ICT-Energy Community Workshop
Barcelona Supercomputing Center
April 23-25, 2014

Asynchronous iteration methods for GPU-accelerated HPC systems

Martin Wlotzka
Vincent Heuveline

Engineering Mathematics and Computing Lab (EMCL)
University of Heidelberg

	Top 500: Flops	Green 500: Flops/W
1	Tianhe-2 Xeon + Phi	TSUBAME-KFC Xeon + K20x
2	Titan Opteron + K20x	Wilkes Xeon + K20
3	Sequoia BlueGene/Q	HA-PACS TCA Xeon + K20x
4	K Computer SPARC	Piz Daint Xeon + K20x
5	Mira BlueGene/Q	romeo Xeon + K20x
6	Piz Daint Xeon + K20x	TSUBAME 2.5 Xeon + K20x

	Top 500: Flops	Green 500: Flops/W
1	Tianhe-2 Xeon + Phi	TSUBAME-KFC Xeon + K20x
2	Titan Opteron + K20x	Wilkes Xeon + K20
3	Sequoia BlueGene/Q	HA-PACS TCA Xeon + K20x
4	K Computer SPARC	Piz Daint Xeon + K20x
5	Mira BlueGene/Q	romeo Xeon + K20x
6	Piz Daint Xeon + K20x	TSUBAME 2.5 Xeon + K20x

Accelerators

Architecture	Manufacturer	Product
GPU = Graphics Processing Unit	nVIDIA	Tesla K20(x), Tesla K40
MIC = Many Integrated Cores	Intel	Xeon Phi

	Tesla K40	Tesla K20x	Tesla K20
double precision peak performance	1.43 TFlops	1.31TFlops	1.17 TFlops
single precision peak performance	4.29 TFlops	3.95 TFlops	3.52 TFlops
no. of CUDA cores	2,880	2,688	2,496
memory GDDR5	12 GB	6 GB	5 GB

Heterogeneous computing

- Success of GPU shows: There is use for <u>two</u> different core sizes in HPC

- <u>Big cores</u> and <u>little cores</u>
  - Big cores: CPU on host computer serial/irregular work
  - Little cores: GPU/MIC as accelerator parallel/regular work

- Little cores: more Flops per Watt and per €

Scientific computing

- Hardware peak performance is mostly not achieved by numerical simulations
- Challenge to transfer the computing power into algorithm performance
- Scalability suffers from synchronizations
- No implicit usage of energy-saving techniques

- **Bottleneck: numerical algorithms**

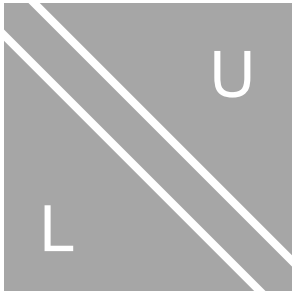
Exa2Green
energy-aware numerics

- Solvers for linear equation systems are the centerpiece of many numerical simulations:
  - nonlinear problems, implicit schemes
- Develop energy-efficient linear solvers
- Improve energy consumption of linear solvers by means of hardware-aware implementations
- Bridge the gap between parallelism inside algorithms and parallelism provided by the hardware

Splitting methods

- Matrix splitting

$$A = L + D + U =$$



- Linear equation system

$$(L + D + U)x = b$$
$$Dx = b - (L + U)x$$
$$x = D^{-1}b - D^{-1}(L + U)x$$

Jacobi method

$$x = D^{-1}b - \underbrace{D^{-1}(L+U)}_{\text{iteration matrix } B}x$$

Jacobi method $$x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij}x_j^k\right)$$

- parallel component updates within one iteration
- synchronization between iterations
- converges if $\rho(B) < 1$

Asynchronous iteration

Jacobi method $x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^k\right)$ update function
shift function

Asynchronous iteration

$$x_i^{k+1} = \begin{cases} \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^{k-s(k,j)}\right) & \text{if } i = u(k) \\ x_i^k & \text{if } i \neq u(k) \end{cases}$$

- introduce asynchronism, reduce communication
- parallel component updates, no synchronization
- converges if $\rho(|B|) < 1$

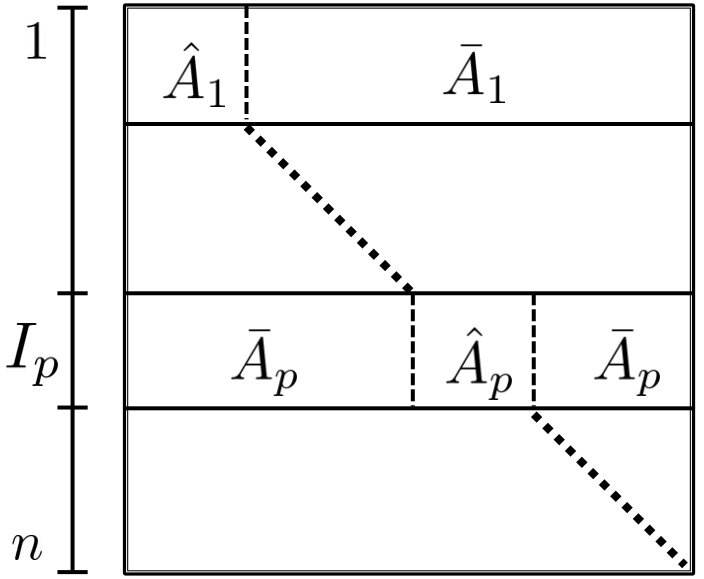
Block-asynchronous iteration

Block-asynchronous iteration

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \in I_p, j \neq i} a_{ij} x_j^k - \sum_{j \notin I_p} a_{ij} x_j^{k-s(k,j)}\right)$$

Block p:

- $\hat{A}_p$ diagonal part
- $\bar{A}_p$ off-diagonal part
- I_p index set



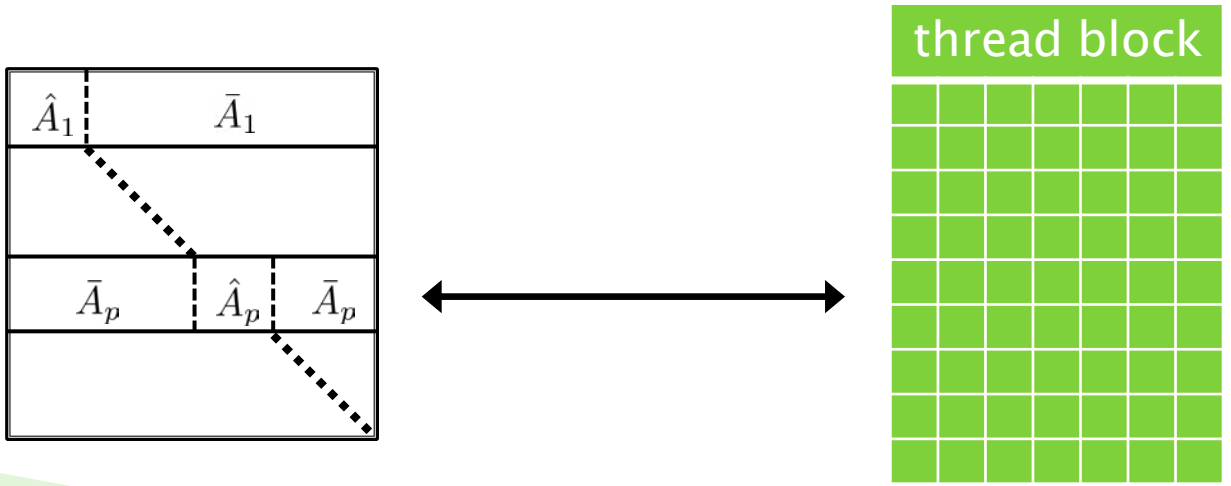
Exa2Green
energy-aware numerics

SEVENTH FRAMEWORK PROGRAMME

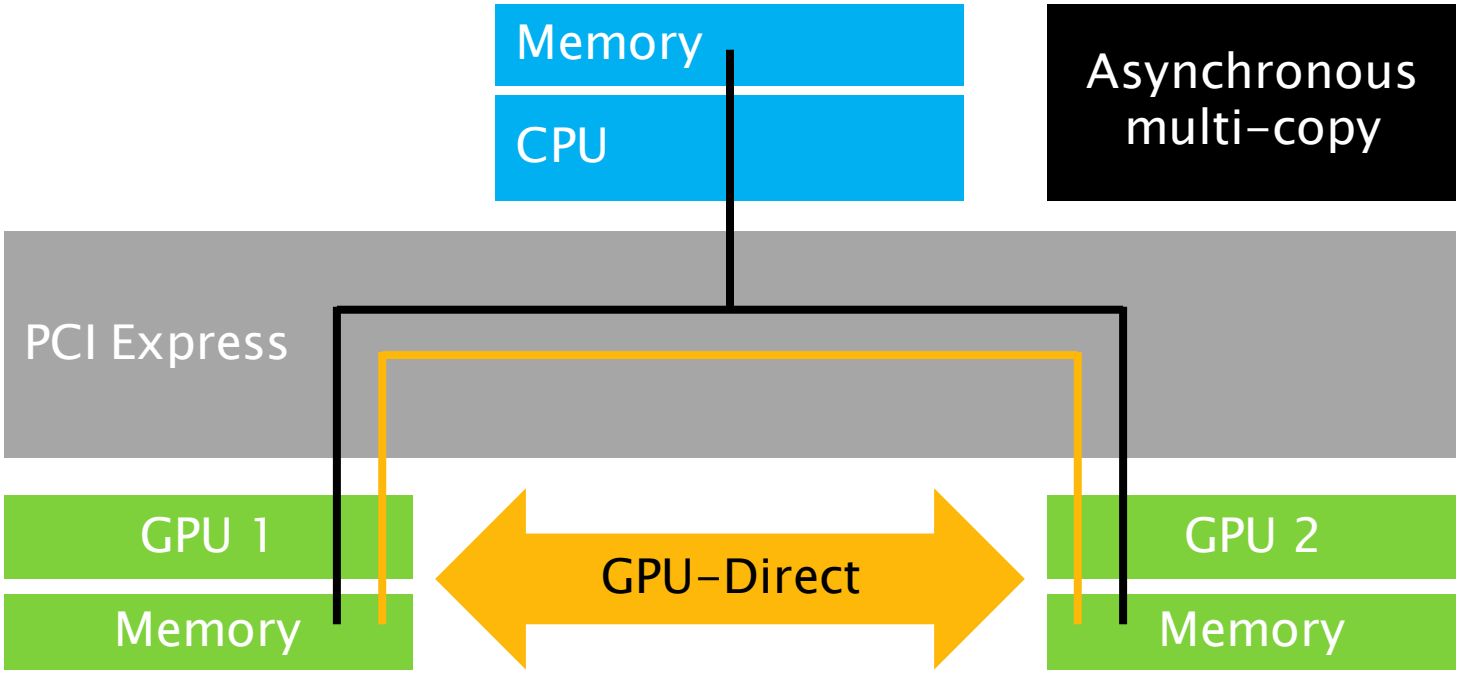
Block-asynchronous iteration on GPU

	SMX = streaming multiprocessor	cores per SMX	max. thread blocks per SMX	max. threads per thread block
Tesla K40	15	192	16	1024

- matrix blocks correspond to thread blocks
- synchronous Jacobi iteration within the thread block
- asynchronous iteration with other thread blocks

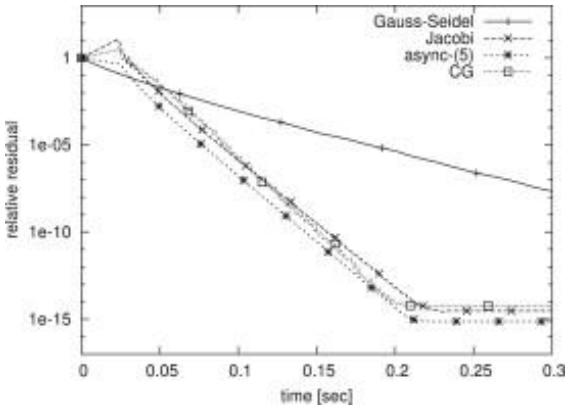


Multi-GPU asynchronous iteration

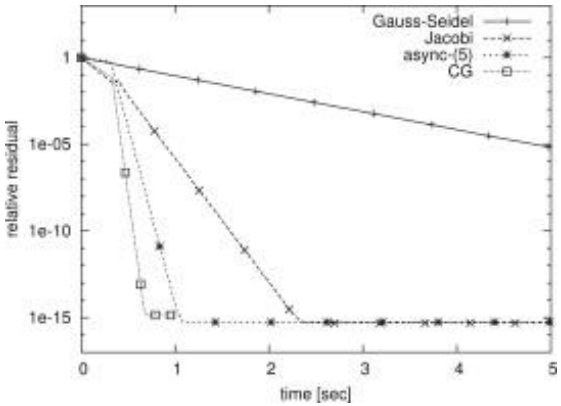


Numerical experiments

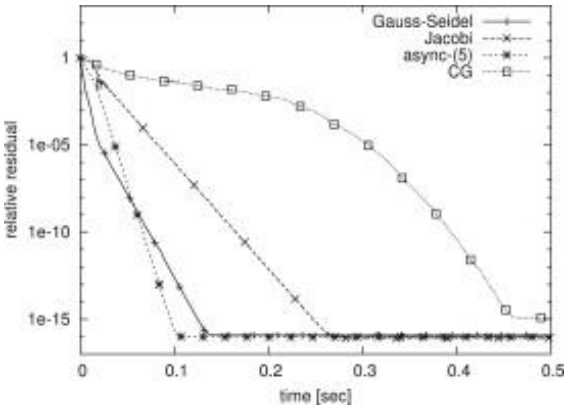
Statistics



2D/3D problem



Combinatorics



Time to accuracy plots

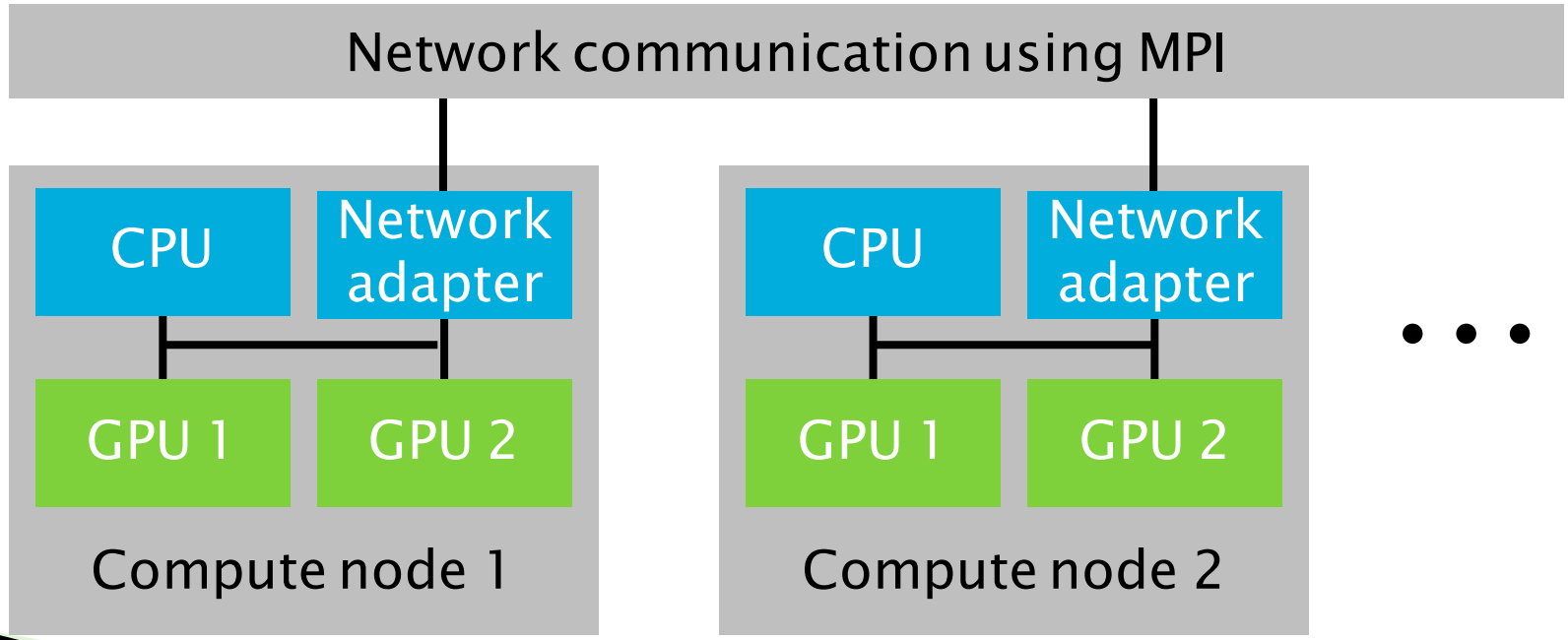
Anzt, Tomov, Dongarra, Heuveline
A block-asynchronous relaxation method for graphics processing units
J. Parallel and Distributed Computing, 2013

Exa2Green
energy-aware numerics

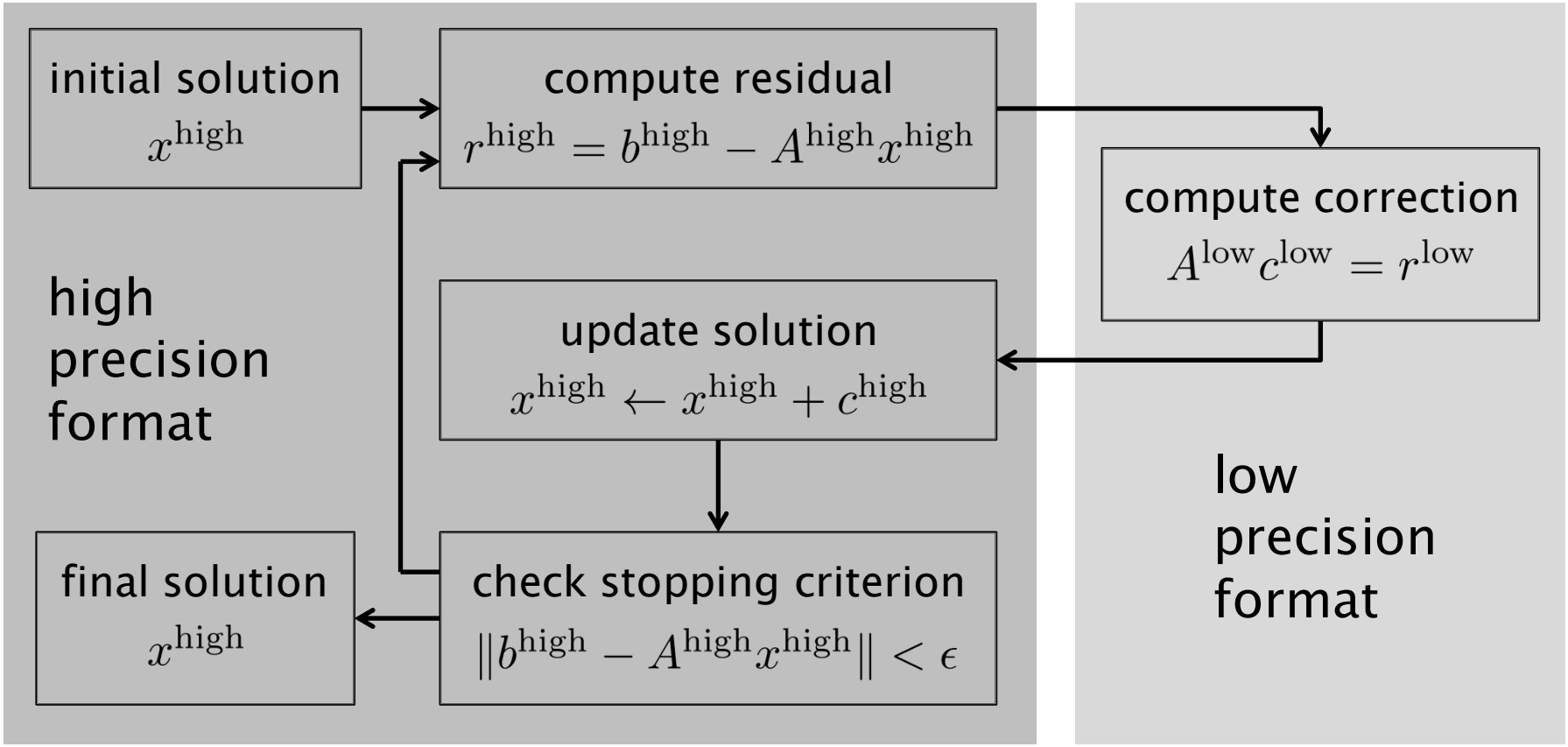
SEVENTH FRAMEWORK PROGRAMME

Multi-GPU implementation for multiple compute nodes

- Multiple compute nodes, each equipped with GPUs, are connected over the network using MPI
- An additional layer of asynchronism is added to the algorithm

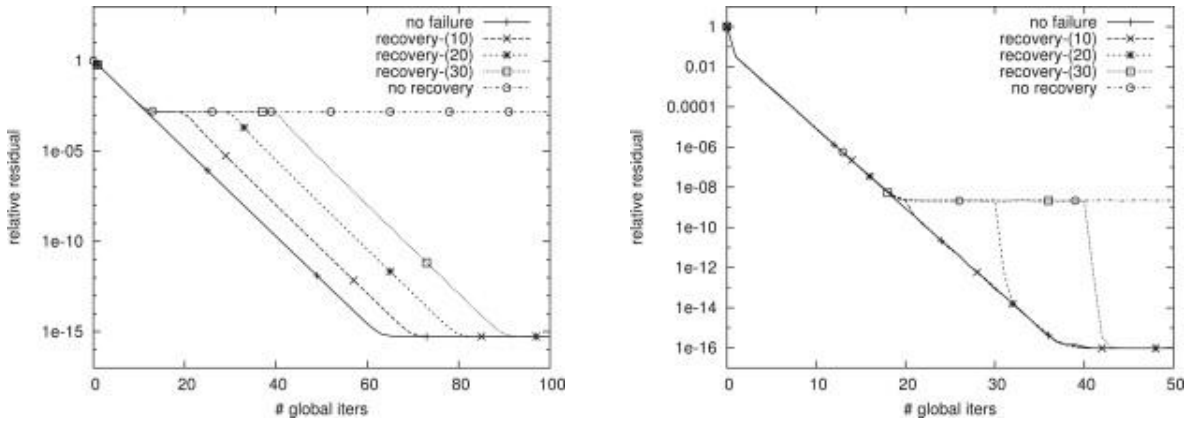


Mixed precision iterative refinement



Fault tolerance

- Increasing number of hardware components
- Increasing probability of hardware failure



Anzt et al., 2013

- Asynchronous iteration is resilient to detectable hardware failure by nature

On the road to exascale... How can devices save power?

Two-tiered parallelism:

- Lower level: lock-step execution (SIMD, SIMT)
  - CPU and MIC: vector registers, SSE → AVX
  - GPU: cores/warps (Tesla K40: 192 wide)
- Higher level: independent execution
  - CPU and MIC: cores (Xeon: 8, Phi: 60)
  - GPU: streaming multiprocessors (Tesla K40: 15)
- Decoding instructions consumes power
- Lock-step execution reduces number of decodings
- Fused multiply add (FMA)
- More lock-step, less independent cores

On the road to exascale... Will we need to handle billions of cores in the near future?

- "Core" has different meaning for different architecture
- An SMX on a GPU is comparable to a CPU/MIC core
  - both act independently
- GPU cores are comparable to vector lanes
  - both act in lock-step
- Tesla K40 has 2,880 cores, but only 15 SMX
- If you count independent compute units, e.g. SMX and CPU/MIC cores, then there will not be billions

On the road to exascale... How will programming look like?

- Programming the K Computer (10 PFlops):
  - 88,000 nodes, 8 cores per node
  - MPI and OpenMP, C/C++ and Fortran
- Programming accelerators:
  - Threads (OpenMP or CUDA), C/C++ and Fortran

- Imagine the year 2020:
  - 2x nodes, each 5x more powerful: 100 PFlops
  - accelerators, 4x more powerful than CPU: 400 PFlops

On the road to exascale... How will programming look like?

- Programming the K Computer (10 PFlops):
  - 88,000 nodes, 8 cores per node
  - MPI and OpenMP, C/C++ and Fortran
- Programming accelerators:
  - Threads (OpenMP or CUDA), C/C++ and Fortran

0.5 Exaflops

- Imagine the year 2020:
  - 2x nodes, each 5x more powerful: 100 PFlops
  - accelerators, 4x more powerful than CPU: 400 PFlops

- Could we still use the four proven standards of HPC, namely MPI and threads, C/C++ and Fortran?