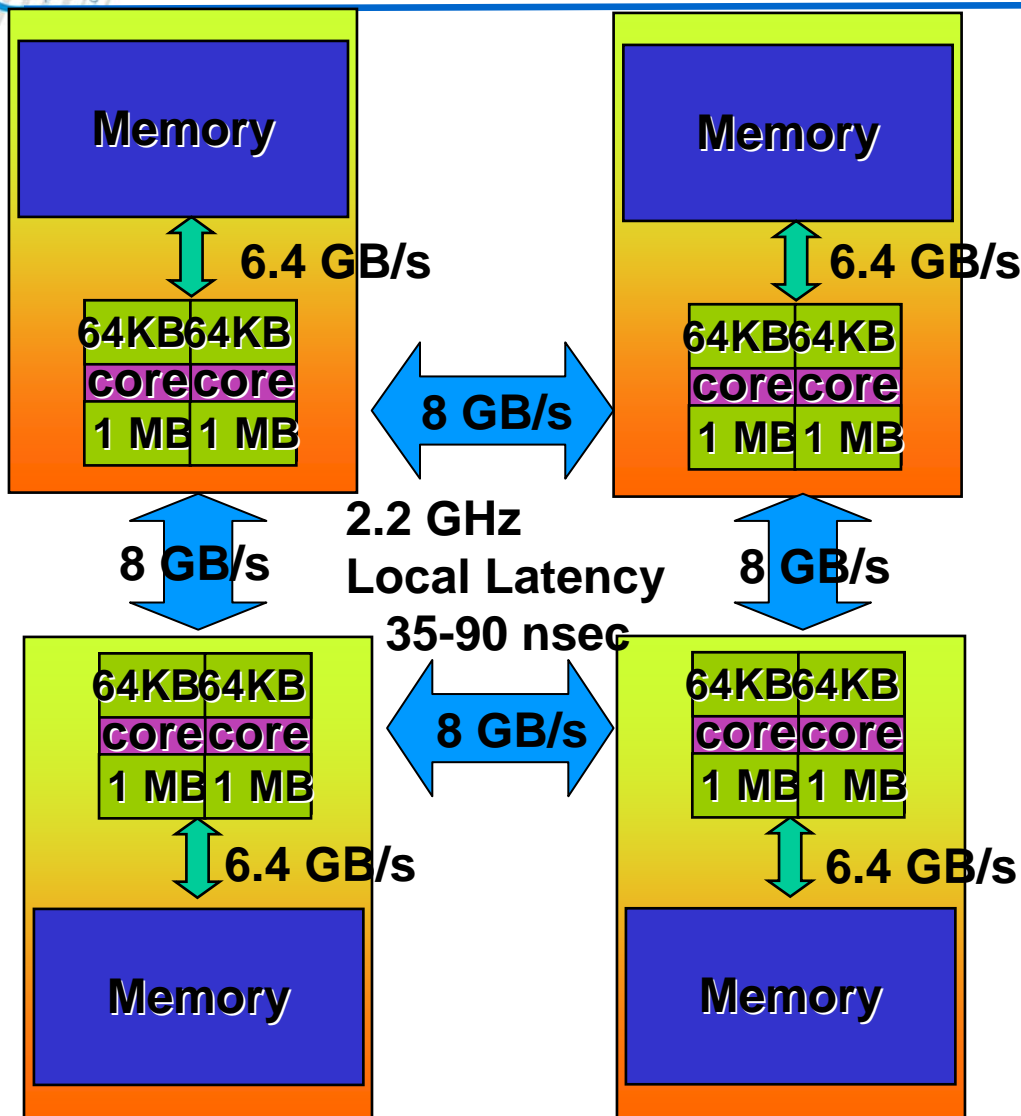
- **Process** = instruction stream
- **Thread** = “light weight” process
much faster to schedule a thread than a process
- **Core** = Central Processing Unit = CPU
= the part of a computer that interprets instructions
and processes data
- **Multi-Core Processors**: #Cores > #processor chips = # sockets

Sun Fire E2900 / E6900

- **Shared Memory Multiprocessor (SMP)**
- From a programmer's perspective: **Uniform Memory Access (UMA)**
- Contents of caches are kept coherent across processors



Sun Fire V40z (w/ AMD Opteron Chip)



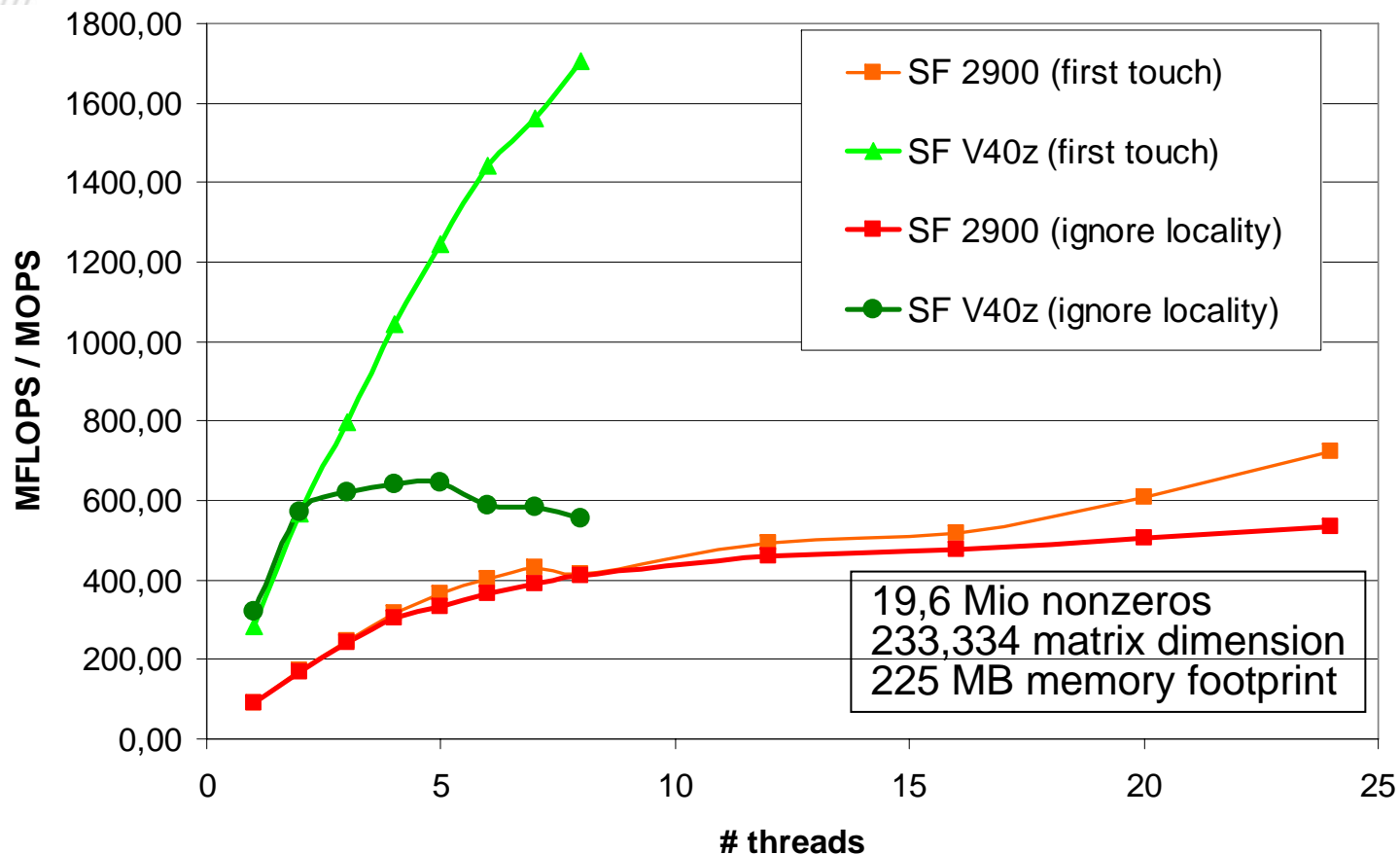
- On a SMP **node**:
Cache contents again kept coherent across chips, but it does make a difference, which memory a processor tries to access:

Cache coherent
Non-uniform memory access
(ccNUMA)

- **SMP** nodes connected by a network (**SMP cluster**)

Distributed Memory
between nodes

Scalability of sparse Mtx-vector Multiply



Performance of a cc-Numa system is very sensitive to data placement.
The first-touch strategy redistributes memory to the processor that requests (touches) it first. This way data is being distributed after being read in.

Throughput vs. Processor Performance

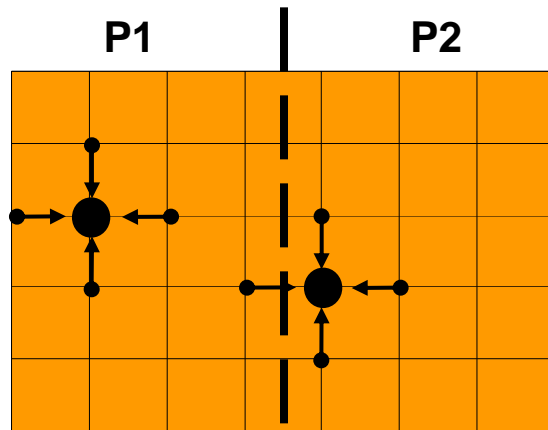
Nodes w/ Interconnect	8 * Sun Fire E6900 w/ Sun Fire Link	64 * Sun Fire V40z w/ Gigabit Ethernet
Memory System	A shared large memory of max.192GB is available for all 48 processor cores.	The maximum memory available for a single process is max 32 GB.
Programming	The flat memory simplifies parallel programming.	ccNUMA and distributed memory make programming challenging.
Interconnect:	Low latency. Bandwidth can only be saturated through multiple simultaneous transfers.	High latency. Transfers require a lot of CPU power. A single transfer consumes the whole bandwidth.
Optimization Goal	Throughput of whole system	Performance of one processor

Parallel Programming

- **Shared Memory Programming: OpenMP and Autoparallelization**
 - Can be very productive
 - Single-source code development
 - Shared Memory simplifies load balancing, but scalability is limited
 - Not trivial: Data Races (result depends on interleaving of threads)
- **Distributed Memory Programming: Message Passing Interface (MPI)**
 - Conceptually simple: Send or receive messages
 - Requires logical decomposition of data: can be a lot of work
 - Separate code base
- **Hybrid Programming: OpenMP within a node, MPI between nodes**

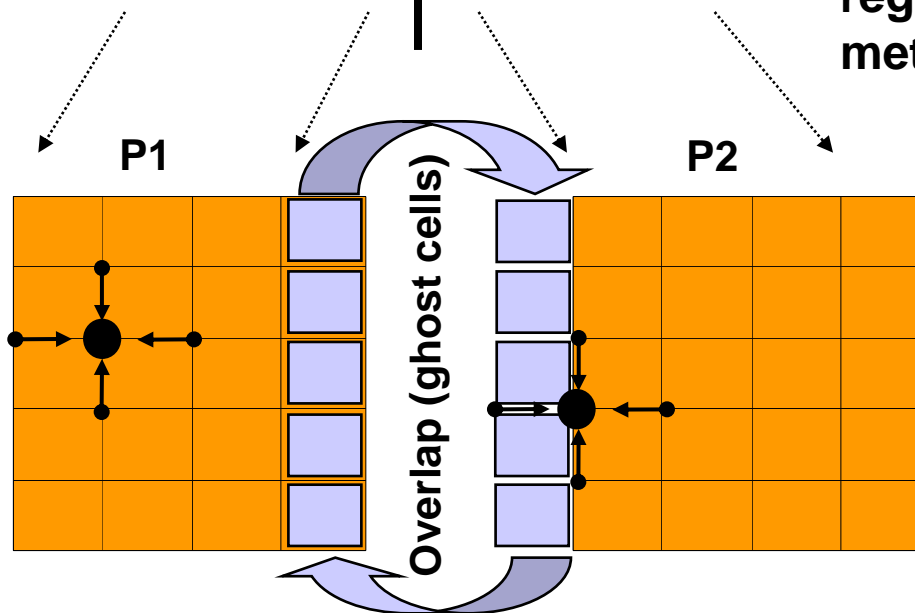
**In either case, good debuggers and performance tools necessary.
CCC regularly runs “bring your own code” workshops**

Jacobi Solver – Domain Decomposition



**Shared Memory
Parallelization**

**Solution of Helmholtz equation on a
regular mesh, using an iterative Jacobi
method with over-relaxation**



**Distributed Memory
Parallelization**

Example: Jacobi Solver – serial

```

error = 10.0 * tol
k = 1
do while (k.le.maxit .and. error.gt. tol)
  error = 0.0

```

! Copy new solution into old

```

  do j=1,m
    do i=1,n
      uold(i,j) = u(i,j)
    enddo
  enddo

```

! Compute stencil, residual & update

```

  do j = 2,m-1
    do i = 2,n-1
      resid = (ax*(uold(i-1,j) + uold(i+1,j))
&             + ay*(uold(i,j-1) + uold(i,j+1))
&             + b * uold(i,j) - f(i,j))/b
      u(i,j) = uold(i,j) - omega * resid
      error = error + resid*resid
    end do
  enddo

```

```

  k = k + 1
  error = sqrt(error)/dble(n*m)
enddo

```

Jacobi Solver – OpenMP

```

error = 10.0 * tol
k = 1
do while (k.le.maxit .and. error.gt. tol)
  error = 0.0
  !$omp parallel private (resid)
  !$omp do
    do j=1,m; do i=1,n
      uold(i,j) = u(i,j)
    enddo; enddo
  !$omp end do

  !$omp do reduction(+:error)
    do j = 2,m-1; do i = 2,n-1

      resid = (ax*(uold(i-1,j) + uold(i+1,j)) ...

      u(i,j) = uold(i,j) - omega * resid
      error = error + resid*resid

    end do; enddo
  !$omp end do nowait
  !$omp end parallel do
  k = k + 1
  error = sqrt(error)/dble(n*m)
enddo

```

Optimization:

Two **parallel loops** in one **parallel region** to avoid

```

do while (k.le.maxit .and. error.gt. tol)
  error = 0.0; reqcnt = 0
  if ( me .ne. 0 ) then ! receive stripe mlo from left neighbour
    reqcnt = reqcnt + 1
    call MPI_Irecv( uold(1,mlo), n, ., me-1, 11, .,reqary(reqcnt),.)
  end if
  if ( me .ne. np-1 ) then ! receive stripe mhi from right neighbour
    reqcnt = reqcnt + 1
    call MPI_Irecv( uold(1,mhi), n, ., me+1, 12, .,reqary(reqcnt),.)
  end if
  if ( me .ne. np-1 ) then ! send stripe mhi-1 to right neighbour
    reqcnt = reqcnt + 1
    call MPI_Isend ( u(1,mhi-1), n, ., me+1, 11, .,reqary(reqcnt),.)
  end if
  if ( me .ne. 0 ) then ! send stripe mlo+1 to left neighbour
    reqcnt = reqcnt + 1
    call MPI_Isend ( u(1,mlo+1), n, ., me-1, 12, .,reqary(reqcnt),.)
  end if
  do j=mlo+1,mhi-1; do i=1,n
    uold(i,j) = u(i,j)
  enddo; enddo
  call MPI_Waitall ( reqcnt, reqary, reqstat, .)
  do j = mlo+1,mhi-1; do i = 2,n-1
    resid = (ax*(uold(i-1,j) + uold(i+1,j)) ...
    u(i,j) = uold(i,j) - omega * resid
    error = error + resid*resid
  end do; enddo
  error_local = error
  call MPI_Allreduce ( error_local, error,1, .,MPI_SUM,...)
  k = k + 1; error = sqrt(error)/dble(n*m)
enddo

```

**Overlapping communication
and computation**

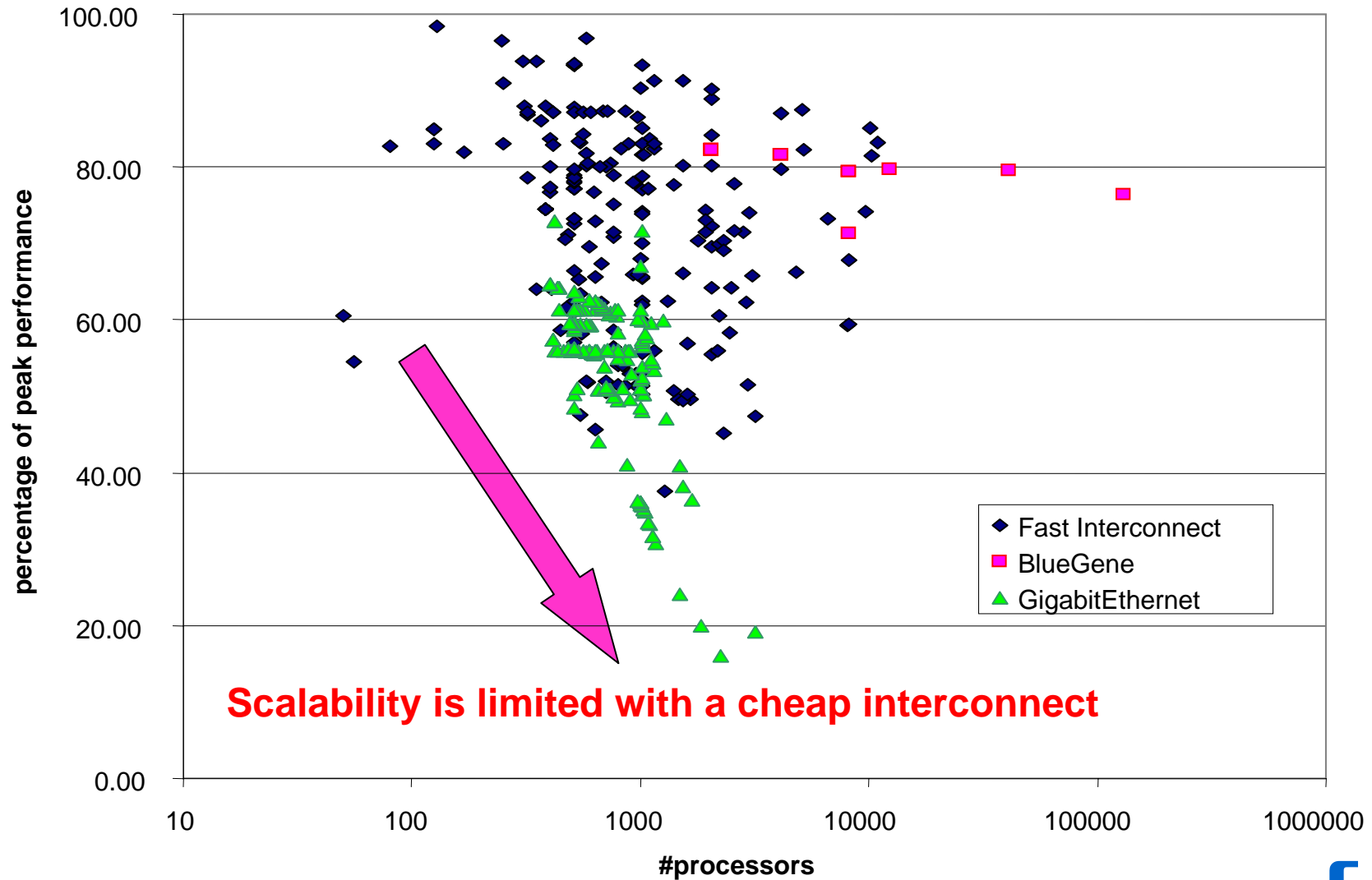
- Standard Building Blocks (from single-core Xeon or Opteron upwards) connected through a network.
- **No shared memory between cluster nodes**
- Cheap standard interconnect: GBit Ethernet
 - 100 MByte/sec bandwidth, 70-150 μ sec latency
- Not cheap interconnects: Infiniband/Myrinet/Quadrics:
 - ~ 1 GByte/sec Bandbreite, 1.5 - 5 μ sec latency
 - PCI-X oder PCI-Express interfaces
 - Multi-Rail connections to increase bandwidth
- Proprietary Interconnects:
 - From NEC, SGI, IBM, Cray, Qlogic
 - Multiple GBytes/sec, 1-2 μ sec latency
- **In theory, can build systems of arbitrary performance by aggregating enough nodes.**

Top 10 of the Top 500 (www.top500.org)

Rank	Site	Computer	Procs	Tflop/s peak	Processor / Node	Interconnect	% of peak
1	LLNL (USA)	IBM BlueGene	131072	281	1 * dual core PowerPC (0.7 GHz)	Proprietary	77
2	IBM (USA)	IBM BlueGene	40960	91	1 * dual core PowerPC (0.7 GHz)	Proprietary	79
3	LLNL (USA) "ASC Purple"	IBM pSeries 575	10240	63	8 * dual-core Power5 (1.9 Ghz)	Federation	81
4	NASA Ames (USA) "Columbia"	SGI Altix Itanium 2	10160	52	512 * Itanium 2 (1.5 GHz)	Numalink/Infiniband	85
5	Sandia Nat'l Lab (USA) "Thunderbird"	Dell PowerEdge 1850	8000	38	2 * Pentium Xeon EM64T (3.6 GHz)	Infiniband	59
6	Sandia Nat'l Lab (USA) "Red Storm"	Cray XT3	10880	36	1 * Opteron 100 (2GHz)	XT3 Internal Intercon.	83
7	"Earth Simulator" (J)	NEC Parallel Vector	5120	36	8 * NEC SX-6 (1 GHz)	Multi-stage crossbar	88
8	Barcelona Supercomp C € "Mare Nostrum"	IBM JS20 Cluster	4800	28	2 * PowerPC 970 (2.2 GHz)	Myrinet	66
9	ASTRON/U Groningen (NL)	IBM BlueGene	12288	27	1 * dual core PowerPC (0.7 GHz)	Proprietary	78
10	Oak Ridge Nat'l Lab (USA) "Jaguar"	Cray XT3, 2.4 GHz	5200	21	1 * Opteron 100 (2.4 GHz)	XT3 Internal Intercon.	84

The TOP500 benchmark measures the performance of solving a (very large) system of dense linear equations with an LU factorization to determine the „fastest“ computer.

The Effect of a Fast Interconnect

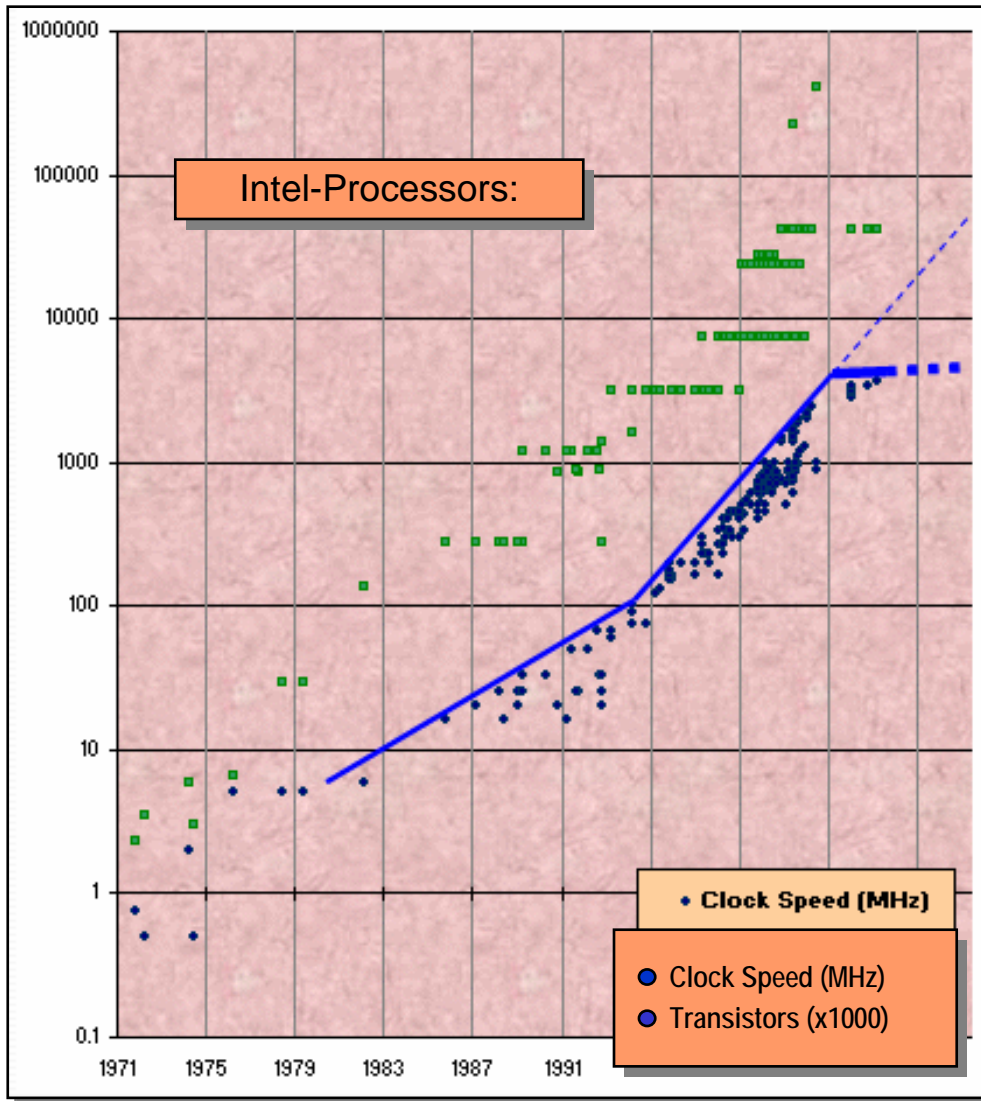


Scalability is limited with a cheap interconnect

IBM Blue Gene

- IBM Blue Gene systems are listed on ranks 1, 2, 9, 12, 13, 22, 29- 31, 32, and 73-81 of the current Top500 list.
- Design Characteristics:
 - **slow processors**
dual-core PowerPC-based at 700 MHz (!)
 - **fast and sophisticated interconnect**
(3D-Torus+Tree+Barrier+Clock)
 - **dense packaging** (1024 CPUs + 512 GB Memory pro Rack).
Possible because slow processors require not much power and do not generate much heat.
- An 8-rack Blue Gene/L system called JUBL (=Jülicher Blue Gene/L) was inaugurated just two months ago at the John von Neumann Institut for Computing, the national supercomputing center at the Research Centre in Jülich.

Moore's Law



The number of transistors on a chip is still doubling every 18 months

... but the clock speed is no longer growing that fast.

The TOP500 increases by 2,4 every 18 months due to increased parallelism in addition to Moore's Law.

Source: Herb Sutter
www.gotw.ca/publications/concurrency-ddj.htm

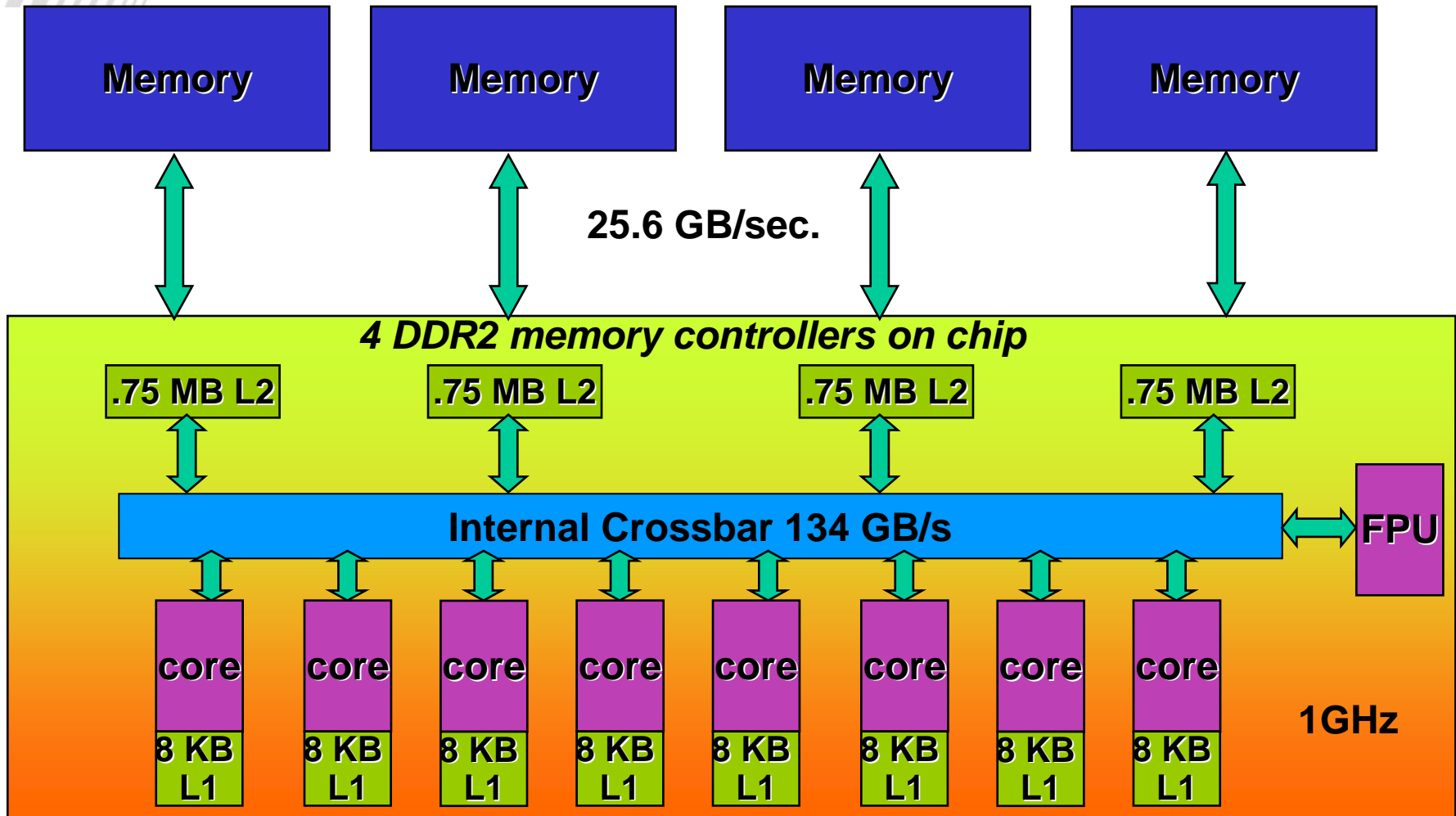
Design Considerations

- Power consumption (and heat generated)
 - Scales linearly with #transistors but quadratically with clock rate
 - With a 2,2 GHz Opteron (95 Watts) as basis:
 1. A twice as fast processor with doubled cache would consume $(2^2 \times 2) \times 95 \text{ Watt} = 760 \text{ Watt}$
 2. A half as fast chip with 8 cores would consume $(1/4 \times 2) \times 95 \text{ Watt} = 50 \text{ Watt}$ and theoretically is twice as fast!

- There is no Moore's Law for memory latency
 - Memory latency halves only every six years.
 - Bigger caches will be of limited use.

- **Conclusion for commodity chips: Take the Blue Gene Route!**
Slower processors with moderate amount of cache tightly integrated on a chip.

Sun Fire T2000 (chip size 378 mm², 72 W)



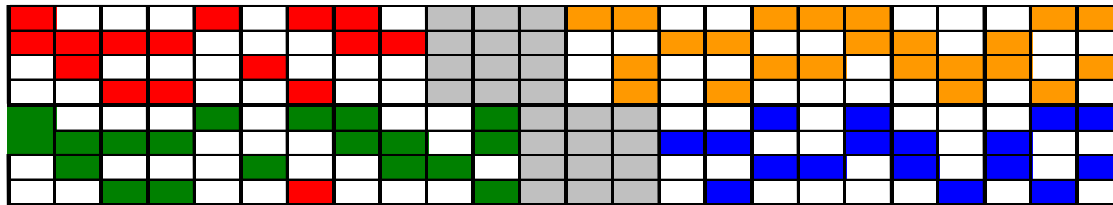
One Floating-Point unit (FPU) shared among all processors.

Terminology (continued)

- **Chip(-level) multiprocessing (CMP) :**
multiple processor "cores" are included in the same integrated circuit, executing independent instruction streams.
- **Chip(-level) multithreading (CMT)**
Multiple threads are executed within one processor core at the same time. Only one is active at a given time (time-slicing).

Chip Level Parallelism

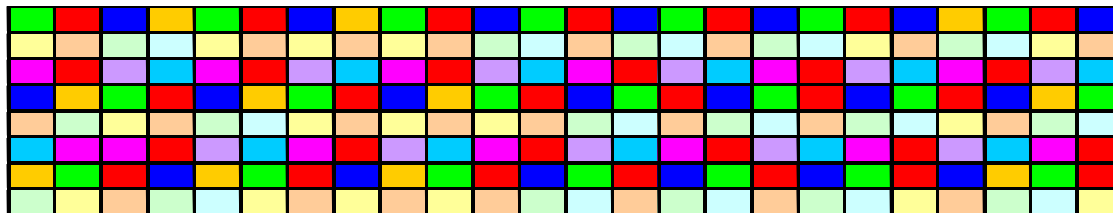
Chip-Level Multiprocessing



↔ = 0.66 ns


*UltraSPARC IV+, CMP
superscalar, dual core
2 x 4 sparc v9 instr/cycle
1 active thread per core*

Chip-Level Multithreading



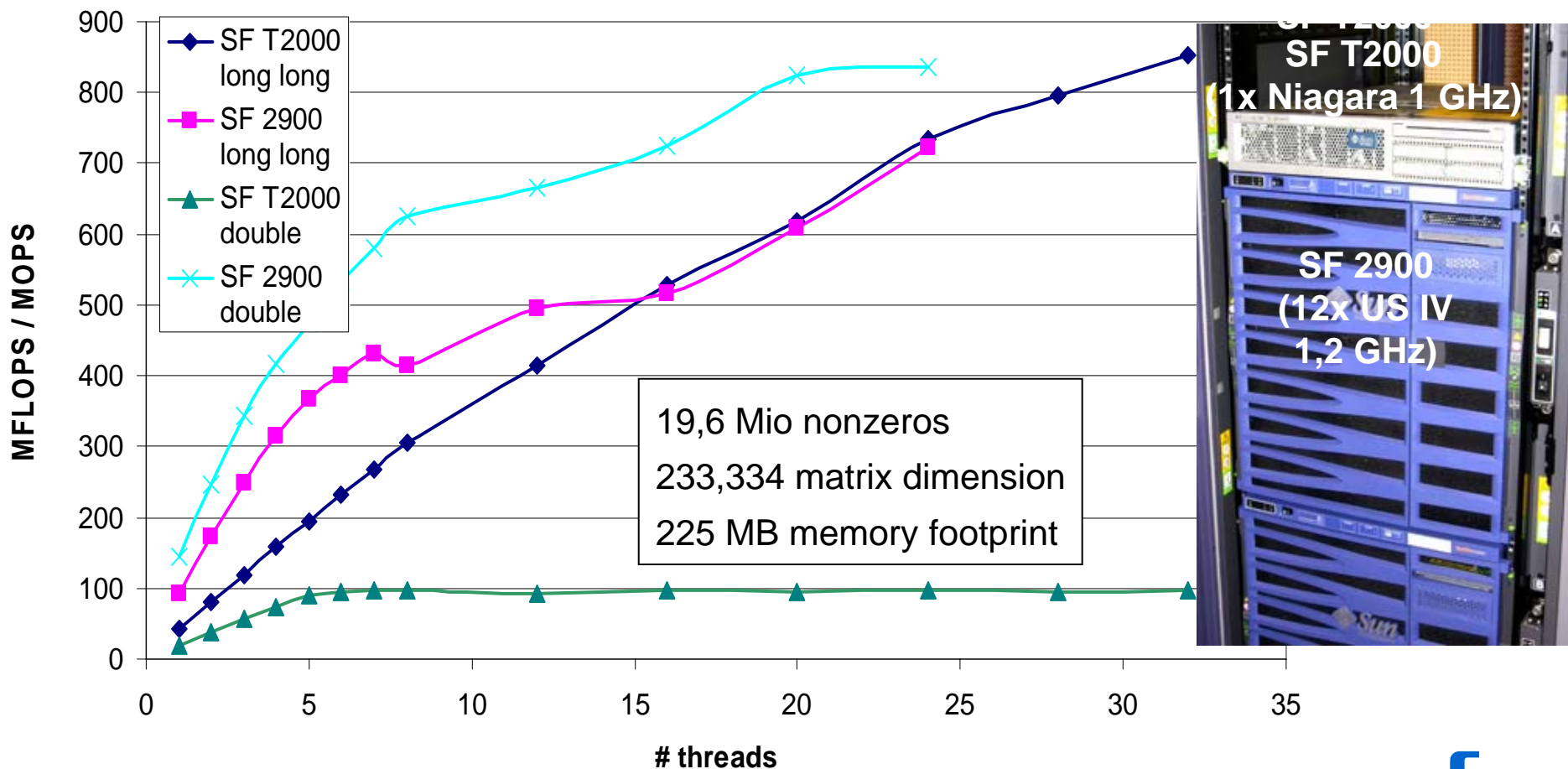
↔ = 1.0 ns

*UltraSPARC T1, CMP+CMT
Single issue, 8 cores
8 x 1 sparc v9 instr/cycle
4 active threads per core
context switch comes for free*

—————→
context switch  time

Sparse Matrix Vector Multiplication

What, if the Niagara could really compute with floating point numbers ...



- The SMP box is shrinking:
 - Substantial multithreading
 - with low latency
 - and high throughput
 - and good floating point performance
 - is coming.
- In a few years from now, many or all applications will be multi-threaded
- Substantially parallel SMP boxes with small footprint will be building blocks of large systems, commodity or custom.
- Nested and/or hybrid parallelization will become a common case in HPC
- **There is no way to escape parallelism!**

Mathematics + Numerics + Data Structures

+

Software Skills (Efficiency, Portability, Maintainability)

+

Parallel Processing Hardware

+

Suitable Methods for understanding results

=

Scientific and Industrial Competitiveness

Put differently:

- **Software development pays in the long run.**
- Progress at the level of the „user-defined subroutine“ is not enough, in particular not at universities.
- „Own code“ has fringe benefits, e.g. **automatic differentiation** for sensitivity-enhancement of codes, thereby easing transition from simulation to design.