SPAI preconditioners for HPC applications

G. Fourestey (CSCS/ETHZ), W. Sawyer (CSCS/ETHZ),
R. Popescu (EPFL), C. Vanini (USI)

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

Monday, November 21, 2011

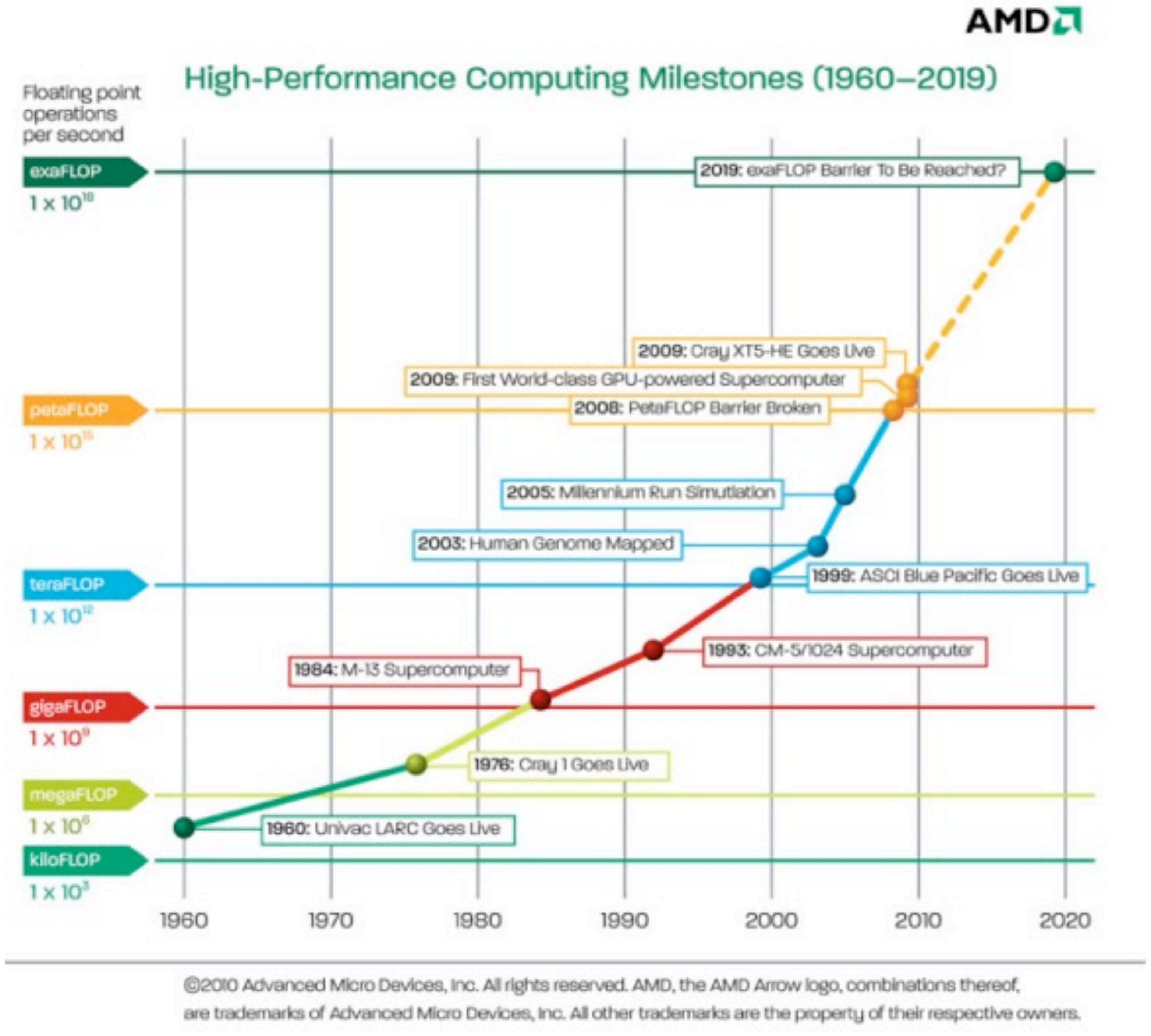
What is HPC

What exactly is High-Performance Computing?

- "... it depends on who you talk to" Richard Dracott (Intel)
- "At IBM we called it scientific computing" Ulla Thiel (Cray)

Scientific computing where computer performance matters, in other words, scientific applications that need a lot of computer performance.

Path to Exascale



Exascale computing is: 1 billion billion sustained FP operations per second (measured using HPL).

Will exascale science exist?

Path to Exascale

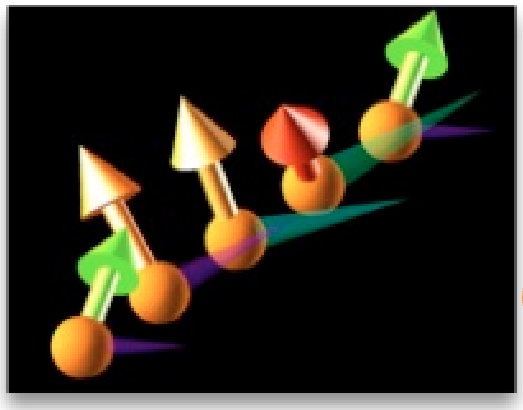
Application performance seems to keep up with supercomputing systems performance (!)

~100 Kilowatts → ← ~5 Megawatts → ← 20-30 MW →

~1 Exaflop/s

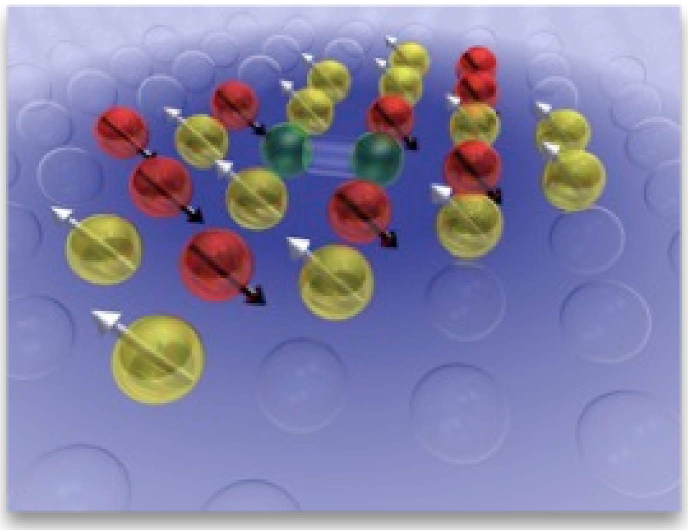
100 million or billion processing cores (!)

1 Gigaflop/s
Cray YMP
8 processors



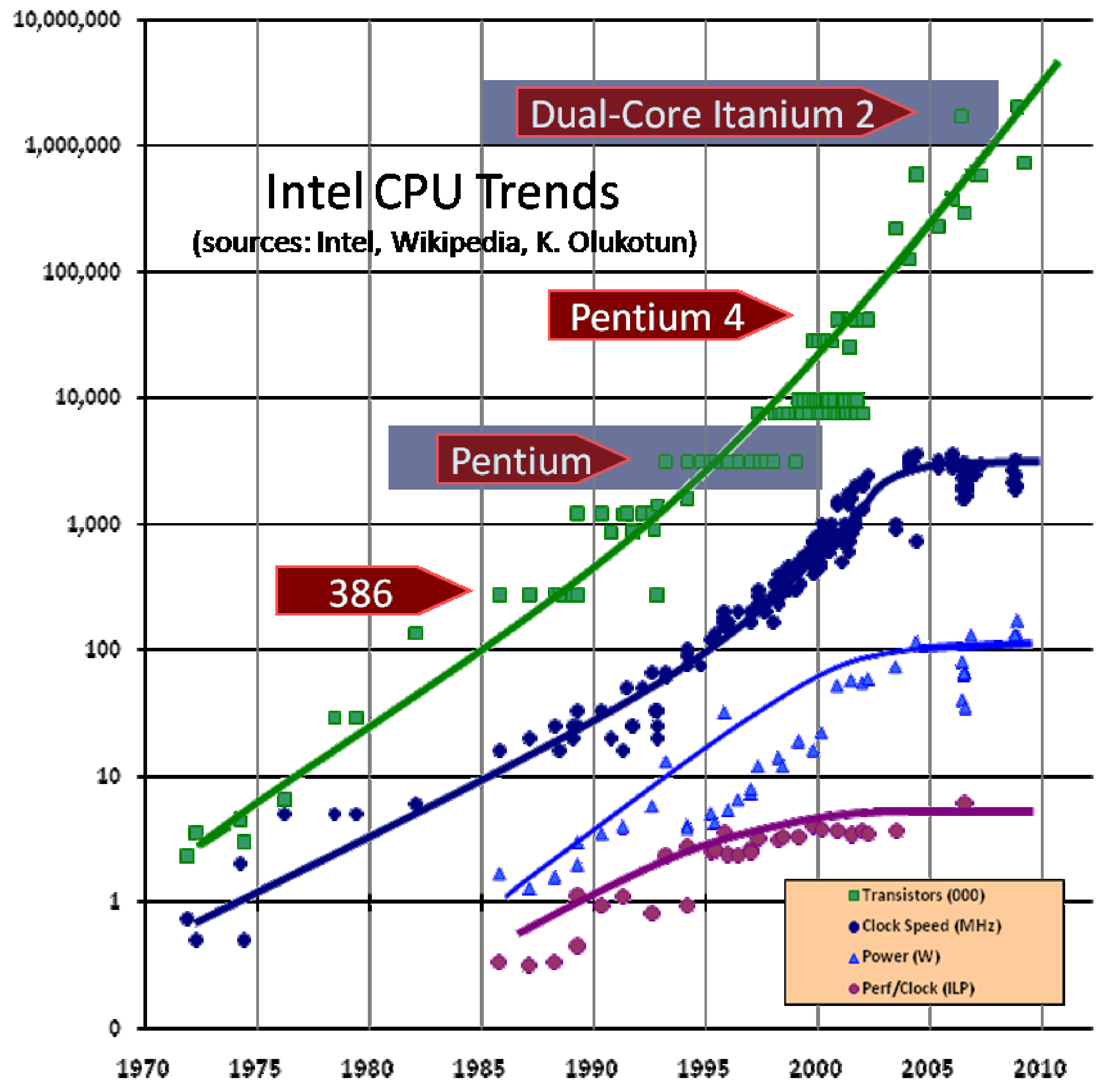
1.02 Teraflop/s
Cray T_{3E}
1’500 processors

1.35 Petaflop/s
Cray XT5
150’000 processors



1988	1998	2008	2018
First sustained GFlop/s Gordon Bell Prize 1988	First sustained TFlop/s Grondon Bell Prize 1998	First sustained PFlop/s Gordon Bell Prize 2008	Another 1,000x in sustained performance increase

CPU Trends



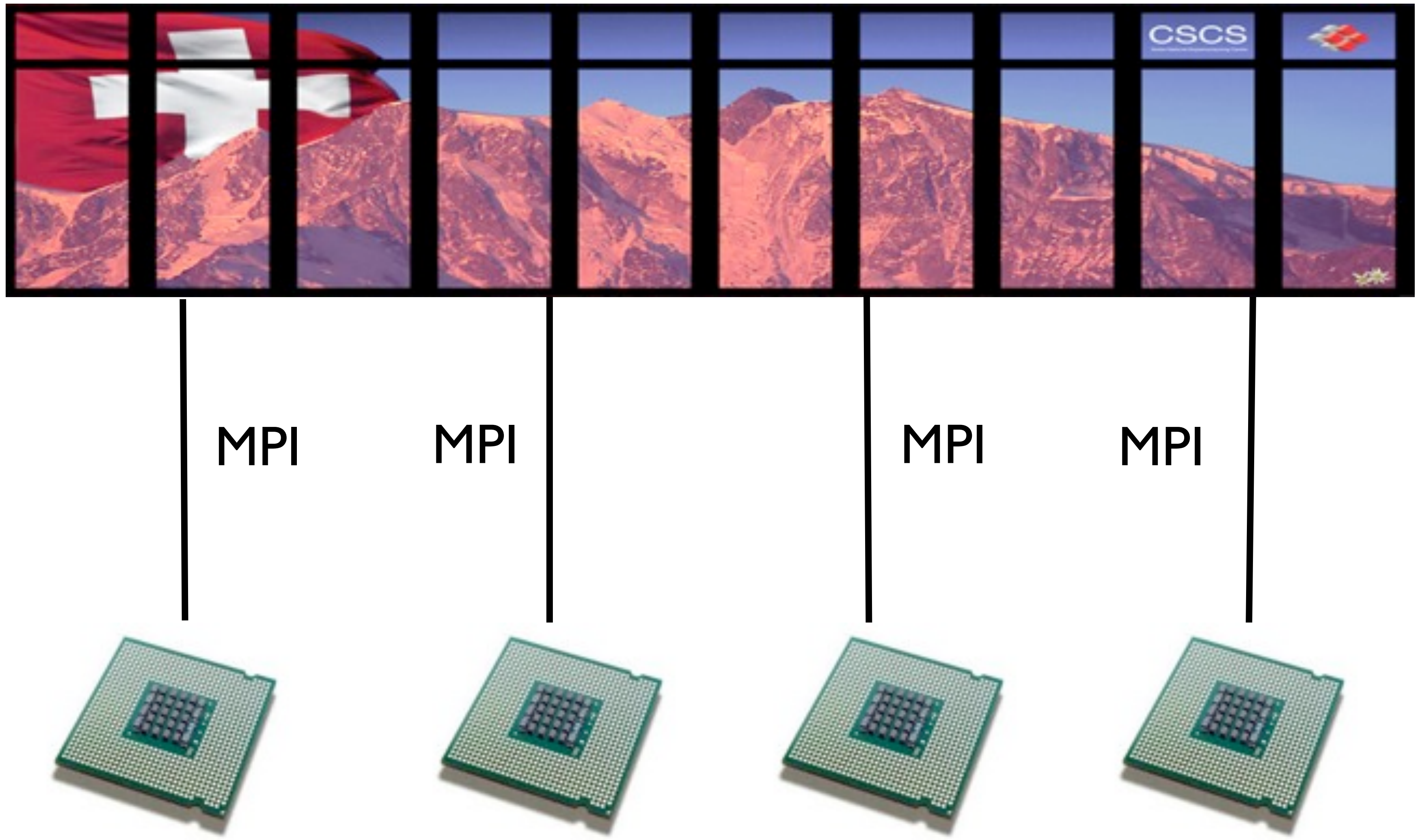
Until 2005, code performance scale wrt the CPU frequency (hardware).

First physical roadblock:
CPU consumption ~ freq^3 !

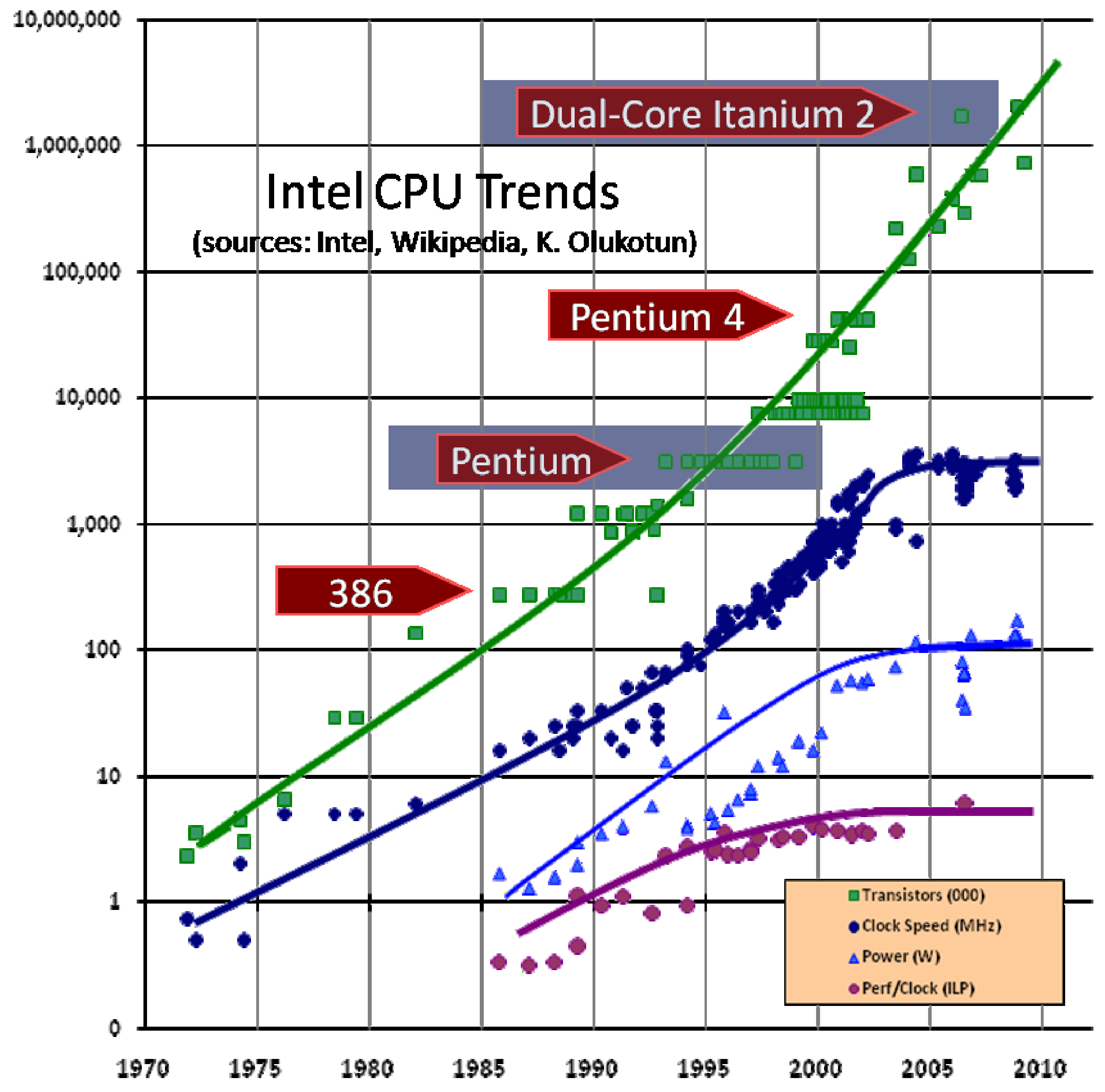
ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS Swiss National Supercomputing Centre

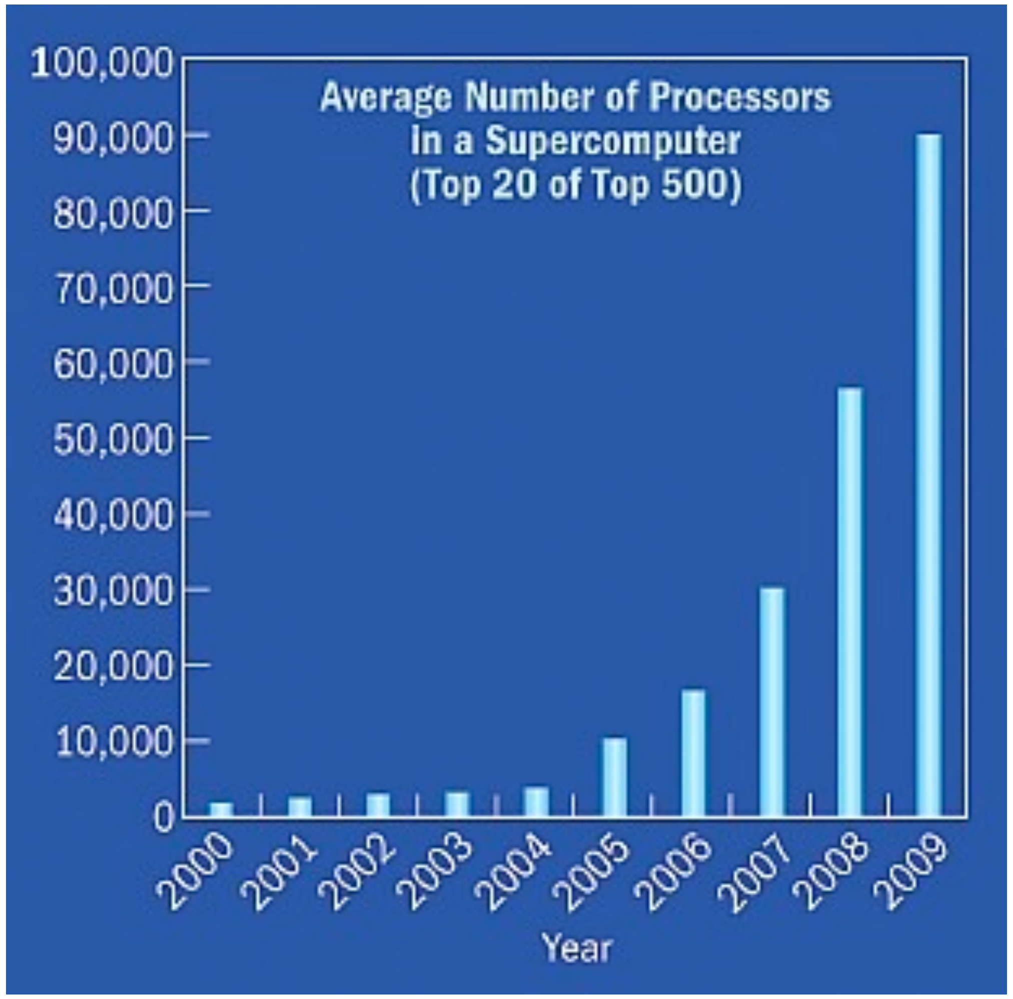
1st Level of Parallelism



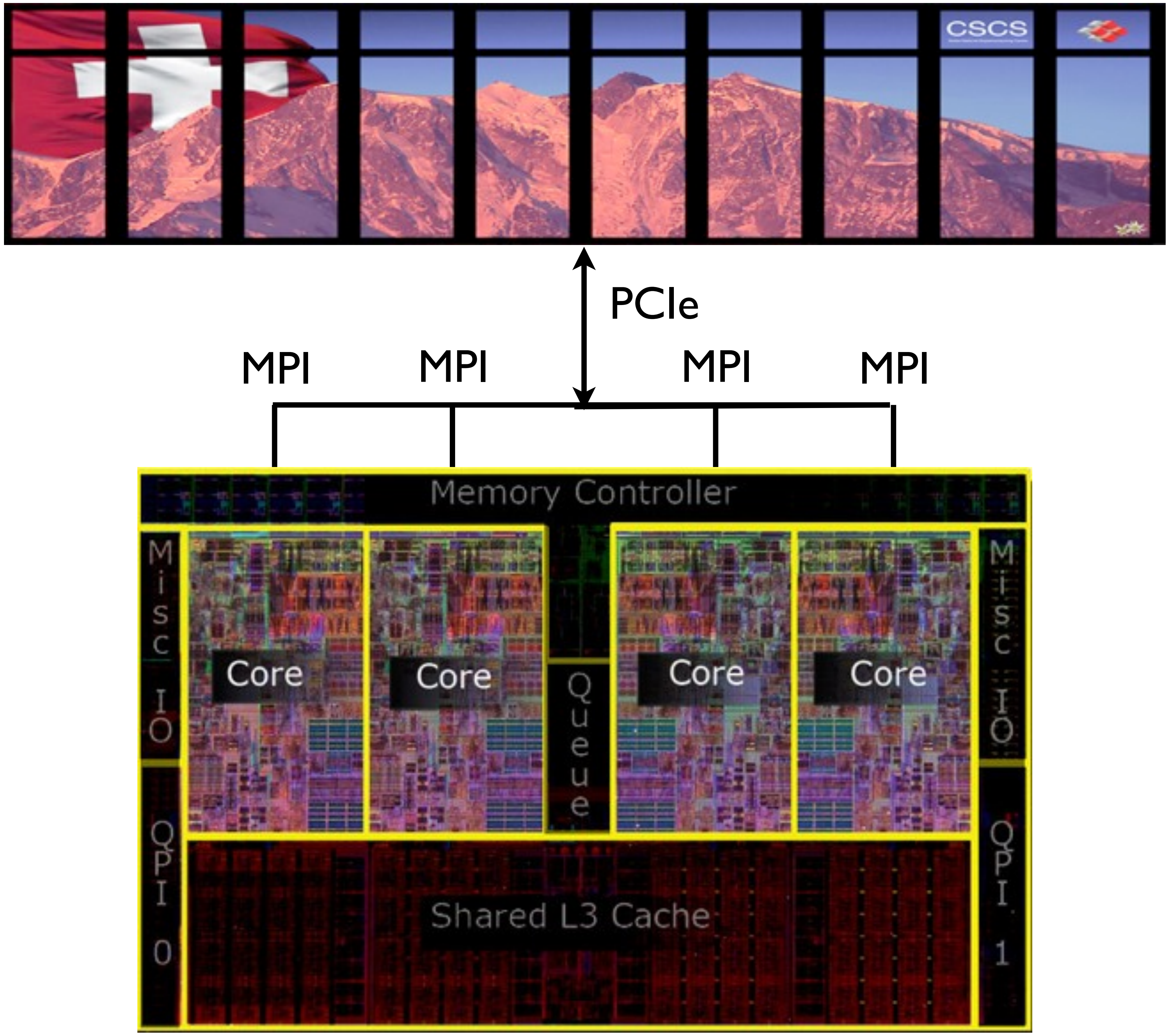
CPU Trends



After 2005, code performance scales wrt the number of core.



1st Level of Parallelism



Petascale computing

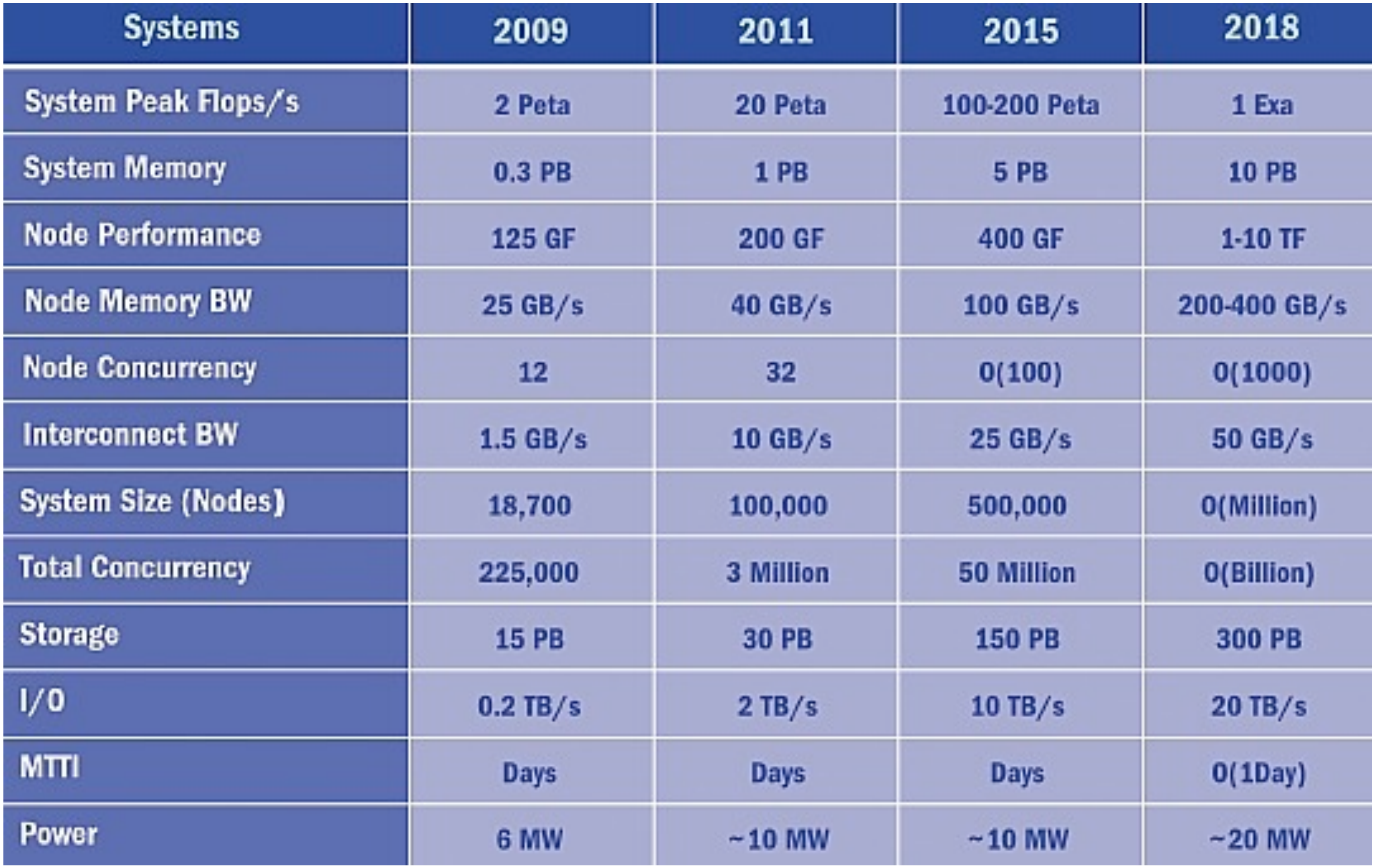
Petascale computing was driven by multi-core technology.

- MPI is at the center of Petascale technology
- Some software (scalability) and hardware (IO) had to be sorted out, but petascale science is working.

Is this good enough to reach Exascale computing ?

Unfortunately, we will hit a new road blocks.

Path to Exascale

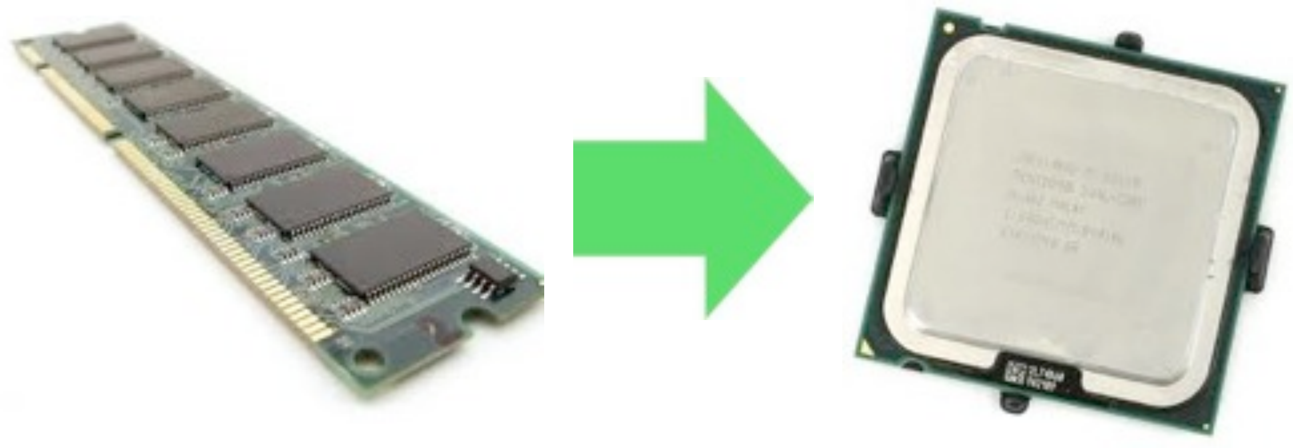


Systems	2009	2011	2015	2018
System Peak Flops/s	2 Peta	20 Peta	100-200 Peta	1 Exa
System Memory	0.3 PB	1 PB	5 PB	10 PB
Node Performance	125 GF	200 GF	400 GF	1-10 TF
Node Memory BW	25 GB/s	40 GB/s	100 GB/s	200-400 GB/s
Node Concurrency	12	32	O(100)	O(1000)
Interconnect BW	1.5 GB/s	10 GB/s	25 GB/s	50 GB/s
System Size (Nodes)	18,700	100,000	500,000	O(Million)
Total Concurrency	225,000	3 Million	50 Million	O(Billion)
Storage	15 PB	30 PB	150 PB	300 PB
I/O	0.2 TB/s	2 TB/s	10 TB/s	20 TB/s
MTTI	Days	Days	Days	O(1Day)
Power	6 MW	~10 MW	~10 MW	~20 MW

Power Consumption

Question: what is faster?

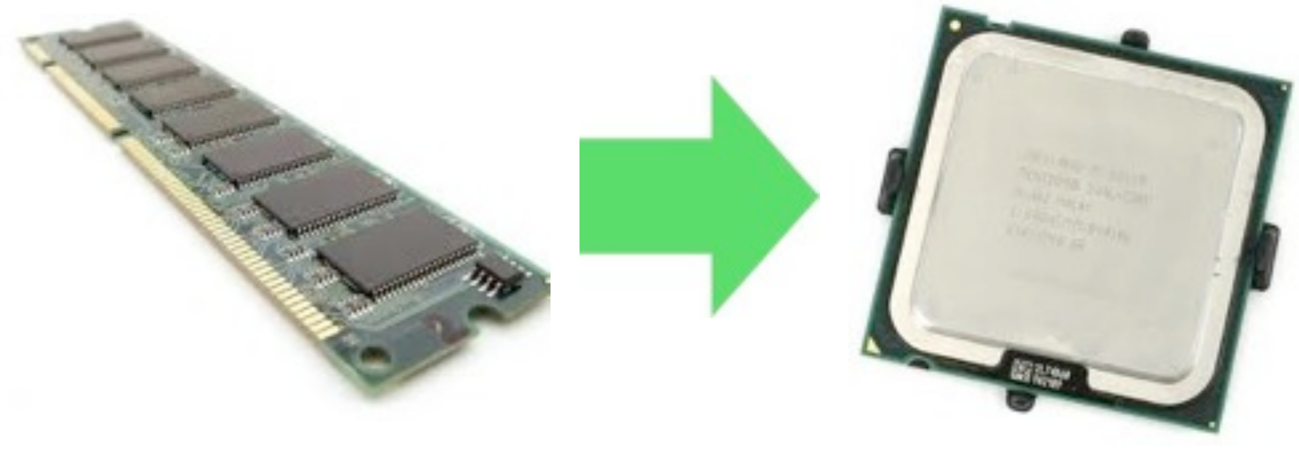
1.23456789101112 +
3.1415 *
1.12111098765432 +



Power Consumption

Question: what is faster?

1.23456789101112 +
3.1415 *
1.12111098765432 +



- ~1 ns

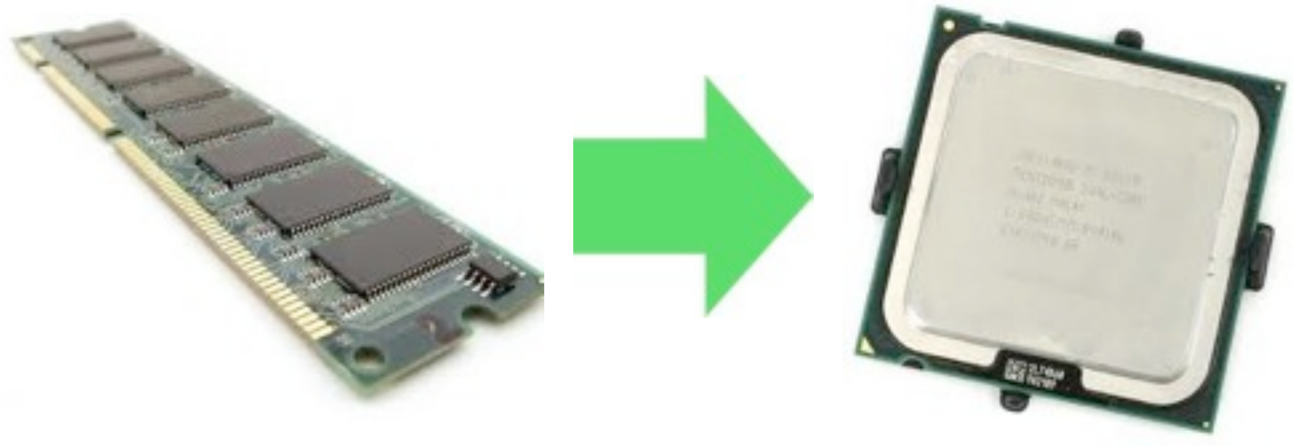
~100 ns

Question: what is the more power hungry?

Power Consumption

Question: what is faster?

1.23456789101112 +
3.1415 *
1.12111098765432 +



- ~1 ns ~100 ns

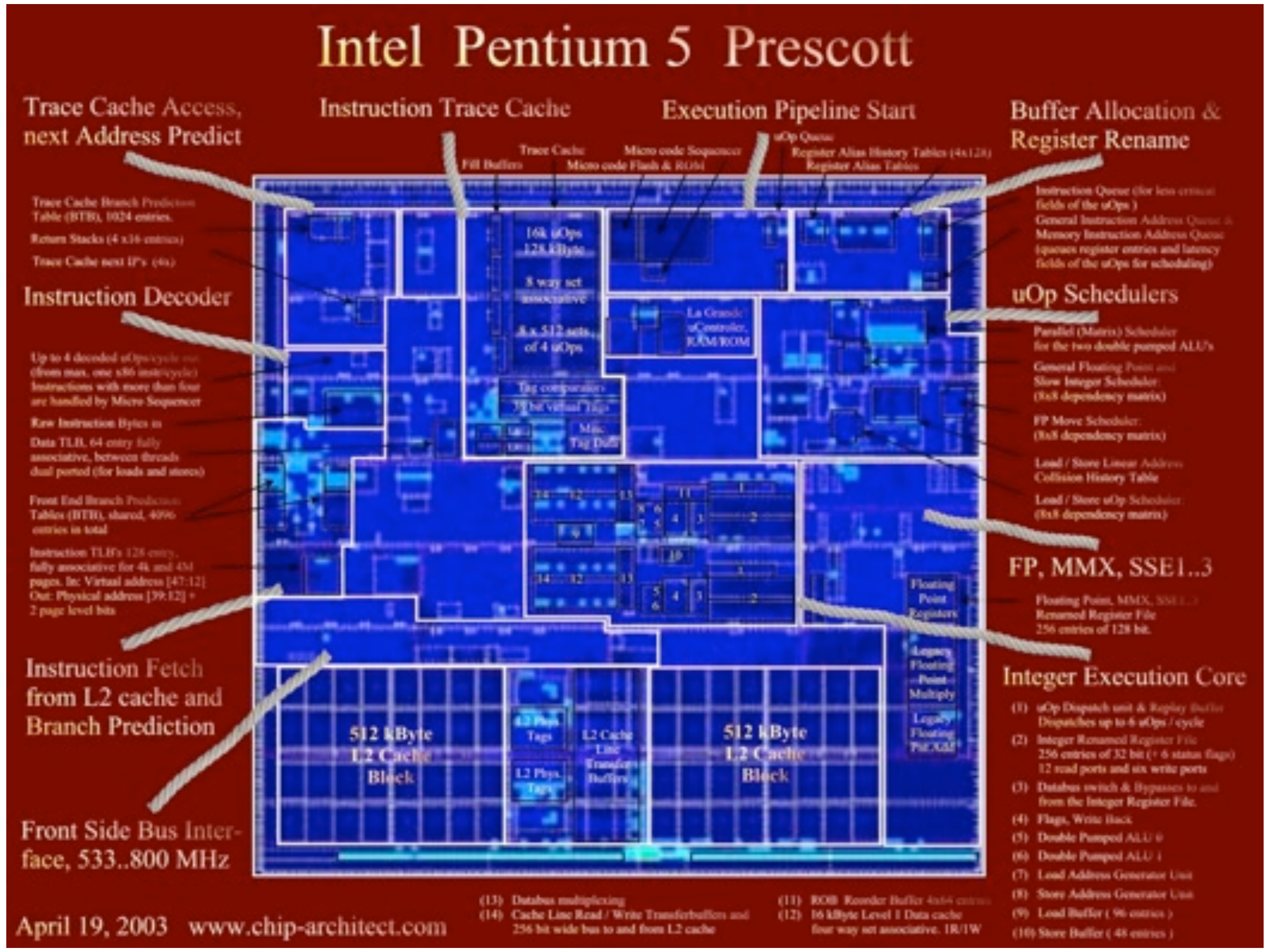
Question: what is the more power hungry?

- X1 X3

Conclusion: performance is not a question of flops but data movements, which are very expensive (time and power consumption)!

Power Consumption

On a CPU, ~70% of the power consumption is memory management

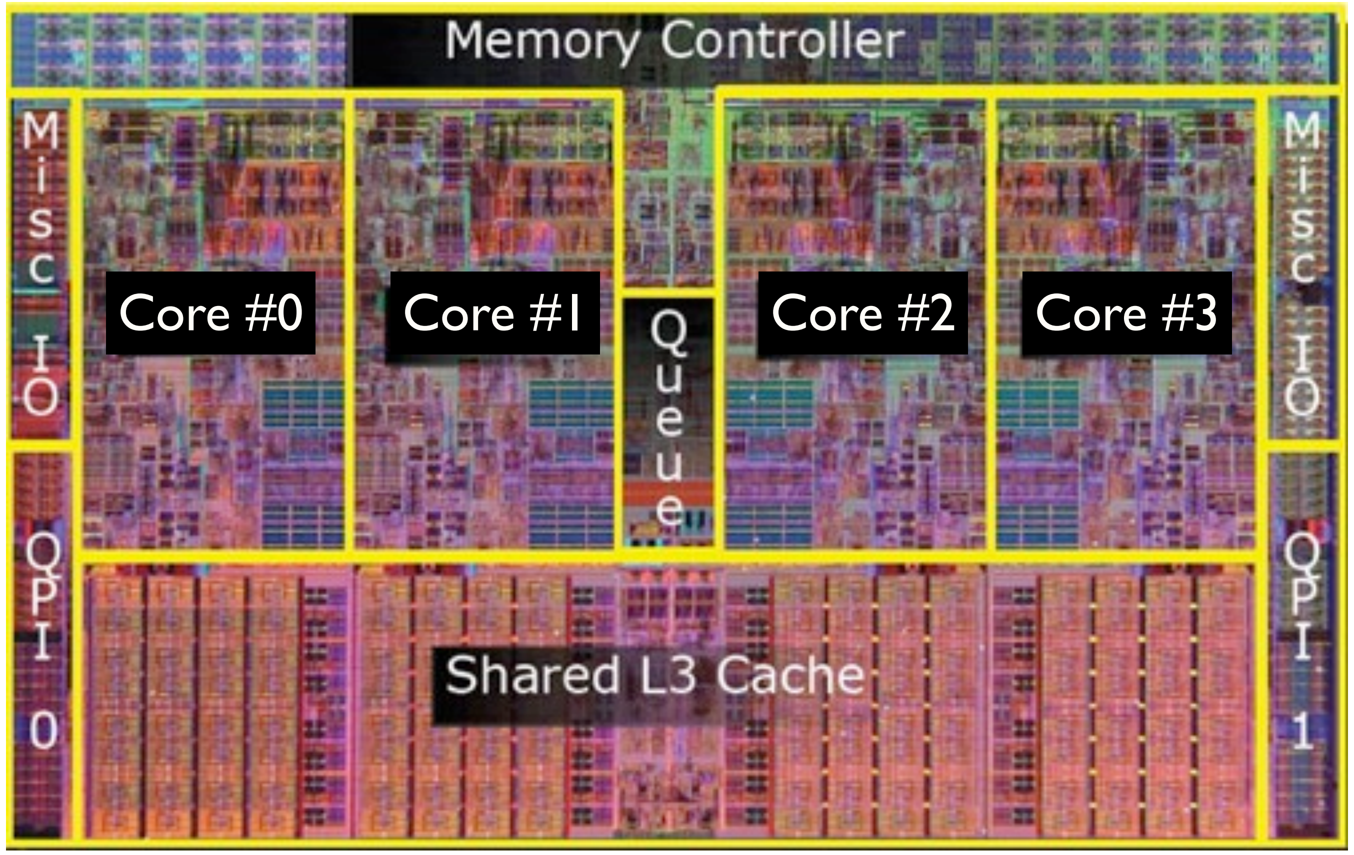

Intel Pentium 5 Prescott
Trace Cache Access, next Address Predict
Instruction Trace Cache
Execution Pipeline Start
Buffer Allocation & Register Rename
Instruction Decoder
uOp Schedulers
FP, MMX, SSE1..3
Integer Execution Core
Instruction Fetch from L2 cache and Branch Prediction
Front Side Bus Interface, 533..800 MHz
512 kByte L2 Cache Block
512 kByte L2 Cache Block
April 19, 2003 www.chip-architect.com

A solution? Not sure yet but high performance/power consumption ratio SIMD accelerators (GPU, MIC...) seem to be the way to go...

2nd Level of Parallelism

MPI PCIe

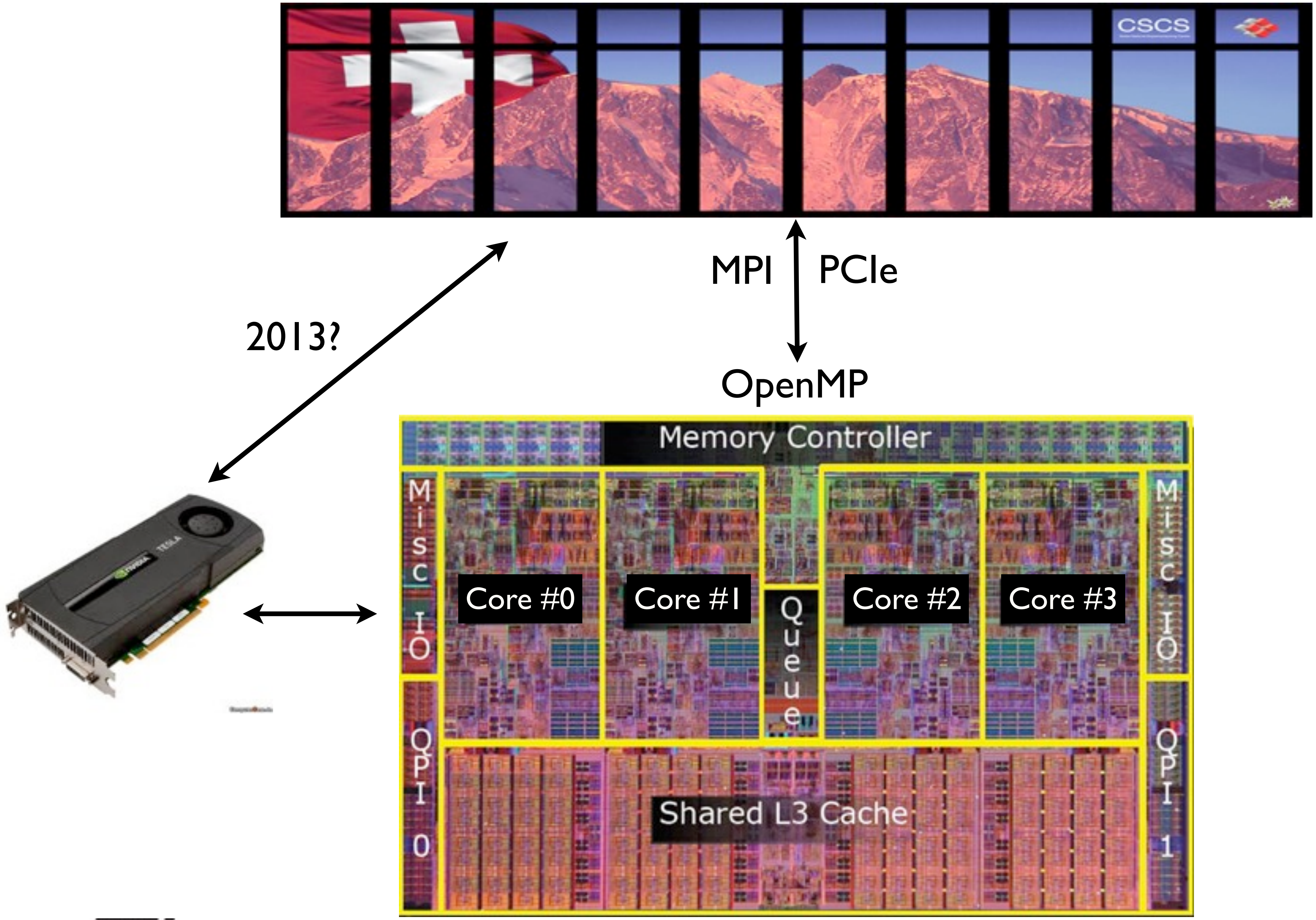
OpenMP



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

3rd Level of Parallelism



Exascale Brutal Facts

- Exascale scientific codes will have to:
  - to deal with millions (poss. billions) of threads,
  - use less MPI taks, possibly one per node,
  - less and slower memory per thread,
  - Small improvements in inter-processor and inter-thread communications,
  - Stagnant I/O subsystems,
  - take advantage of accelerators such as GPUs/manycore to maximize flops/watt,
  - redesign the code to use Hardware/Software co-degisn, such as heterogeneous models,
  - extract 3 levels of parallelism: MPI, OpenMP and vectorization (SIMD units, GPUs, ...)

Using MSPAI: Goals

The primary goal of MSPAI is to provide a High Performance preconditioner for the HP2C project LifeV, which is:

- a C++ open source HPC Library for Finite Elements computation, in particular for biomedical applications (FSI, multi-scale, multi-physics ...)
- developed at mainly EPFL (CH), MOX (IT), Emory Univ. (USA)

The serial core was developed at INRIA (FR), EPFL, MOX. The core parallel version was developed at EPFL (S. Deparis, G. Fourestey @CMCS, Pr. A. Quarteroni).

Lifev uses Trilinos for linear algebra solvers. See www.lifev.org .

Trilinos

"The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems."

- C++ based code,
- Matrix Classes, preconditioners, solvers ...
- a lot different packages, most of them are independant or interchangeable,
- complete MPI framework for communications.

Trilinos Preconditioners

Trilinos is using a very powerful class derivation system which allows the usage of third party codes (UMFPack, SuperLU ...)

Tilinos has several types of preconditioners:

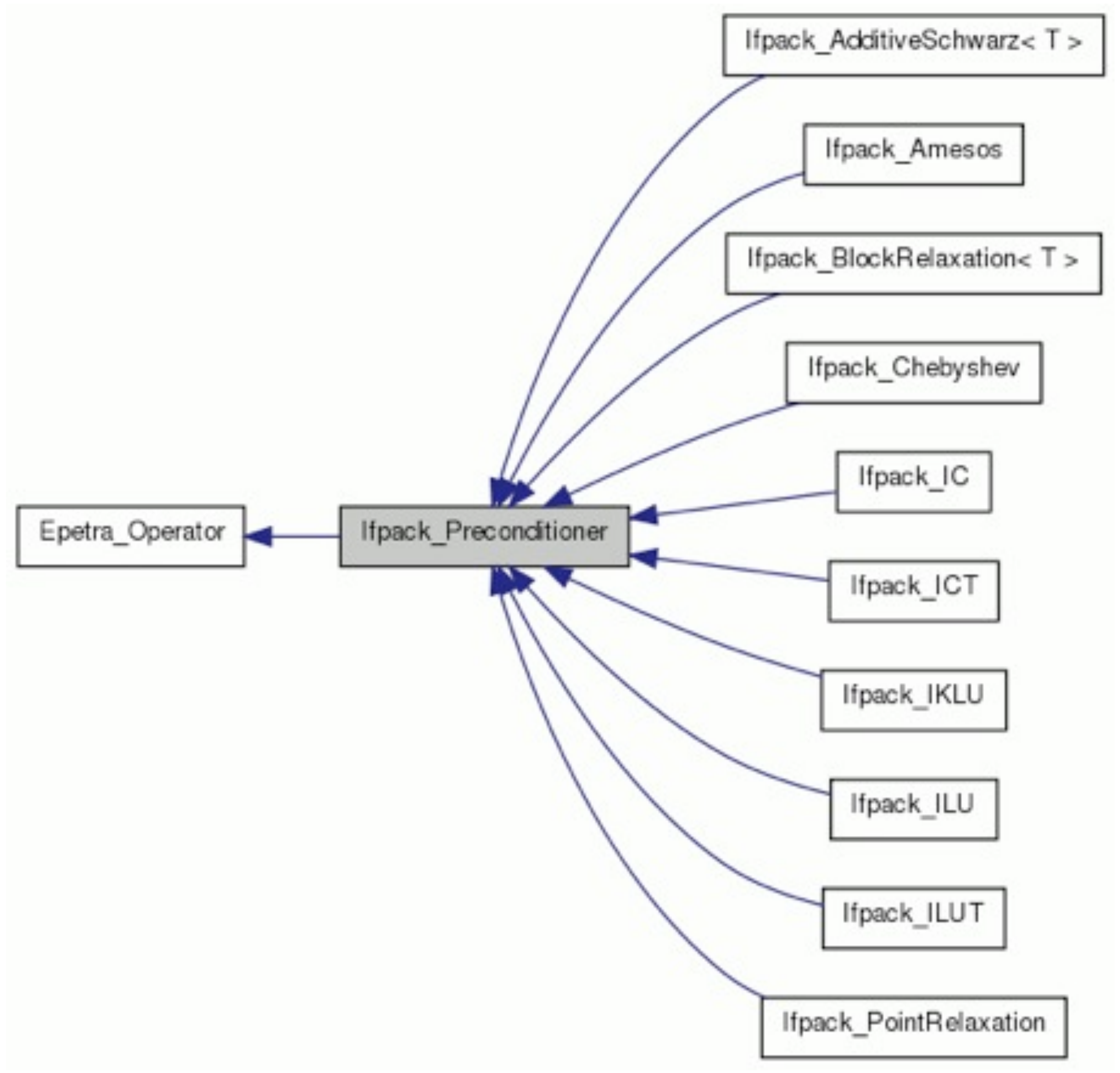
- Block preconditioners: Meros
- ILU factorization: Ifpack + AztecOO
- Multigrid methods: ML
- Domain decomposition: Ifpack

Ifpack

Point relaxation preconditioners

- Jacobi
- Gauss-Seidel
- symmetric Guass-Seidel

- Block relaxation preconditioners
  - Jacobi
  - block Gauss-Seidel
  - block symmetric Gauss-Seidel
- Point incomplete factorizations.
- Exact factorizations:
  - all Amesos classes can be used to compute the LU factorization of a given matrix.
- Chebyshev polynomials.



ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Swiss National Supercomputing Centre

Trilinos

Our goal is to create a new package, e.g Inverse Approximation, that could be used in the Trilinos preconditioners framework.

- General preconditioner used in Ifpack for domain decomposition,
- Schur complement approximation,
- Smoother for ML,
- ...

Now, let's link MSPAI and Trilinos.

MSPAI Implementation

The MSPAI code was standalone:

- it compiled an executable
- reading matrix and rhs in Matrix Market format
- the matrix columns were cut in contiguous chunks

First step: turn MSPAI code into a library

- the code compiled a library file as well as the executable
- Insert a namespace “mspai”

MSPAI Implementation

Second step: make the code accept another matrix format:

- Use Epetra_Map: each local map is saved in a file called <matrix name>.<rank #>
- Matrix values are read from the matrix market file then transformed into the local format/pattern.

Now, using EpetraExt::RowMatrixToMatrixMarket et Epetra_Map exporters, MSPAI can use Epetra_CrsMatrix matrices

MSPAI Implementation

Third step: modification in the data structure, each process has to know where each column is:

- each process is reading its own map file only!
- allreduce of this array (only once at the begining)

all processes are aware of their next columns as well as what process has a specific column.

Ifpack_Preconditioner Class

Public Member Functions:

- virtual int SetParameters (Teuchos::ParameterList &List)=0 Sets all parameters for the preconditioner.
- virtual int Initialize ()=0 ; Computes all it is necessary to initialize the preconditioner.
- virtual int Compute ()=0 ; Computes all it is necessary to apply the preconditioner.
- virtual bool virtual int ApplyInverse (const Epetra_MultiVector &X, Epetra_MultiVector &Y) const =0 ; Applies the preconditioner to vector X, returns the result in Y.

Ifpack_Preconditioner Class

Public Member Functions:

- virtual int SetParameters: store the MSPAI parameters (eps, level, ...),
- virtual int Initialize: tranforms the Epetra Matrix in the MSPAI format and performs the initial computations (Pattern, remote cols...),
- virtual int Compute: the actual computation calling the SPAI_Algorithm routine,
- virtual bool virtual int ApplyInverse: "applying" the preconditioner, i.e doing the matrix vector multiply:

M*(Ax - y)

Ifpack_Preconditioner Class

Now it is possible to dynamically cast this class into Epetra_Operator and use it as:

- a general preconditioner for AztecOO (Gmres),
- a smoother for ML,
- ...

Numerical Results

	SPAI			UMFPACK		
MPI ranks	Comp. Time	GMRES Time	GMRES iter.	Comp. Time	GMRES Time	GMRES Iter
16	4.3	7.76	107	5.05	5.3	25
32	1.6	1.72	107	1.61	2.67	28
64	0.7	0.98	107	0.62	1.40	29
128	0.37	0.55	107	0.27	0.73	31
256	0.21	0.32	107	0.11	0.42	34
512	0.14	0.23	107	0.04	0.25	37
1024	0.05	0.18	107	0.02	0.16	41
2048	0.03	0.23	107	0.02	0.12	45

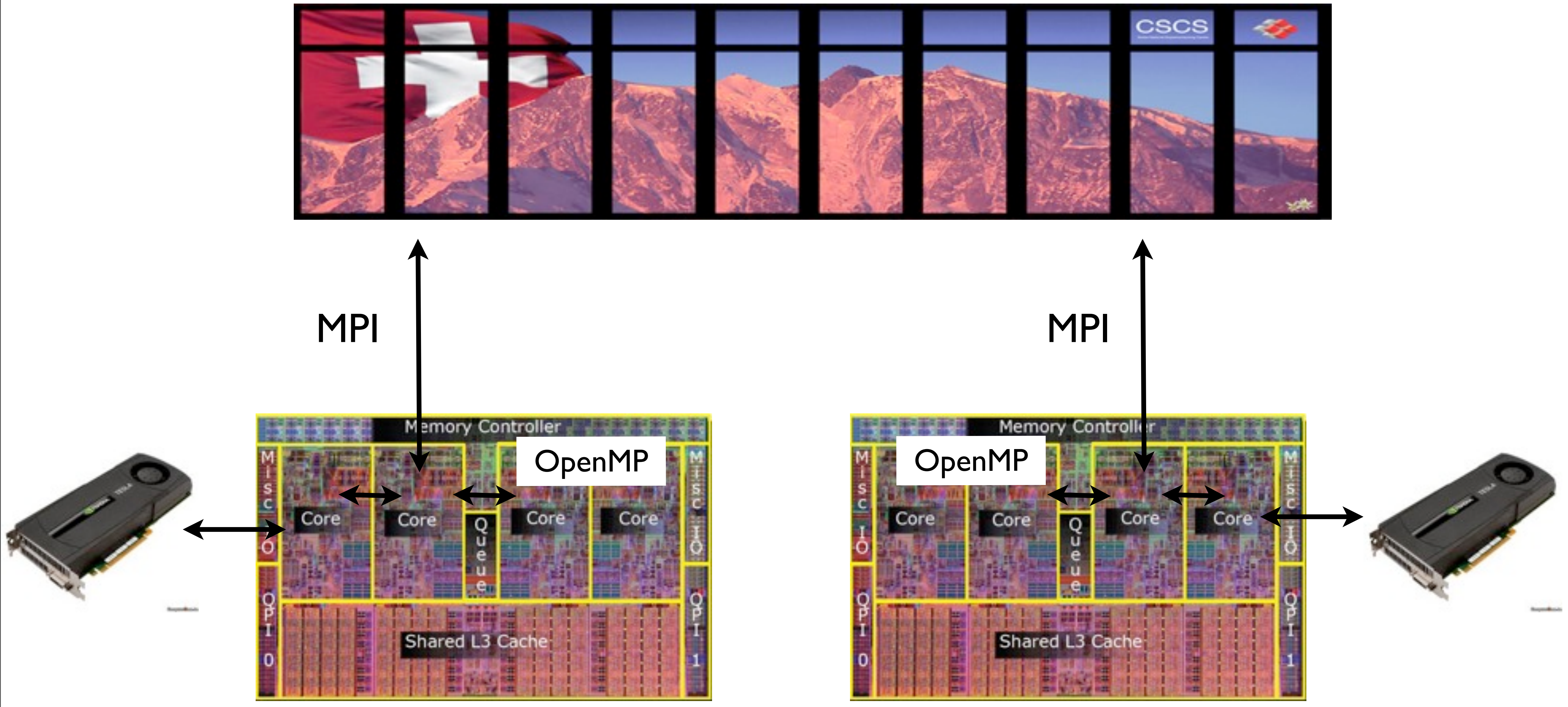
Comparison for AAS with MSPAI and UMFpack for the ADR problem, 216K dofs

Numerical Results

	SPAI			UMFPACK		
MPI ranks	Comp. Time	GMRES Time	GMRES iter.	Comp. Time	GMRES Time	GMRES Iter
16	69.13	11.27	283	5.28	7.94	11.27
32	46.66	7.74	272	2.91	6.46	7.74
64	37.15	5.65	262	2.16	6.18	5.65
128	32.15	4.81	260	1.47	5.38	4.81
256	15.12	2.61	253	0.61	2.83	2.61
512	12.38	2.24	253	0.42	2.47	2.24

Comparison for AAS with MSPAI and UMFpack for the NS problem, 48K dofs

3rd Level


CSCS
MPI
MPI
Memory Controller
Memory Controller
OpenMP
OpenMP
Core
Core
Core
Core
Core
Core
Core
Core
Queue
Queue
Misc IO
Misc IO
Misc IO
Misc IO
QPI 0
QPI 1
QPI 0
QPI 1
Shared L3 Cache
Shared L3 Cache

Threaded Implementation

So, instead of having one MPI process per core, let's have one per node using a threaded version.

Question: is MSPAI thread-safe?

Threaded Implementation

So, instead of having one MPI process per core, let's have one per node using a threaded version.

Question: is MSPAI thread-safe?

Answer: yes (well, sort ...)

Threaded Implementation

The only problem was a race condition:

```
unsigned int    *bitvec      = NULL,
                *reset_vec = NULL;
```

Making those arrays local inside Get_I_Set did the trick.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

Threaded Implementation

2 possible versions:

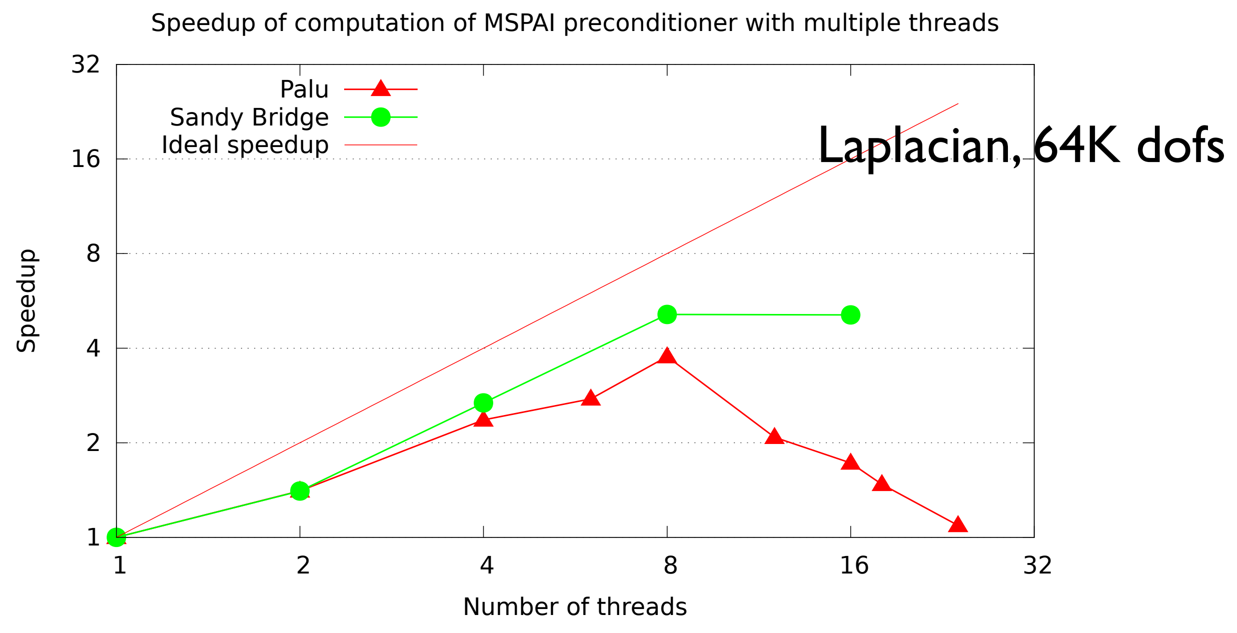
- #omp parallel region
- #omp task (OMP 3.0)

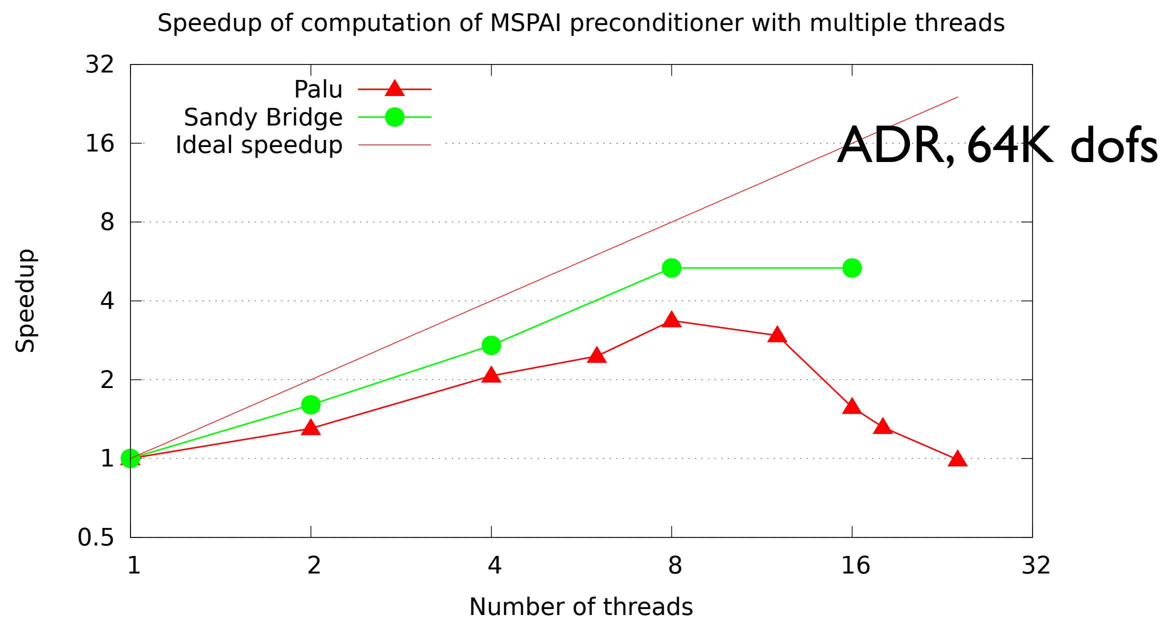
# threads	time	speedup	time task	speedup
1	20.60	1	23.12	1
2	11.33	1.81	13.49	1.71
3	7.83	2.63	7.89	2.93
4	6.02	3.42	5.78	4
5	4.57	4.5	4.67	4.95
6	3.93	5.24	3.98	5.81

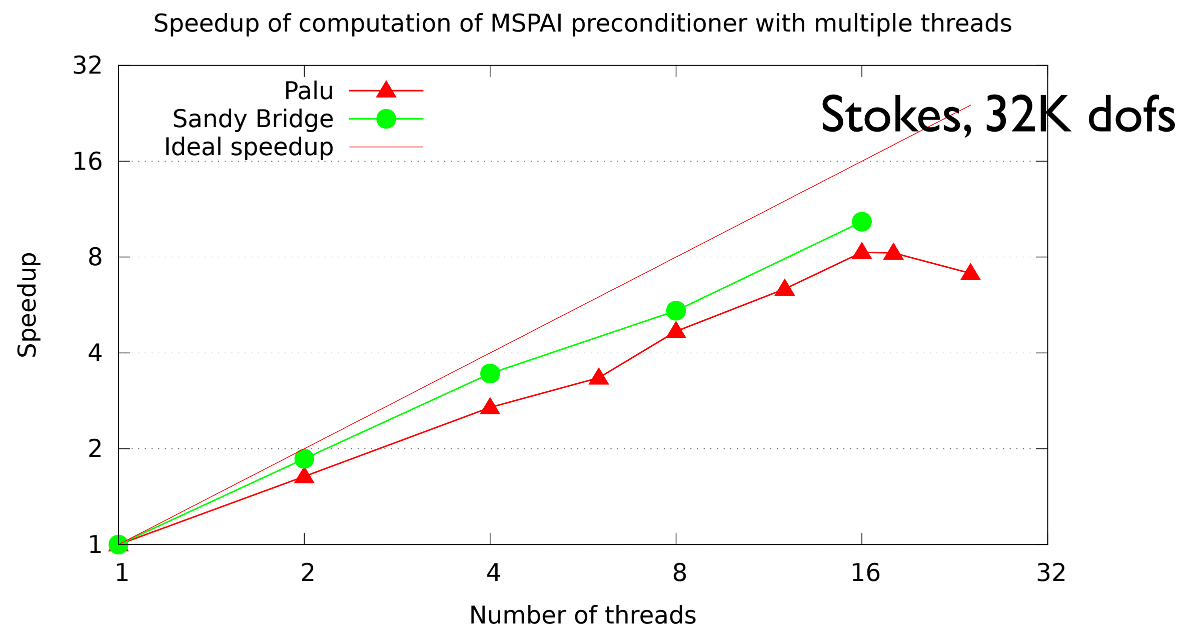
```
#pragma omp parallel private(col)
#pragma omp single
    {
        int             ncol = 0;
        while(col >= 0)
        {
            col = o_load.next_Col(A, M, B, P, UP);


            if (col >= 0)
            {
#pragma omp task untied
                {
                    ncol++;

                    SPAI_Column(A,
                            M,
                            B,
                            P,
                            UP,
                            U_UP,
                            col,
                            epsilon_param,
                            maxnew_param,
                            max_impr_steps,
                            ht,
                            use_mean,
                            pre_k_param,
                            pre_max_param,
                            bitvec,
                            reset_vec);
                }
            }
        }
    }
```

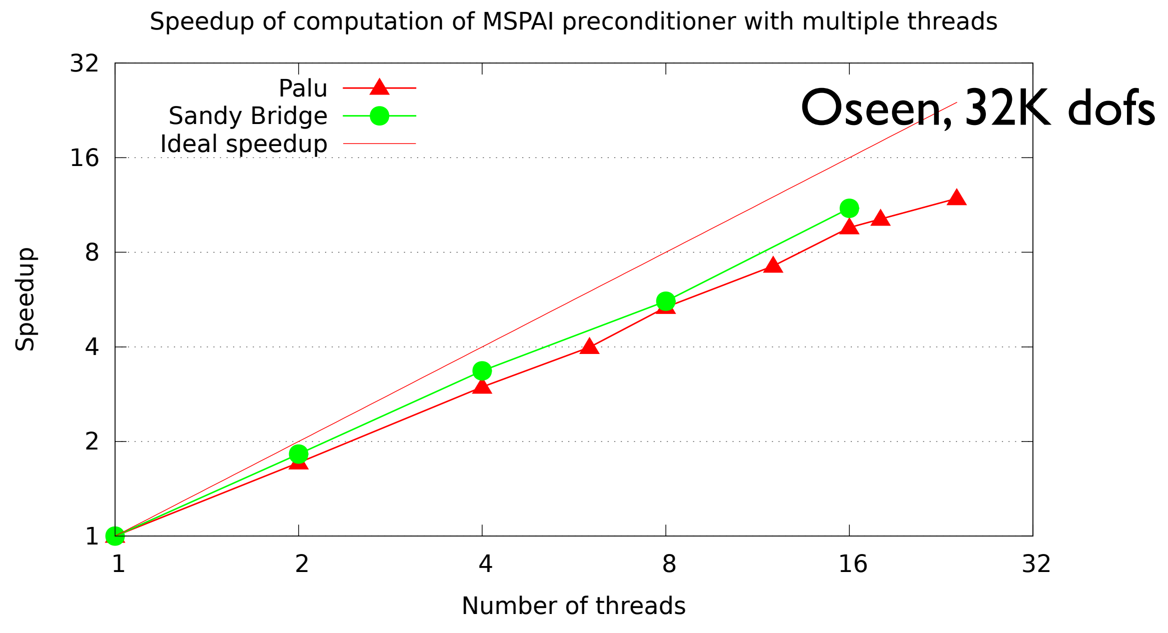
Threaded Results

Test systems: Palu@CSCS Magny-Cours, 24 cores, and Sandy Bridge, 16 cores.









ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

CUSP: Sparse Lin. Alg. for GPUs

```
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    cusp::krylov::cg(A, x, b);

    return 0;
}
```

CUSP: Sparse Lin. Alg. for GPUs

- CUda SParse: a highly templated library for GPUs and CPUs, providing a high level interface that hides GPU complexities

```
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    cusp::krylov::cg(A, x, b);

    return 0;
}
```

CUSP: Sparse Lin. Alg. for GPUs

- CUda SParse: a highly templated library for GPUs and CPUs, providing a high level interface that hides GPU complexities

- Uses Thrust, a C++ standard template library for GPUs

```
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;

    // load a matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

    // allocate storage for solution (x) and right hand side (b)
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);

    // solve the linear system A * x = b with the Conjugate Gradient method
    cusp::krylov::cg(A, x, b);

    return 0;
}
```

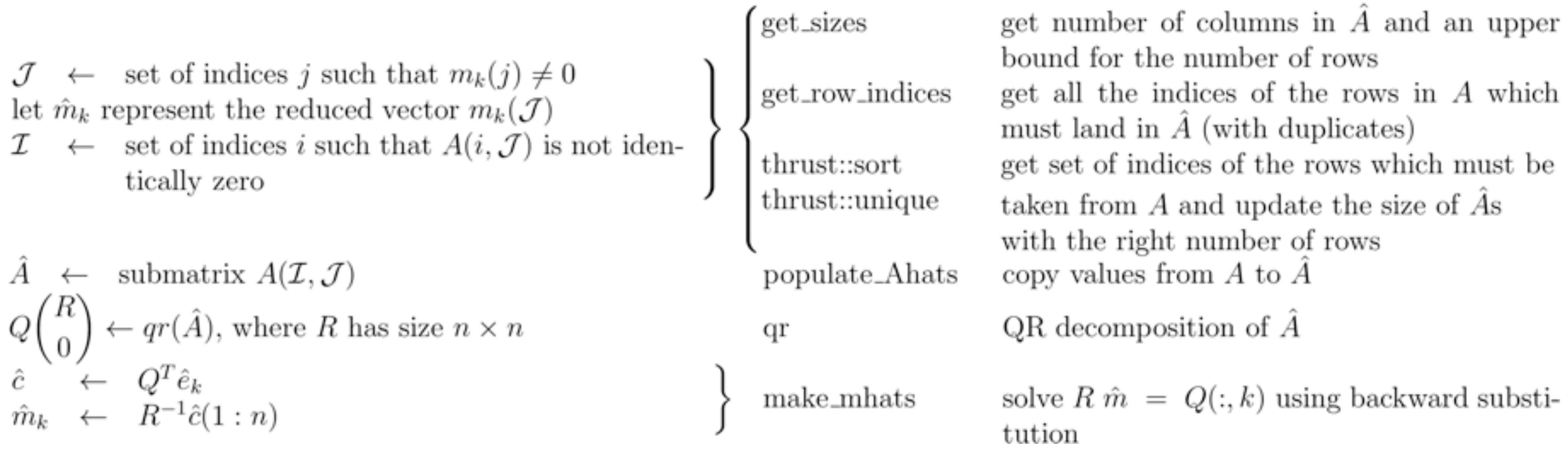
Goals of student project

- CUSP implementation of SPAI for a fixed sparsity pattern
- Evaluate CUSP implementation of the QR factorization
- Incorporate SPAI into GMRES solver implemented in another student project
- Evaluate GPU performance of optimized SPAI

```
        std::cout << "\nSolving with SPAI preconditioner" << std::endl;
// allocate storage for solution (x) and right hand side (b)
        cusp::array1d<ValueType, MemorySpace> x(A.num_rows, 0);
        cusp::array1d<ValueType, MemorySpace> b(A.num_rows, 1);
              // set stopping criteria (iteration_limit = 1000, relative_tolerance = 1e-6)
        cusp::default_monitor<ValueType> monitor(b, 1000, 1e-6)
// setup preconditioner
        cusp::csr_matrix<IndexType, ValueType, MemorySpace> I;
        cusp::gallery::eye(I, A.num_rows, A.num_cols);
        cusp::precond::spai<ValueType, MemorySpace> M(A, A);
// solve
        cusp::krylov::bicgstab(A, x, b, monitor, M);
```

Fixed Sparsity SPAI

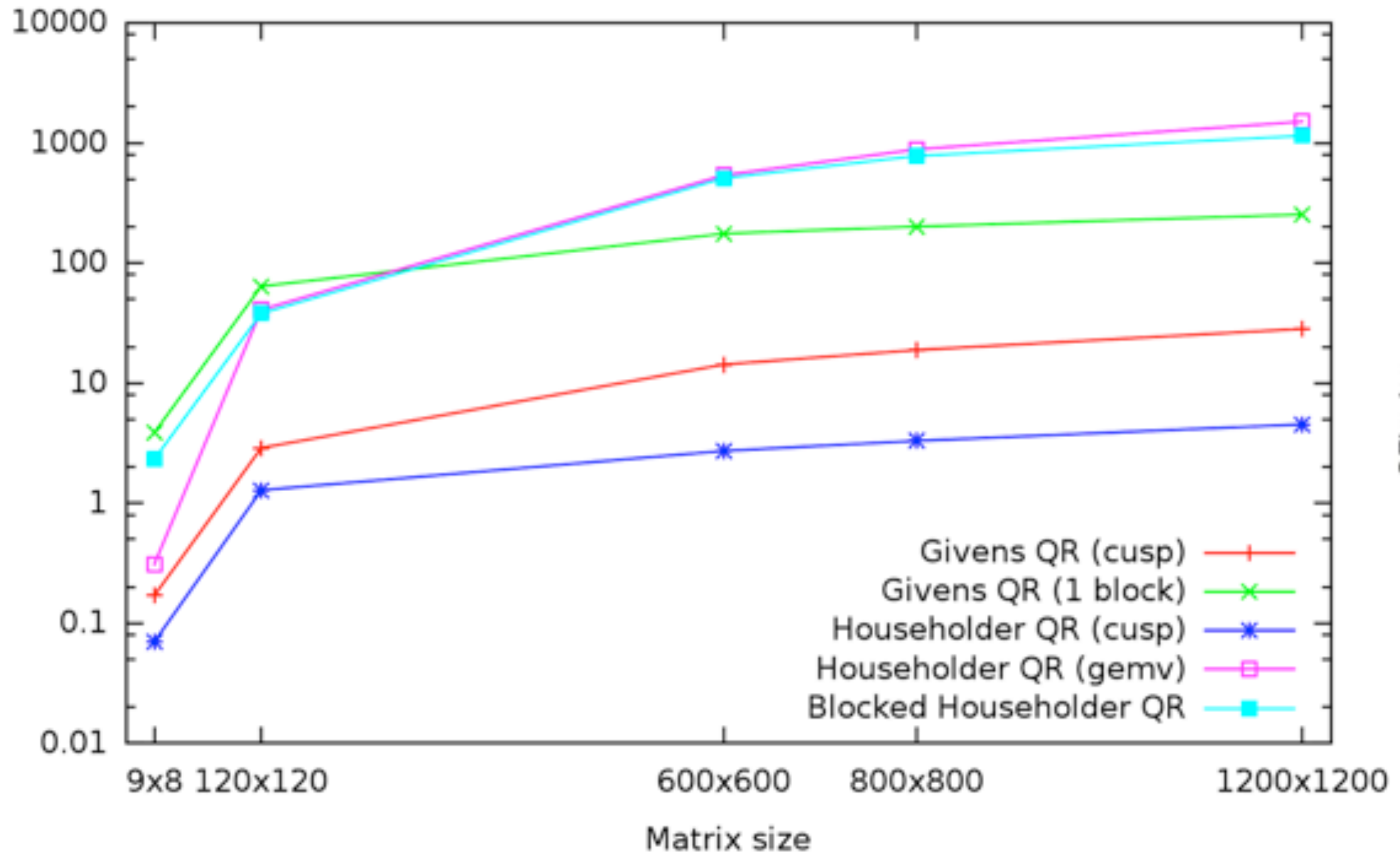
```
for each column k from 1 to n
     find sets of row and column indices
     build Â(k) matrix
     do QR decomposition of Â(k)
     compute $\hat{m}$
     put $\hat{m}$ into preconditioner matrix
end for
```

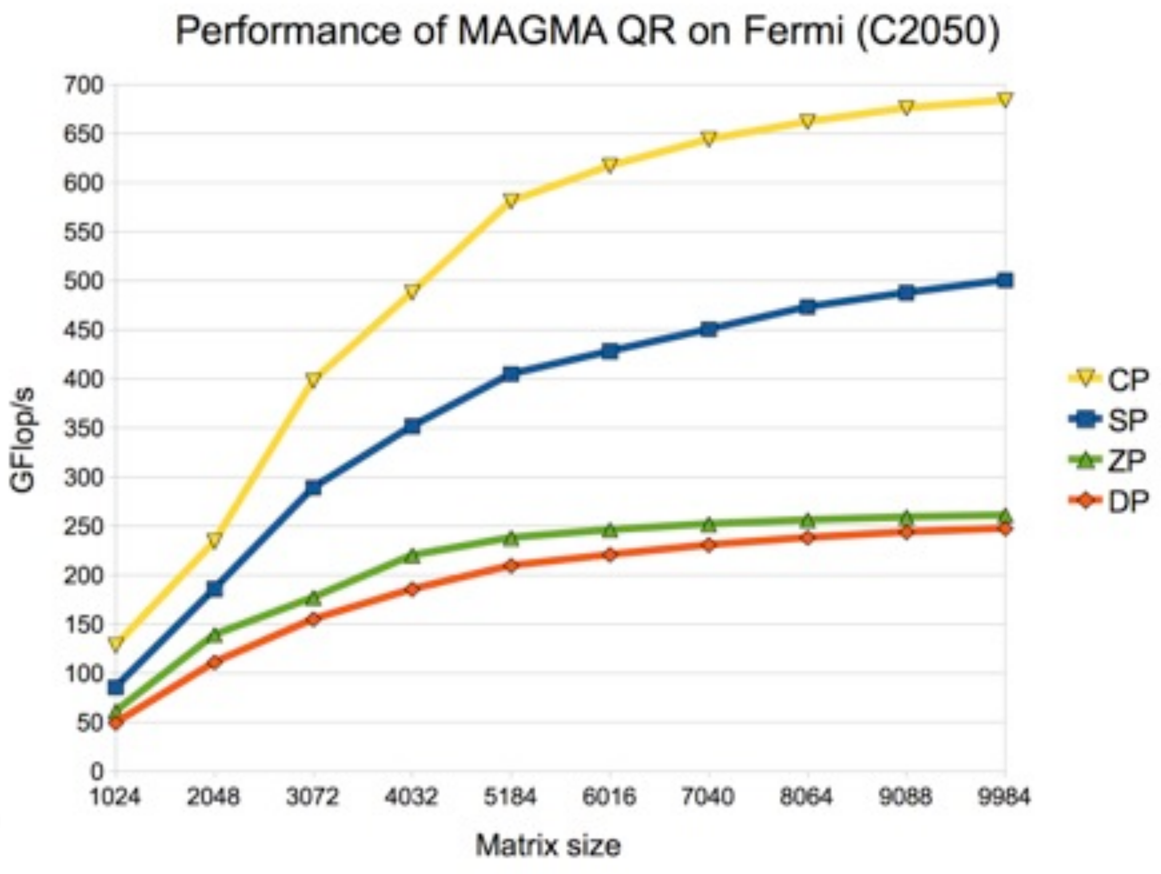


$\mathcal{J} \leftarrow$ set of indices j such that $m_k(j) \neq 0$ let $\hat{m}_k$ represent the reduced vector $m_k(\mathcal{J})$ $\mathcal{I} \leftarrow$ set of indices i such that $A(i, \mathcal{J})$ is not identically zero	get_sizes	get number of columns in $\hat{A}$ and an upper bound for the number of rows
	get_row_indices	get all the indices of the rows in A which must land in $\hat{A}$ (with duplicates)
	thrust::sort thrust::unique	get set of indices of the rows which must be taken from A and update the size of $\hat{A}$s with the right number of rows
$\hat{A} \leftarrow$ submatrix $A(\mathcal{I}, \mathcal{J})$	populate_Ahats	copy values from A to $\hat{A}$
$Q\begin{pmatrix} R \\ 0 \end{pmatrix} \leftarrow qr(\hat{A})$, where R has size $n \times n$	qr	QR decomposition of $\hat{A}$
$\hat{c} \leftarrow Q^T \hat{e}_k$ $\hat{m}_k \leftarrow R^{-1}\hat{c}(1:n)$	make_mhats	solve $R\,\hat{m} = Q(:,k)$ using backward substitution

QR implemented in CUSP

- QR performs well on GPU only for large matrices
- Still an order of magnitude slower than MAGMA
- SPAI requires many independent *small* QR factorizations



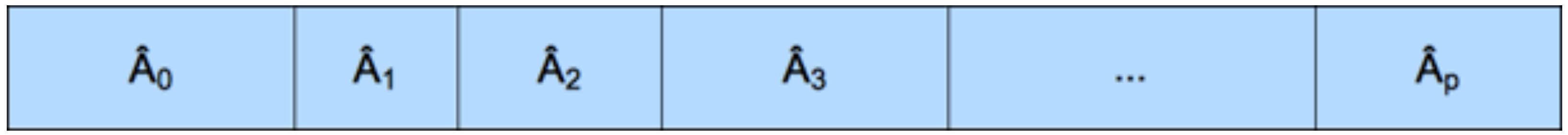


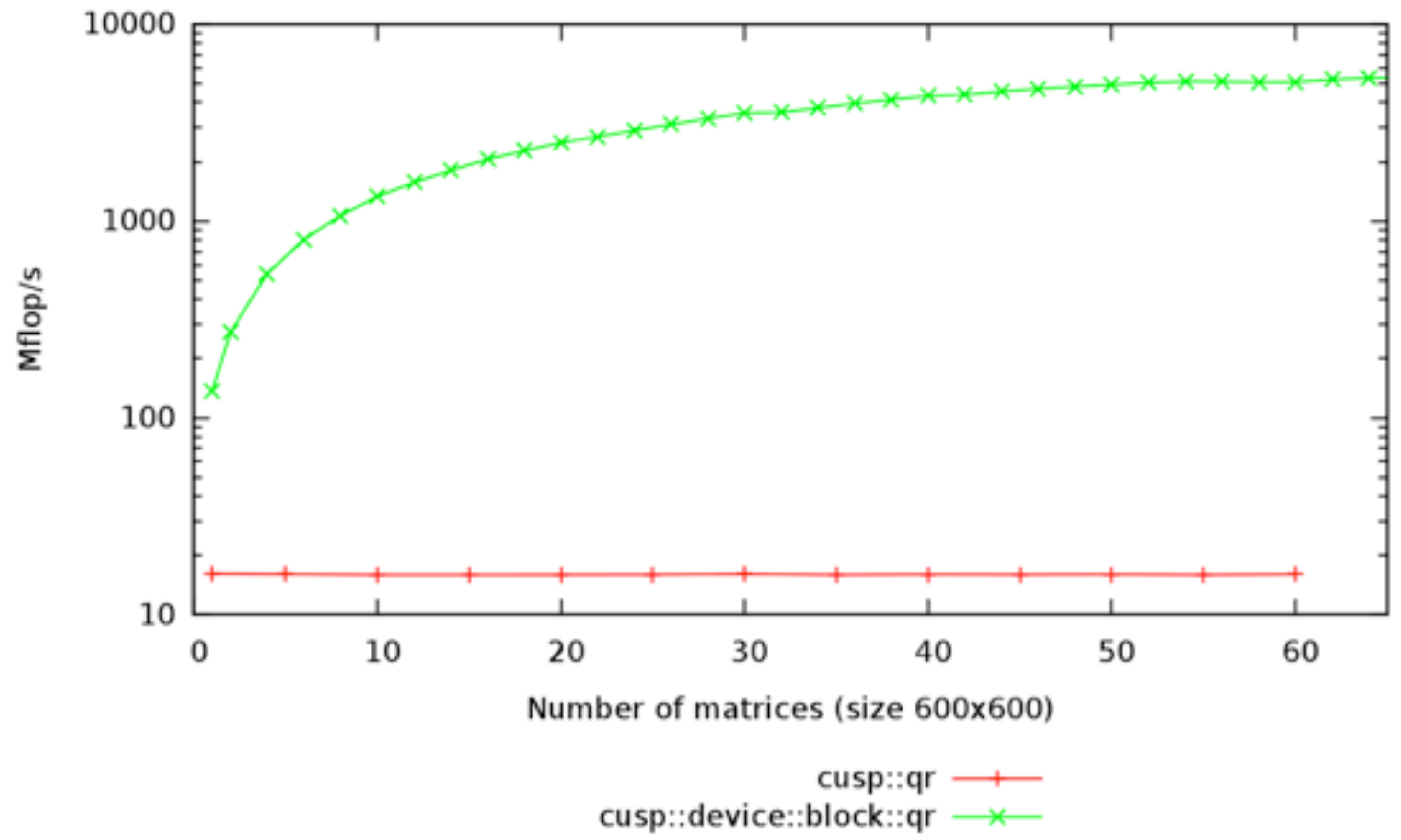
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

Memory Allocation Scheme

- Place a large number of $\hat{A}$ into a contiguous buffer
- Use a single thread block for each matrix





Present work

- First QR implementation calculated full Q, not just a row => memory problems. *Now fixed.*
- First implementation put *all* $\hat{A}$ into buffer => memory problems. Solution: performance plateaus at roughly 30 $\hat{A}$; use fixed size buffers, add only the as many $\hat{A}$ as will fit in the buffer. *Implementation ongoing.*
- Use MAGMA QR... BUT: MAGMA not currently capable of assigning thread blocks to individual matrices. *Discussion with MAGMA developers ongoing.*

Future Work

- Mix all 3 levels of parallelism:
  - 1 core for MPI communications
  - 1 core for accelerator communications
  - the rest for OpenMP

Overlap communications with computation:

- fetching rows while treating local rows, computing QR or LU ...
- use accelerator to compute all local QR/LU factorization
- any idea?

Future Work

- Have the GPU use a non-constant sparsity pattern,
- Use the GPU to compute QR factorizations by “glueing” Ahat matrices together,
- Use cashing, hashing and probing ...

Future Work

More general work:

- improve classes, templeting, derivation, abstraction design in general,
- use of Trilinos matrix class directly to avoid transformation cost?
- improve MSPAI as a general preconditioner,
- use MSPAI as a smoother for the ML package.

Thank You!

Any questions?...