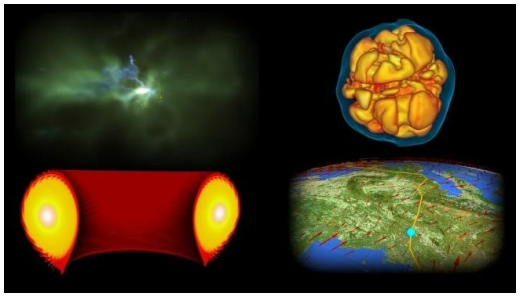
High-performance computing
– technological trends and programming challenges –

Markus Rampp

Max Planck Computing and Data Facility (MPCDF)



HPC applications group

Topics

where we are – technologically - in scientific high-performance computing (HPC)

where we are heading for: technological evolution and programming challenges

- *extreme parallelism*
- *heterogeneous hardware*
- *hardware reliability*

… with some practical notes, hints and opinions

Introduction

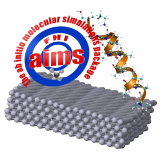
About the Max Planck Computing and Data Facility (MPCDF)

- HPC and data center of the Max Planck Society, a publicly funded organization for basic research *O*(1B €/y)
- formerly known as the RZG, located at the research campus in Garching near Munich, Germany
- MPCDF operates a Petaflop-class supercomputer and numerous mid-range clusters

HPC applications group provides high-level support for the development, optimization, analysis and visualization of HPC applications … + *technology watch*

original contributions and long-term support for optimization and porting of HPC codes developed in the Max Planck Society, e.g.

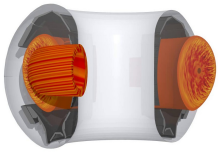
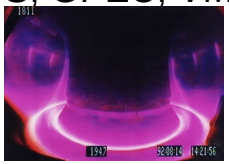
- FHI-aims, OCTOPUS, S/PHI/nX (materials and bio science), ELPA (eigensolver library)



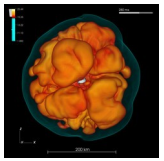
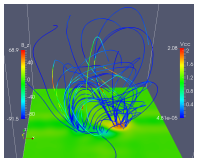
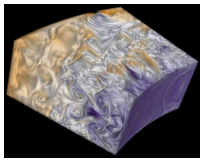

S/PHI/nX

ELPA

- GENE, SOLPS, GPEC, VMEC (plasma physics/fusion research)


- VERTEX, GOEMHD3, MagIC (astrophysics, geophysics), NSCOUETTE (comp. fluid dynamics)

Processor-technology evolution

"We show that, in the context of Moore's Law, overall productivity can be increased for large enough computations by `slacking' or waiting for some period of time before purchasing a computer and beginning the calculation."

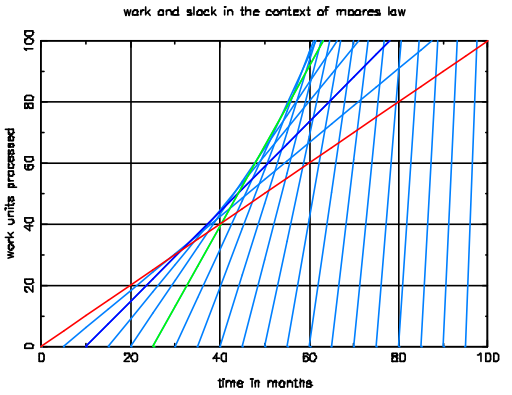
The Effects of Moore's Law and Slacking on Large Computations

Chris Gottbrath, Jeremy Bailin, Casey Meakin, Todd Thompson, J.J. Charfman
Steward Observatory, University of Arizona

Abstract

We show that, in the context of Moore's Law, overall productivity can be increased for large enough computations by `slacking' or waiting for some period of time before purchasing a computer and beginning the calculation.

Figure 1:



… those days are over …

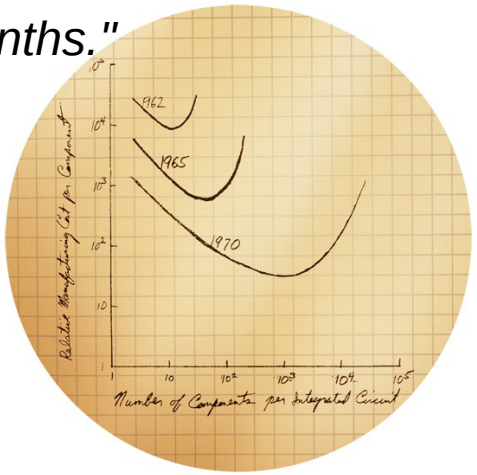
when ? → since around 2005
why ? → power limitations
what's next ? → exascale
how to prepare ? → be brave …
… and don't mourne the "good old days"

… exciting times are ahead !

Processor-technology evolution

Moore's law

"The number of transistors and resistors on a chip doubles every 18 months."
(Intel co-founder Gordon Moore, 1965)



=> performance of computers, applications ???

- from 60 devices (1965) to billions of devices (2010) per chip
- background:
  - Moore's law as a self-fulfilling prophecy
  - Dennard scaling: V~L, A~L => P~L^2 => P/A~const.
    (miniaturization at constant power density, currently 14nm)
  - P~f^3 => CPU clock frequency leveled off at ~ 3 GHz (2005)

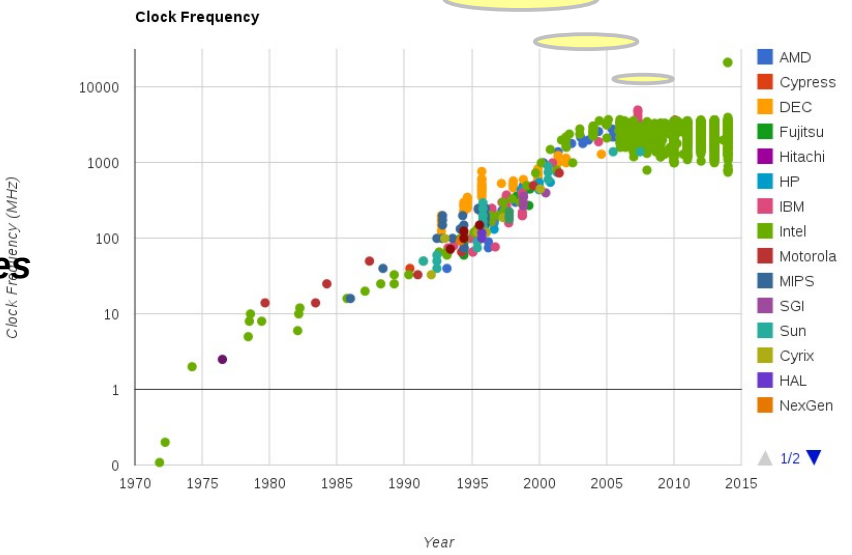
    => computing gets power limited

    => the dawn of the *multicore era*

    multiple processors ("cores") on a chip, not faster cores

    => *"free lunch" is over* *(H. Sutter 2005: The free lunch is over. A fundamental turn toward concurrency in software)*

note the dominance of Intel chips since ~2005



http://cpudb.stanford.edu

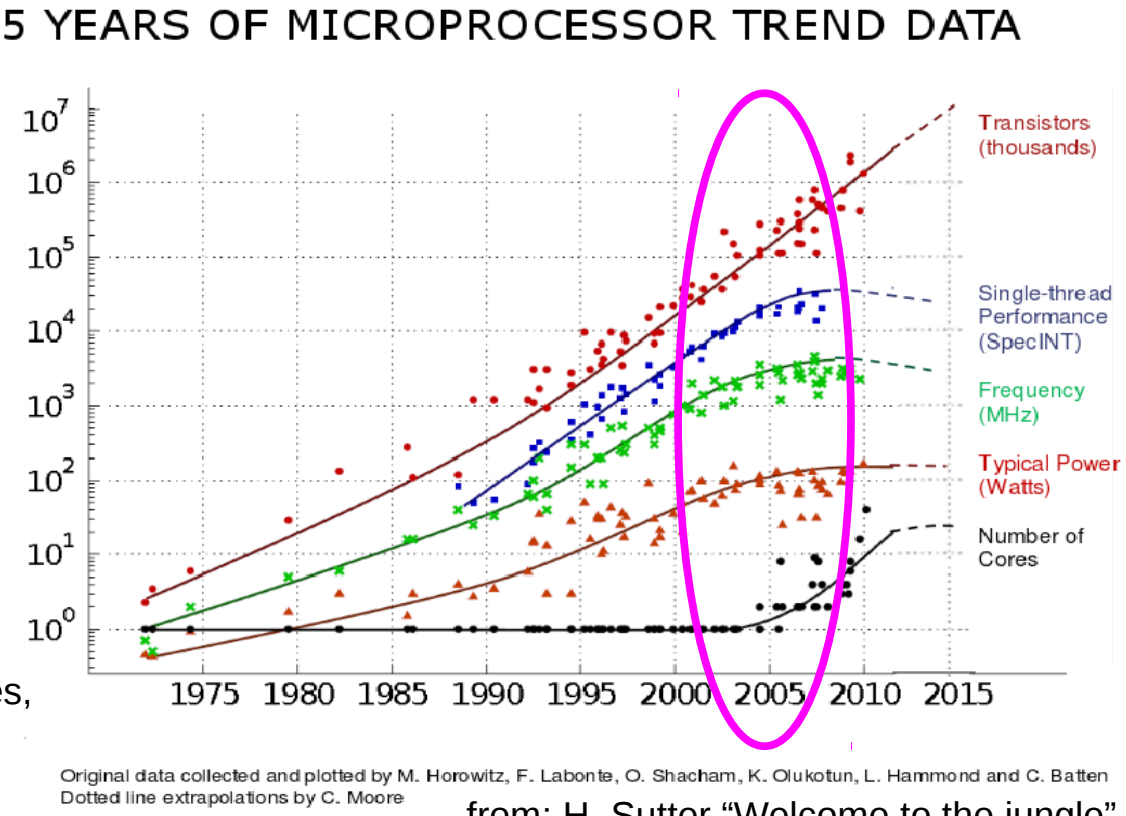
Processor-technology evolution

Trends

- Multicore ('big', out-of-order cores, latency optimized)
- Manycore, GPU ('little'/in-order cores, throughput optimized)
- heterogeneous nodes (different flavours of cores)

Drivers:

- technology: energy-efficiency, power density, ...
- economics: commoditization (mobile devices, gaming, cloud, …)



from: H. Sutter “Welcome to the jungle”

single-threaded “free lunch” (-2005)

multi core (2005-)

het. core (2009-)

=> Moore's law currently holds in its original formulation, but not in the pre-2005 interpretation (“CPU speed doubles every 18 months”)

Predictions:

- towards the end of Moore's law (*era of diminishing returns*):
  14% / year → 8nm (cf. *Dark silicon and the end of multicore scaling*, Esmaeilzahde et al. ISCA 2011)

HPC trends

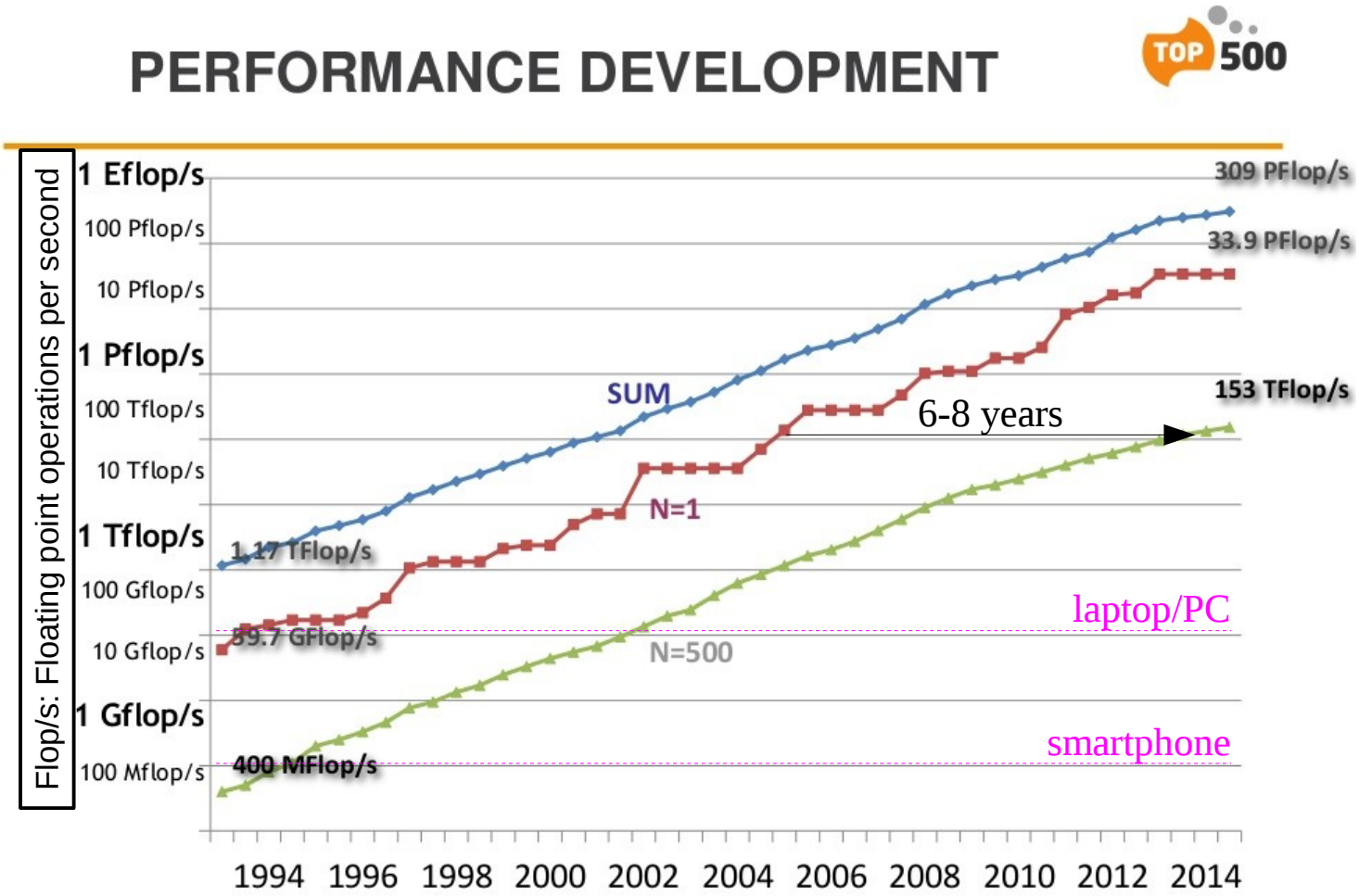
The Top 500 list of supercomputers (based on HPL benchmark)

notes:

HPL traces evolution of *peak floating point performance* (resp. technology and economics), rather than *"sustained" application performance* (which really matters)

HPL benchmarks *dense linear algebra* (BLAS3)

Top500 race resulted in highly platform-optimized linear algebra libraries (BLAS, LAPACK, ScaLAPACK)



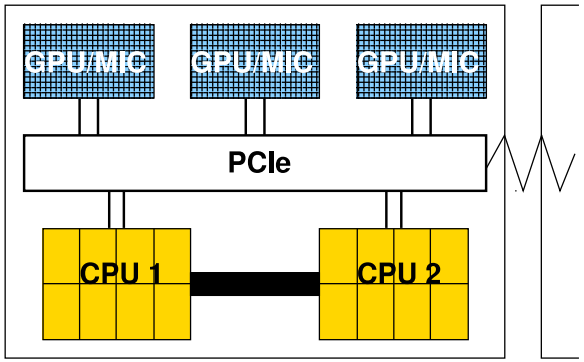
from: Highlights of the 44th TOP500 List (E. Strohmaier, SC'14, New Orleans)

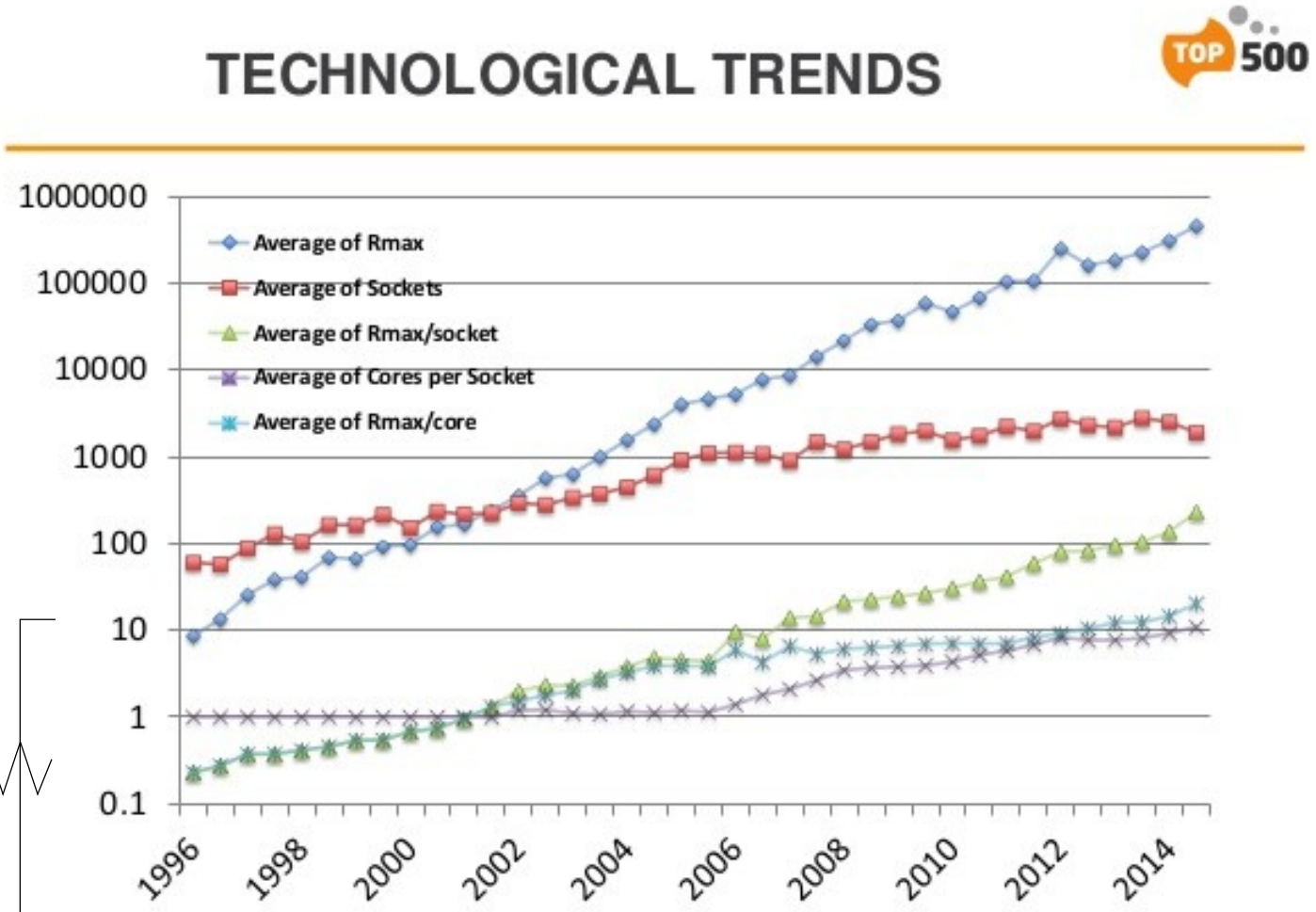
HPC trends

The Top 500 list of supercomputers (based on HPL benchmark)

notes:

- moderate performance increase *per core*
- steady performance increase per node (socket)
- typical architecture:





Highlights of the 44th TOP500 List (E. Strohmaier, SC'14, New Orleans)

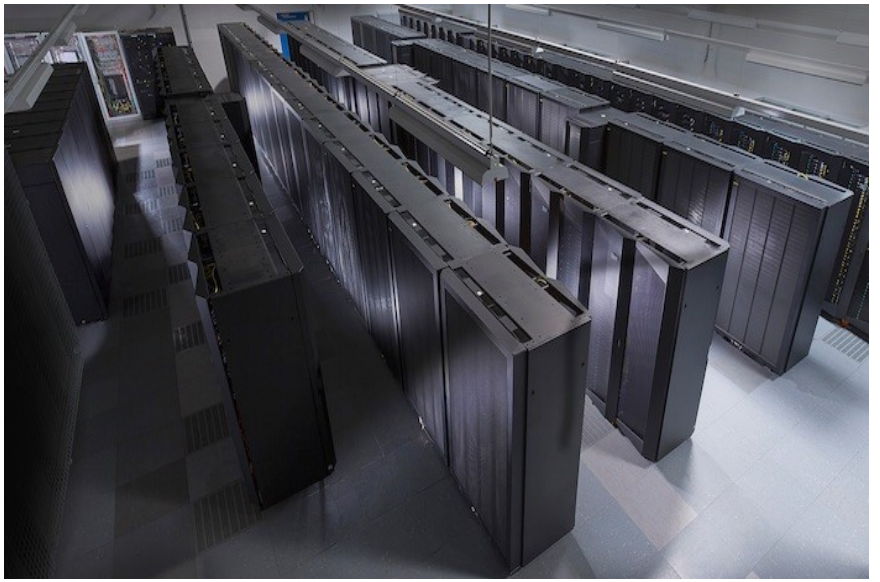
HPC trends

… where we are:

Hardware

MPG Hydra, a typical Petaflops-class x86_64 HPC cluster in Europe (cf. SuperMUC, CURIE, Hermite, ...):

- 4200 compute nodes (~ 80000 cores, 260 TB RAM)
- 700 GPU accelerators (2 PCI cards per node)
- InfiniBand network (FDR 14 x4: 5.8 GB/s, 5 µs)
- Power: ~ 1MW



Applications:

- FORTRAN, C, C++, few GPU/MIC
- plain MPI, some hybrid MPI/OpenMP
- typical job size: 1000...10000 cores
- typical time of individual jobs: n x 24h
- top-dogs:

  FHI-aims*, GENE*, NAMD, FLASH, GROMACS, NECI*, PENCIL, LAMMPS, PLUTO, VERTEX*, VASP, OCTOPUS*, MAGIC*, … (*: actively developed in the MPG)

(reflects the operational policies of the machine, not the scalability of the codes!)

HPC trends

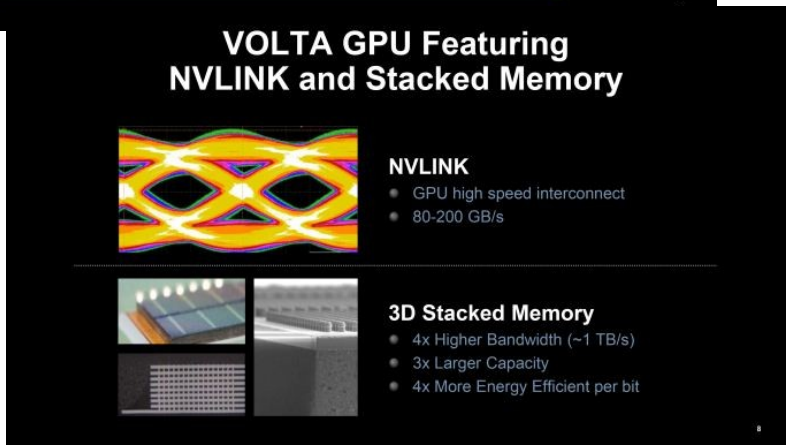
… where we are heading for:

Systems	2011	2012	2015	2018-2022	change 2022/"today"
System Peak [PF]	2	25	200	1000	O(1000)
Power [MW]	6	6-20	15-50	20-80	O(10)
System Memory [PB]	0.3	0.3-0.5	5	32-64	O(100)
GB RAM/Core	0.5-4	0.5-2	0.2-1	0.1-0.5	**O(0.1)**
Node Perf. [GFlop/s]	125	160-1000	500-7000	1000-10000	O(10)-O(1000)
Cores/Node	12	16-32	100-1000	1000-10000	**O(100)-O(1000)**
Node memory BW [GB/s]	40	70	100-1000	400-4000	O(100)
Number of nodes	20000	10.000-100.000	5000-50.000	100.000-1.000.000	O(10)-O(100)
Total concurrency	~200000	$O(10^6)$	$O(10^7)$	$O(10^9)$	O(10.000)
MTTI	days	days	O(1 day)	O(<1 day)	O(0.1)

source: DARPA exascale computing study (reproduced from EESI report, slightly updated)

US project CORAL

- upcoming “pre-exascale” systems in 2017/2018 (~ 300 PFlop/s, 10MW)
- 2 systems (SUMMIT, SIERRA) based on 2x IBM Power9 CPU (10%)+ 6x Nvidia Volta GPU (90%)
- only 3400 nodes!, 40TFlop/s/node



- *“exascale is done from the hardware perspective”* (J. Sexton, IBM, at SC'14, New Orleans)

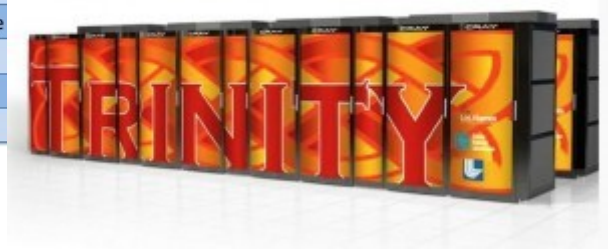
HPC trends

- CRAY Aurora (Intel's Knights Hill)

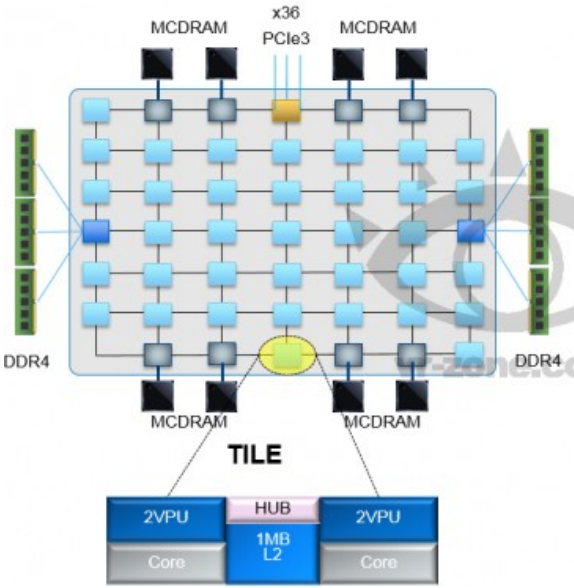
First massive manycore system: Trinity @NERSC (2015)

Trinity Architecture Overview

Metric	Trinity		
Node Architecture	KNL + Haswell	Haswell Partition	KNL Partition
Memory Capacity	2.11 PB	>1 PB	>1 PB
Memory BW	>7PB/sec	>1 PB/s	>1PB/s +>4PB/s
Peak FLOPS	42.2 PF	11.5 PF	30.7 PF
Number of Nodes	19,000+	>9,500	>9500
Number of Cores	>760,000		
Number of Cabs (incl I/O & BB)	112		
PFS Capacity (usable)	82 PB usable		
PFS Bandwidth (sustained)	1.45 TB/s		
BB Capacity (usable)	3.7 PB		
BB Bandwidth (sustained)	3.3 TB/s		

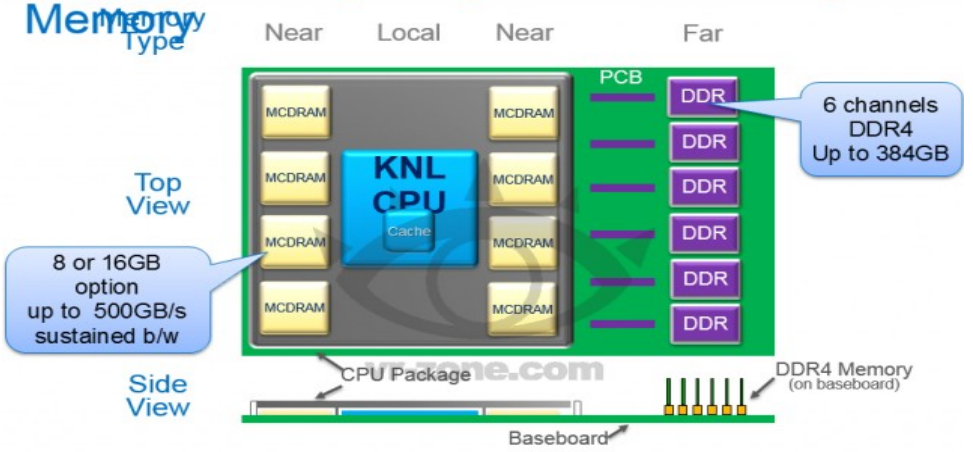


Knights Landing Processor Architecture



- Up to 72 Intel Architecture cores based on Silvermont (Intel® Atom processor)
  - Four threads/core
  - Two 512b vector units/core
  - Up to 3x single thread performance improvement over KNC generation
- Full Intel® Xeon processor ISA compatibility through AVX-512 (except TSX)
- 6 channels of DDR4 2400 MHz -up to 384GB
- 36 lanes PCI Express* Gen 3
- 8/16GB of high-bandwidth on-package MCDRAM memory >500GB/sec
- 200W TDP

Knights Landing Integrated On-Package Memory



Integrated on-package MCDRAM brings memory nearer to CPU for higher memory bandwidth and better performance

agram is for conceptual purposes only and only illustrates one [illegible] t to scale and does not include all functional areas of the CPU, nor does it represent

Motivations

… so what?

… stunt machines
… relevant for my research ?
… "1000 cores is enough for everybody" ?

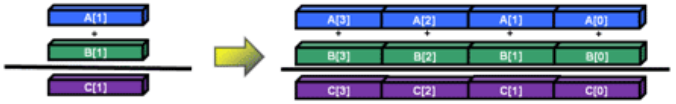
→ today's supercomputers are tomorrows commodity clusters

→ there seems no convenient way forward (beyond keeping the status quo in application performance)

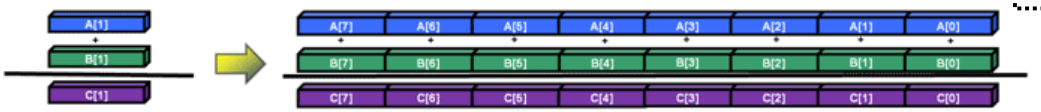
→ competition aspects: the resources are simply there!

→ "trickle down" of technology (much quoted) and knowledge transfer, e.g. SIMD vectorization:

SSE (4-way SIMD)



AVX (8-way SIMD)



AVX 256 (16-way SIMD)

Intel Xeon NH (*"commodity"* CPU)

Intel Xeon Phi (*"petascale"* tech.)

Intel Xeon SB (*"commodity"* CPU)

Intel Xeon Phi ("exascale" tech.)

Intel Xeon SKL (*"commodity"* CPU)

Programming challenges

Challenges for applications: *"... the Good, the Bad, and the Ugly ..."*

1) extreme concurrency and the "memory wall"

   → can probably be handled by hierarchica

      MPI + X , X Є {OpenMP, OpenACC, CUI

   → algorithms need to scale (even more)

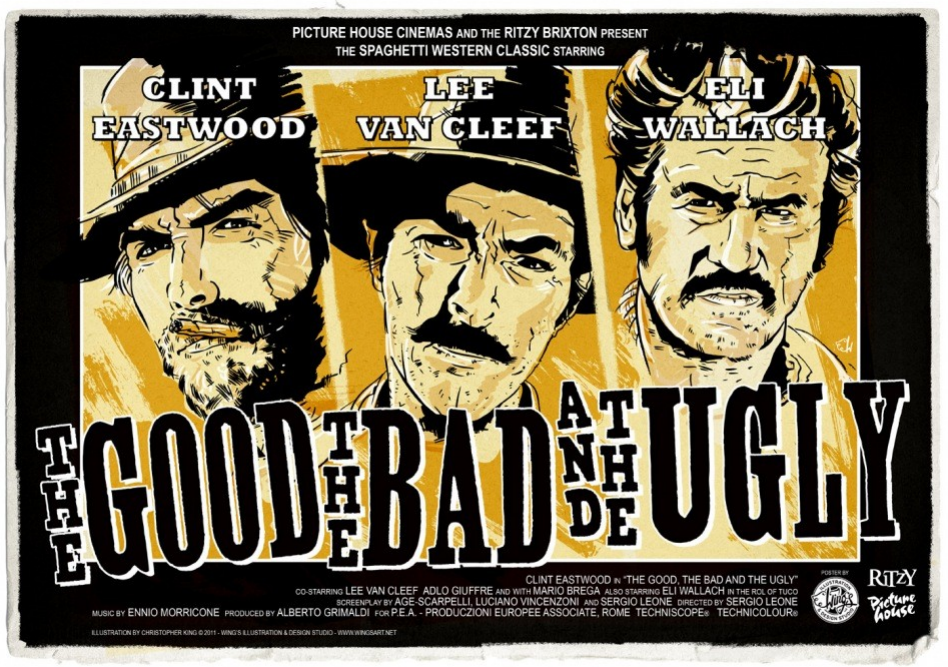
   → stagnating core-performance *forces* ap

2) heterogeneous nodes

   → likely: a mix of 'big' (Xeon like, OoO) a

   → memory hierarchy: towards a single ac

   but: *data locality is key ! → will "high-level" approaches like OpenACC save us the day???*



3) hardware reliability

   → hard and soft errors will be common for every job

   → towards "fault-tolerant" applications

Programming challenges (1)

Challenges for applications: *"... the Good ..." (?)*

1) extreme concurrency and the "memory wall"

   → can probably be handled by hierarchical parallelism

   MPI + X , X ∈ {OpenMP, OpenACC, CUDA, …} ?

   → algorithms need to scale (even more)

   → stagnating core-performance *forces* applications into a strong scaling scenario

2) heterogeneous nodes

   → likely: a mix of 'big' (Xeon like, OoO) and 'little' (MIC/GPU-like, InO) cores

   → memory hierarchy: towards a single address space

   but: *data locality is key ! → will "high-level" approaches like OpenACC save us the day???*

3) hardware reliability

   → hard and soft errors will be common for every job

   → towards "fault-tolerant" applications

The curse of Amdahl's law

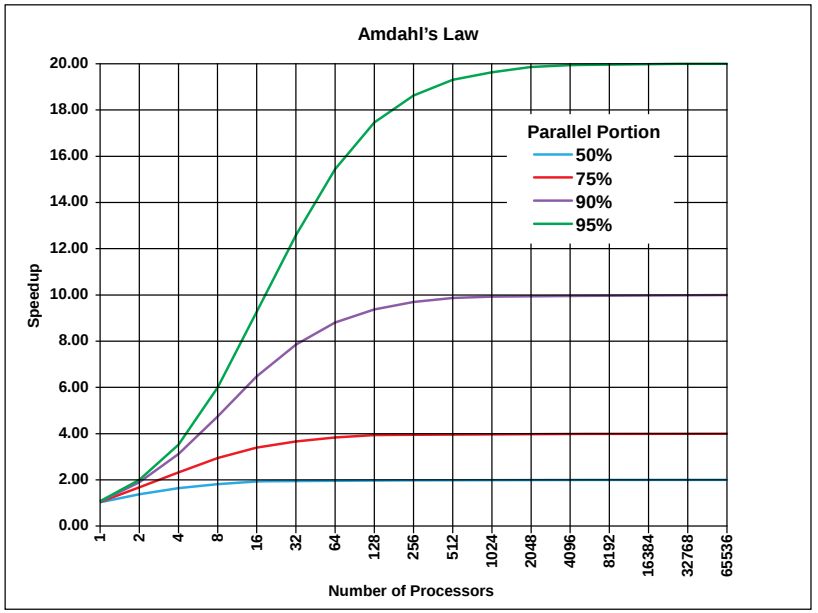
- recall: parallel **speedup on n cores = 1/(s+(1-s)/n)**

  where s = runtime share of serial code parts

  => **speedup < min(1/s,n)**

  - 1000 cores (threads): requires s < 0.1%
  - 10000 cores (threads): requires s < 0.01%
  - 100000 cores (threads): requires s < 0.001%

  → **are we completely lost? …**



→ strong scaling (fixed problem size): time stepping/iterative algorithms

- serial runtime per time step: O(1 s) per time step
- O(10^7) timesteps => serial runtime ~ 3000 h

  desired speedup ~ 1000 => "parallelize away" everything which is > 1 ms

  → latency, overhead (OpenMP, MPI,...) ~ 10 µs => still O.K.

→ weak scaling (problem size increases with compute resources): sounds like a "weak" justification but has great relevance … are our simulations numerically converged already ?

→ ensemble computing: a typical justification of HPC vendors/providers … how scientifically useful are single "hero" runs compared to parameter studies at 10% of the size of the machine ?

The “memory wall”

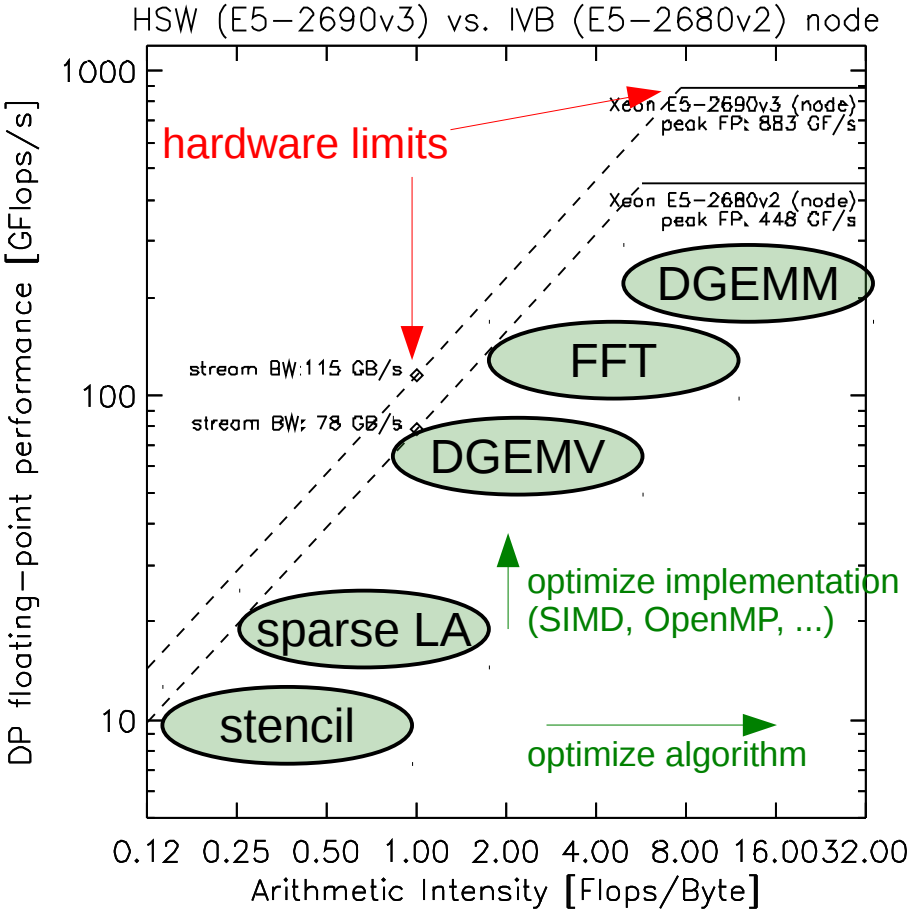
The memory wall:

- memory performance evolution [GB/s] lags behind floating-point performance [GFlop/s]:
  from >8 Byte/Flop (vector machines) to 1/8 Byte/Flop (current hardware)

- use algorithmic intensity, AI [Flops/B], to classify algorithm (kernels) and relate to hardware limits

$$R < \min(R_{FP}, AI * R_{Mem})$$

(S. Williams et al. Roofline: an insightful visual performance model for multicore architectures, Comm. ACM, 2009)



examples (“double precision” = 8 Bytes/word):

- x(i)=a(i)+c*b(i) → AI=2/3
- r(i)=x(i)**2+y(i)**2 → AI=3/2
- DGEMV → AI ~ 2
- DGEMM → AI ~ n
- FFT → AI ~ log(n)

practical and caveats:

- AI is not additive
- AI depends on implementation (e.g. cache)

→ measurement: AI=(Flops/s)/(Bytes/s)

- Flop/s counters broken in Intel server CPUs:
  SandyBridge, IvyBridge, Broadwell, …?

(e.g. J. Treibig's blog at http://likwid-tools.blogspot.de/2012/02/intel-sandybridge-and-counting-flops.html)

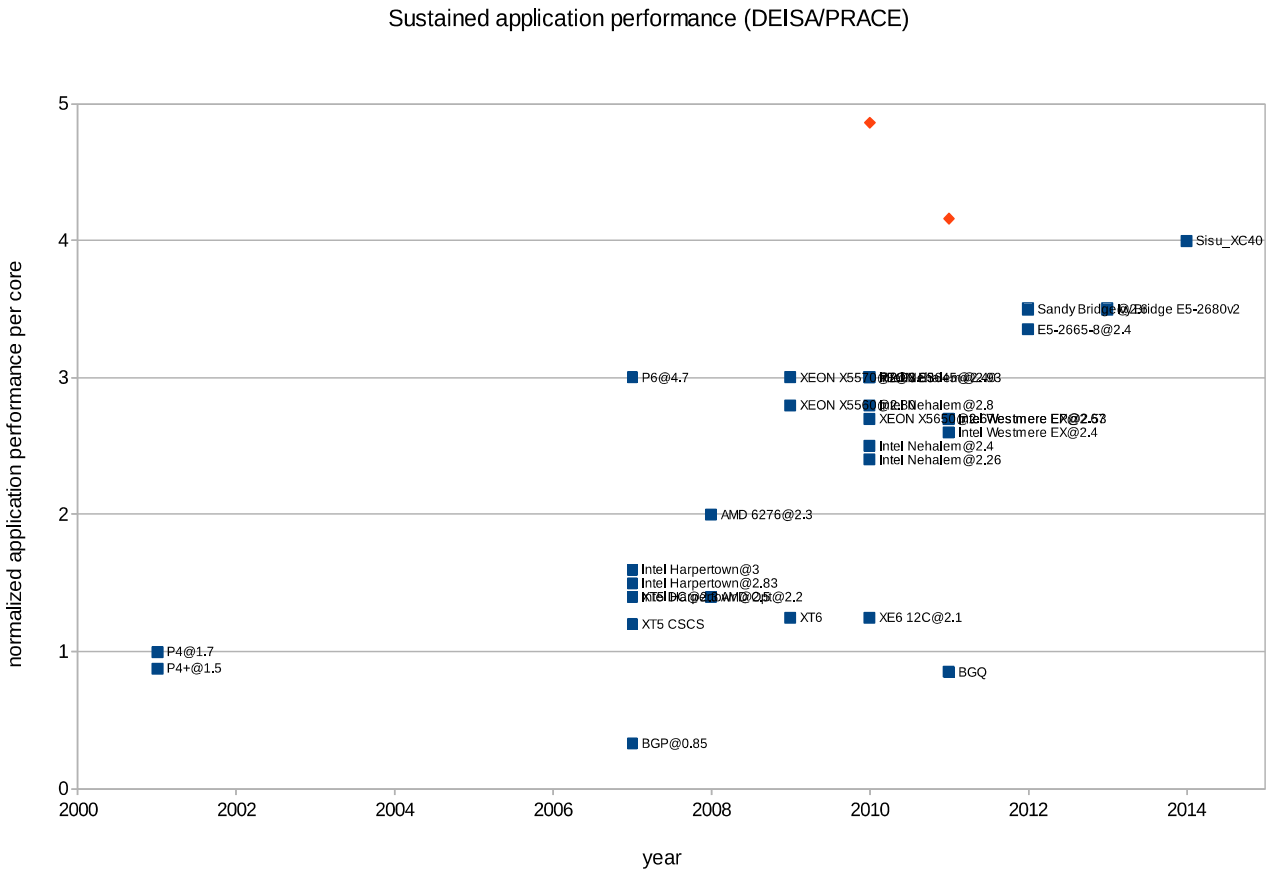
Implications

Implications of modern system architectures:

- a widening gap between nominal (“peak”) and sustained (“real”) application performance
  → from >50% peak (vector computers) to <10% peak (current hardware)
- main reasons: memory (→ per-core performance), parallelism (→ overall performance), I/O (never quoted in benchmarks)

Per-core-performance evolution
“in the field” (2001-2014)

- *application performance per core*: *4x (DEISA/PRACE)*
- *peak performance per core*: 20x (top500.org)
- *cf. peak performance per system*: ~2000x (Moore's law, Top500)



Programming

Programming models: manage parallelism on **multiple levels** *("how to tame the beast")*

Current:

- FORTRAN, C, C++ (+compiler hints for SIMD vectorization: **core level**)
- OpenMP, pthreads (thread parallelism: **node level**)
- MPI (inter -and intra- node parallelism: **cluster level**)
- CUDA, OpenACC (**accelerators**)
- issues: understanding and expressing memory locality, thread affinity, node-level performance

Towards exascale:

- best guess: "MPI + X"
- X= OpenMP and/or OpenACC + FORTRAN or C/C++
- alternatives: co-array FORTRAN (standard, communication built into the language, coexistence with MPI) ?

=> the basic tools are already there

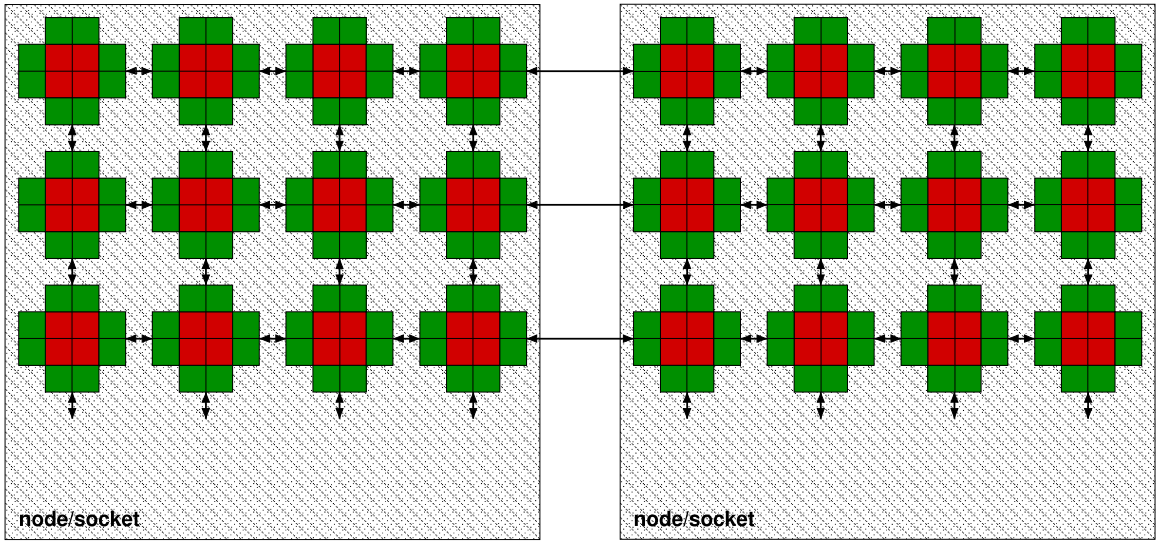
cause or consequence ?

Hybrid parallelization

Example:

- standard domain decomposition with halo-exchange
- explicit message passing (MPI)

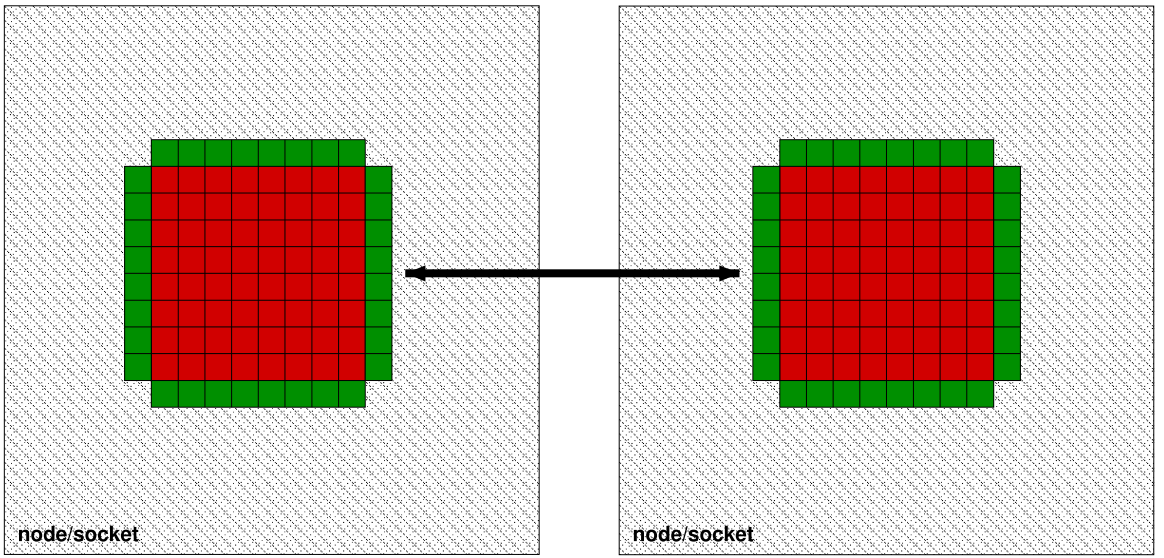
Plain MPI



Advantages:

- fewer MPI-ranks: * 1/cores per node (can be very beneficial for all-to-all type messaging)
- fewer but larger messages → reduce latency
- smaller surface/volume ~ communication/computation *(for contemplation: surface/volume in higher dimensions, e.g. 6D phase space ?)*
- smaller buffer size
- sharing of common data (e.g. EOS table, …)
- mitigate load-imbalances within domain

Hybrid: e.g. MPI + OpenMP or MPI + MPI (shared memory)



Hybrid parallelization: GOEMHD3

Example (GOEMHD3, w/ J. Skala, J. Buechner): 2D domain decomposition of a 3D Cartesian grid (explicit MHD code)

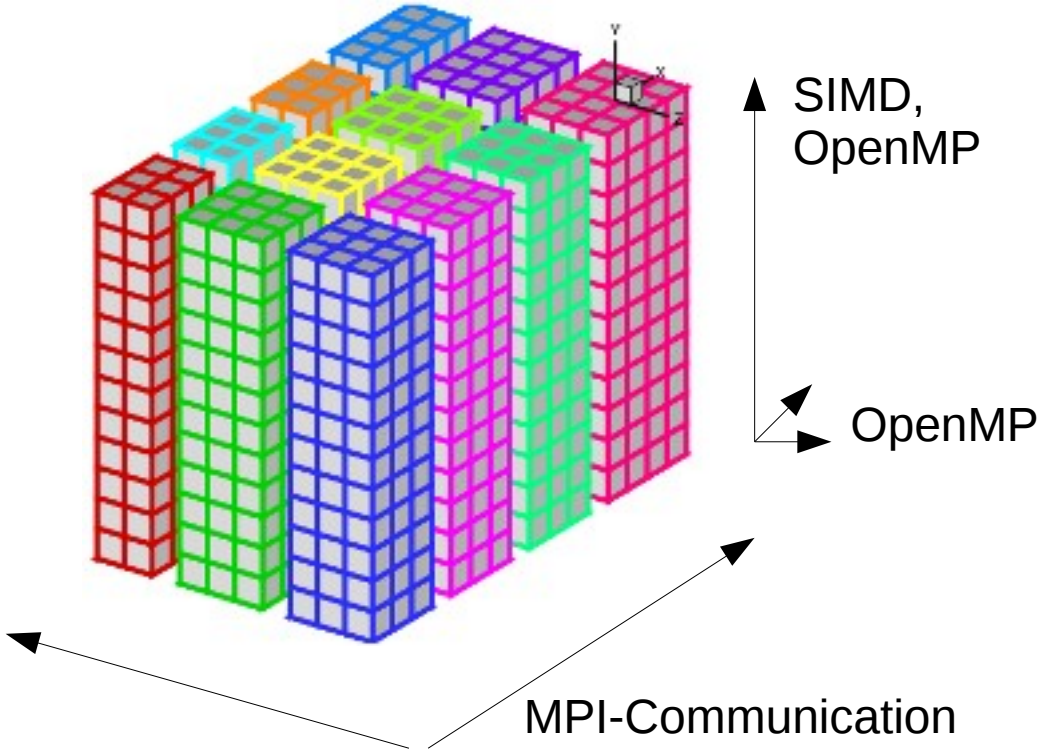
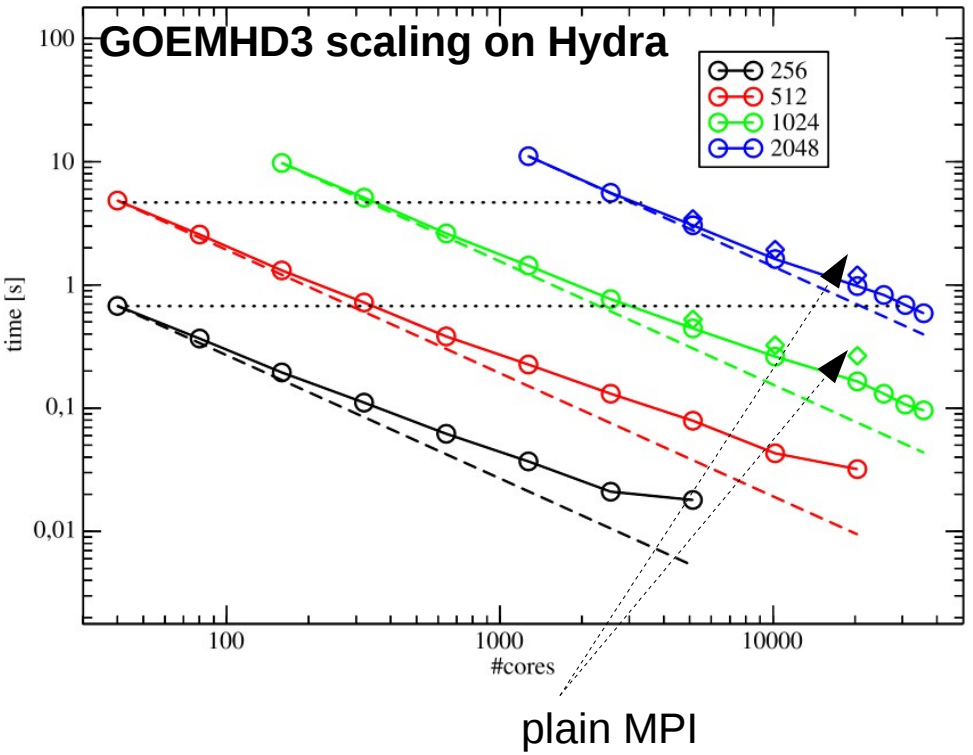


Illustration: www.2decomp.org

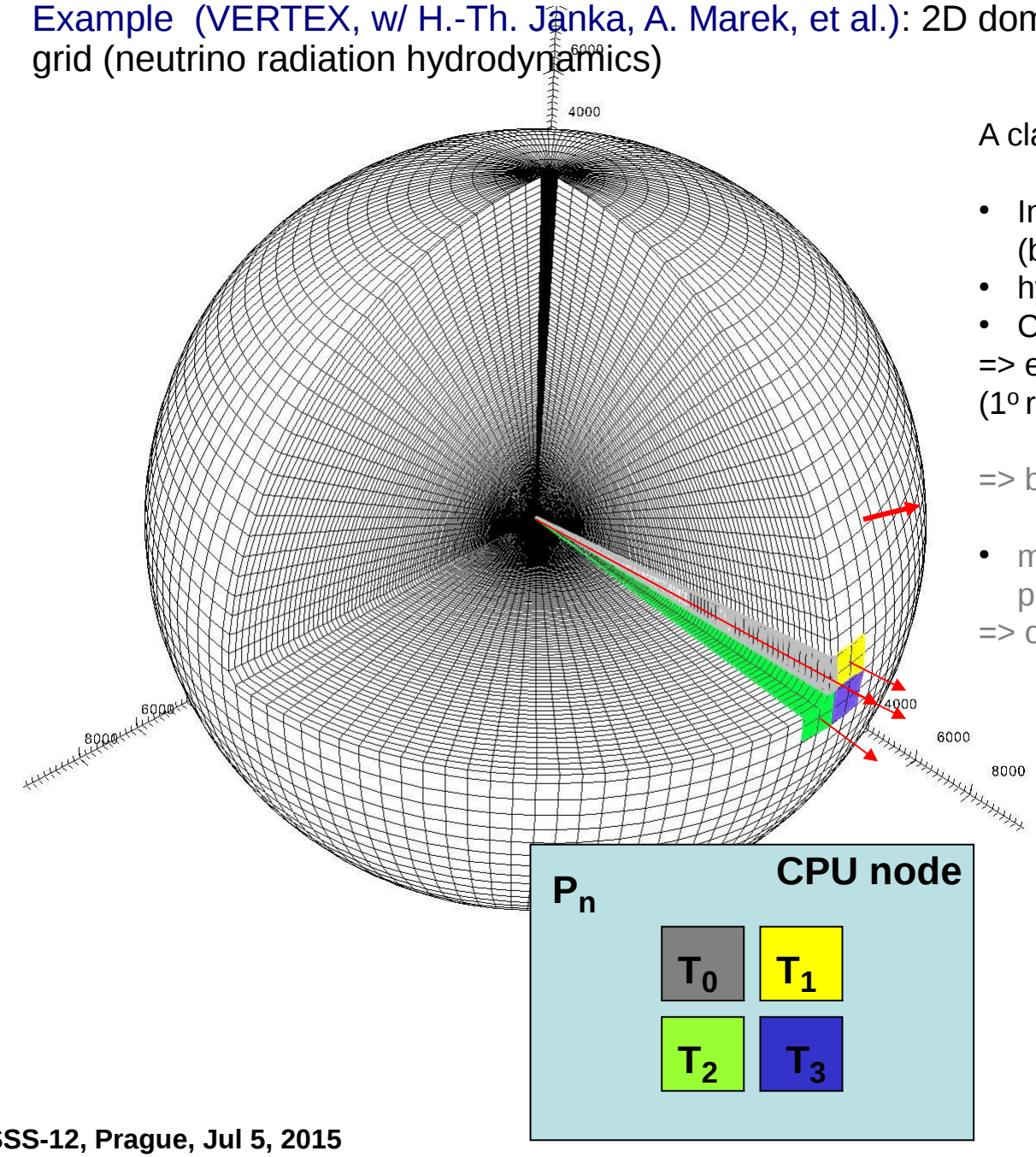


(Skala et al.: *The 3D MHD code GOEMHD3 astrophysical plasmas with large-Reynolds-numbers* to appear in A&A 2015)

- time-explicit MHD code (simple stencil)
- avoids 3D decomposition (or: enables a “simple” 2D decomp. to scale up by 10x...20x)
- surface-to-volume ratio (=communication/computation) decreases
- message size increases
- requires high threading parallel efficiency (close to ideal: 10 per CPU)
- **2000 MPI tasks vs 20000 MPI tasks**

Hybrid parallelization: VERTEX

Example (VERTEX, w/ H.-Th. Janka, A. Marek, et al.): 2D domain decomposition of a 3D Cartesian grid (neutrino radiation hydrodynamics)



A classical 2D domain decomposition in real space

- Individual (θ,Φ)-"rays" are weakly coupled (by approximation)
- hybrid MPI/OpenMP parallelization
- CPU-node: max 1 core per angular zone

=> excellent parallel scaling
(1^o resolution → 180x360 zones → max 64k cores)

=> but: OpenMP parallelism already exploited on the coarse-grained level

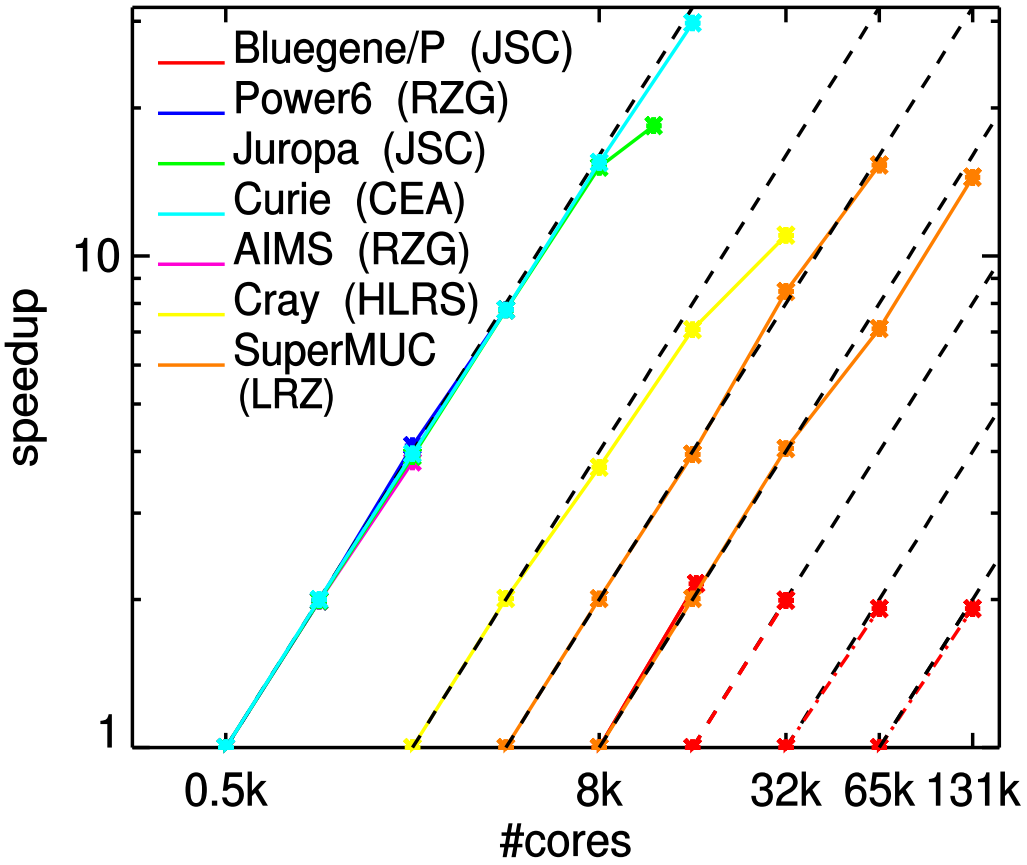
- momentum-space: lots of fine-grained parallelism: requires nested OpenMP

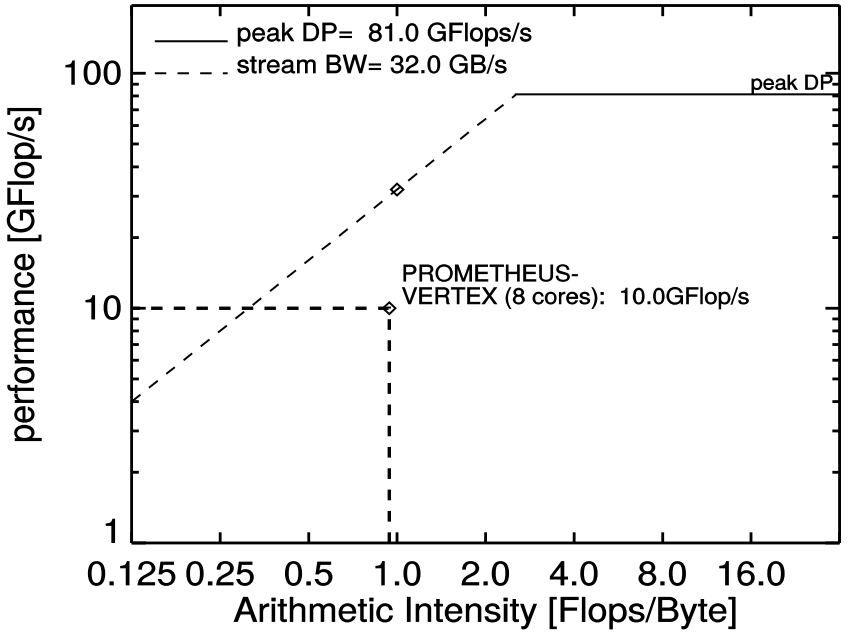
=> offload to GPU

phase-space coordinates

Hybrid parallelization: VERTEX

VERTEX on HPC clusters:





evolution of "legacy" codes is possible
VERTEX started as a serial vector code (NEC SX-4) in 1997
access to "stunt machines" has been invaluable for preparing production projects (at 1/10 of #cores)

0.25 Pflop/s on 131.000 cores (SuperMUC@LRZ)
=> VERTEX runs at ~ 10% of the nominal peak performance
(Marek et al.: *Towards Petaflops Capability of the VERTEX Supernova Code*, Adv. Par. Computing, 2014)

successful GPU-port (local physics): ~ 2x speedup (socket-to-socket comparison)
(Dannert et al.: *Porting Large HPC Applications to GPU Clusters: The Codes GENE and VERTEX*, Adv. Par. Computing, 2014)

Hybrid parallelization: VERTEX

Notes based on personal history with VERTEX:

- 1997 (started development): 1 GFlop/s (NEC SX, serial)
- 2000: 16 cores standard (OpenMP), end of the vector era
- 2000: first MPI parallelization (with help of RZG)
- 2003: few 100 cores standard production runs (MPI+OpenMP) → dreaming of TFlop/s
- 2006: few 1000 cores standard production runs → dreaming of PFlop/s
- 2013: successfully ported to GPUs → 2x speedup (failure on Xeon Phi)
- 2014: PFlop/s capability demonstrated, running production on 10000's of cores
- 2015: improved strong scaling w/ nested OpenMP
- … still at least on par with scientific competitors

should I really look into this OpenMP thing ?
→ … for the time being let's not tell the supervisor

this MPI stuff is horribly complex, should a phycisist really be bothered with an explicit communication library ("high-level" language?)
→ computing center forced and supported us ...

… GPU efforts driven by computing center

how relevant are such benchmark runs? production runs on 16k cores. 100k cores are not available/stable anyway, …
→ essential for preparing the code for next-gen. production runs (remaining serial parts, bottlenecks, I/O)

=> Exascale is not a "sonic barrier", be brave and just do it (others have done it before and will do it)

=> gradual code evolution is possible

Programming challenges (2)

Challenges for applications: *"... the Bad ..." (?)*

1) extreme concurrency and the "memory wall"

   → can probably be handled by hierarchical parallelism

      MPI + X , X Є {OpenMP, OpenACC, CUDA, …} ?

   → algorithms need to scale (even more)

   → stagnating core-performance forces applications into a strong scaling scenario

2) heterogeneous nodes

   → likely: a mix of 'big' (Xeon like) and 'little' (MIC/GPU-like) cores

   → memory hierarchy: towards a single address space

   but: *data locality is key ! → will "high-level" approaches like OpenACC save us the day???*

3) hardware reliability

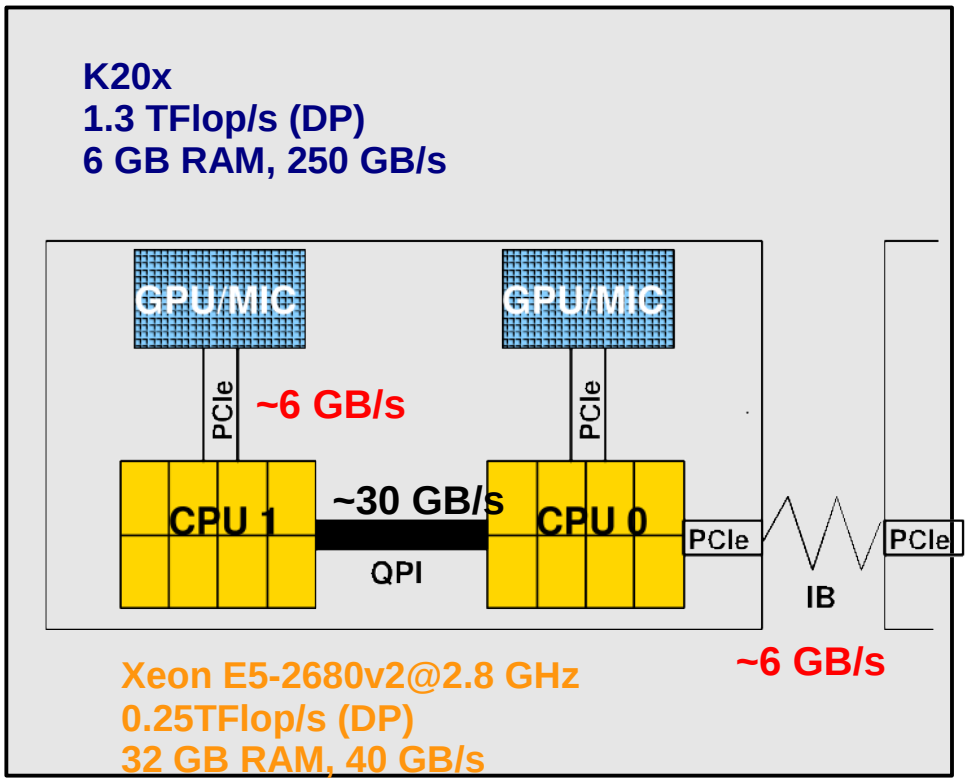
   → hard and soft errors will be common for every job

   → towards "fault-tolerant" applications

GPU and many-core computing

A view from the top (today's GPUs, many-core MIC processors):

- accelerator “cards” for standard cluster nodes (connected via PCIe)
- many (~50...500) “lightweight” cores (~ 1 GHz)
- high thread concurrency, fast (local) memories



System architecture:

- currently: x86 “Linux-clusters” with nodes comprising
  - 2 CPUs (8...16 cores)
  - 2 accelerator cards (GPU, MIC)
- future:
  - GPU/CPU with unified memory (OpenPower, Nvlink)
  - “host-less” MICs (Intel Knights Landing, Knights Hill, ...)

Programming paradigms (today):

- use CPU for program control, communication and maximum *single-thread performance*
- “offload” data-parallel parts of computation to accelerator for maximum *throughput performance*
- requires heterogeneous programming & load-balancing, careful assessment of “speedups”

GPU and many-core computing

Programming GPUs:

- CUDA → low level, C only (note: PGI provides CUDA-FORTRAN)
- OpenACC, OpenMP → high-level, directive-based just maturing
- today: discrete memory → coarse granularity
- to come: unified memory Nvlink (OpenPower)

Programming Many core (Intel Xeon Phi):

- OpenMP, compiler directives

Experiences and observations

- programming for GPU and many-core is not easy
- heterogeneous programming (the *need* to utilize both the CPU and the GPU) is even harder
  → dynamically balancing CPU and GPU workload can be a huge challenge for complex codes
- future “host-less” systems are expected to alleviate the burden of heterogeneous programming
- low-level, proprietary programming models like CUDA appear problematic for scientific HPC (sustainability: 10k...100k lines of code, dozens of person years, FORTRAN, …) → high-level models: OpenACC, OpenMP
- exploiting SIMT and SIMD parallelism in our algorithms is crucial for reasons of energy-efficiency (GPU or many-core, but also for the CPU)
- realistic application speedups: 1.5x … 3x (node with accelerators vs. node without accelerator)

Challenges and opportunities

2x...3x speedups *(time to solution)* appear competitive for complex HPC codes, but:

- do not enable *qualitatively* new science objectives right now
- sustainability ? (10k ...100k LoC)
- regular CPUs (Xeon) still do a very good job, further performance increases (?)

Don't expect a convenient way forward (cf. pre-exascale systems ~2018)

- OpenPower (Power CPUs, Nvidia VOLTA, Nvlink), SUMMIT @ORNL https://www.olcf.ornl.gov/summit/
- ManyIntegratedCore (KNL, KNH) AURORA@ANL http://aurora.alcf.anl.gov/
- recall: with current (CPU) technology 1 PFlop/s * 1y ~ 1 MW y ~ 1M € => 1 EFlop/s with 1 GW ???

when to start and how?
OpenMP, SIMD → start now (immediate benefits on CPU)
GPUs → explore OpenACC (PGI, CRAY, gcc)
code development, refactoring → start now (strategy!)

Programming challenges (3)

Challenges for applications: *"... and the Ugly ..." (!!!)*

1) extreme concurrency and the "memory wall"

   → can probably be handled by hierarchical parallelism

      MPI + X , X ∈ {OpenMP, OpenACC, CUDA, …} ?

   → algorithms need to scale (even more)

   → stagnating core-performance forces applications into a strong scaling scenario

2) heterogeneous nodes

   → likely: a mix of 'big' (Xeon like) and 'little' (MIC/GPU-like) cores

   → memory hierarchy: towards a single address space

   but: *data locality is key ! → will "high-level" approaches like OpenACC save us the day???*

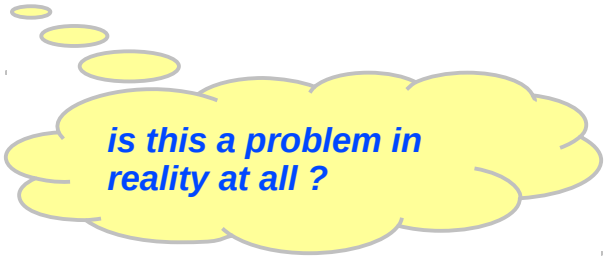
3) hardware reliability

   → hard and soft errors will be (are) common for every (large) batch job

   → towards "fault-tolerant" applications

Hardware failures

Estimates (based on present-day numbers):



- P_{node}: probability for a single node to fail during 24h
- N_{node}: number of nodes for a batch job
- $P_{job}(N,P)$: probability for a 24h job running on N nodes to crash

$P_{job}(N_{node}=1000, P_{node}=10^{-4}) = 1-(1-0.0001)^{1000} \sim 10\%$

$P_{job}(N_{node}=500, P_{node}=5*10^{-4}) = 1-(1-0.0005)^{500} \sim 22\%$

$P_{job}(N_{node}=1000, P_{node}=5*10^{-4}) = 1-(1-0.0005)^{1000} \sim 39\%$

Measures (basic):

(M. Snir et al. Addressing failures in exascale computing, 2012 Park City USA)

- regular application checkpoints to disk, restart on failure
- current: (10...30 GB/s) → $T_{read/write\ CP}$ ~ 2000s for a 64 TB dump on 1000 nodes
- upcoming: NVRAM (I/O “burst buffer”) technologies (Intel, CRAY, …)
- optimal checkpoint interval T to minimize “waste” $\approx T_{CP}/T+T/T_{MTBF}$ *(Young 1974, Dally 2004):*

$T=(2*(T_{MTBF}-(T_{down}+T_{read\ CP}))*T_{write\ CP})^{1/2}$ *(Aupy et al., J.Par. Distrib. Comp., 2013)*

Hardware failures

Measures (advanced, includes issue of soft errors):

- redundant calculations with checkpoints (local memory), replication or (approximate) reconstruction of data
- ABFT: algorithm-based fault tolerance

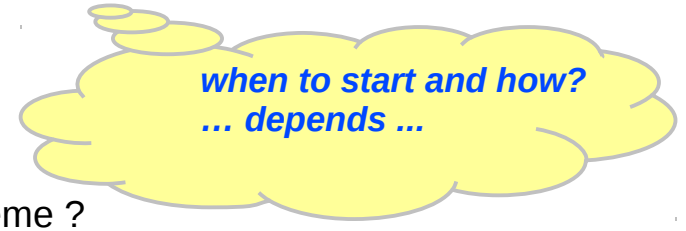
  resilient programs (adaptive scheduling, master-worker scenarios: job level, MPI level, iterative solvers, ensembles, sparse grids, …, …)

→ ongoing research in computer science, application development and HPC technology

- F. Capello et al. *Toward exascale resilience: 2014 update*, Supercomputing frontiers & innovations, 2014
- M. Snir et al. *Addressing failures in exascale computing*, 2013, http://www.mcs.anl.gov/papers/P5022-0913.pdf
- ...

Programmer's tools (no convenient – *transparent* – solutions on the horizon):

- Open-MPI prototype (http://fault-tolerance.org/) → standardization efforts: "MPI-next"
  - user directed, and communicator-driven fault tolerance → notification of the application (ULFM)
  - coordinated and uncoordinated process checkpoint and restart
  - data reliability and network fault tolerance
- GASPI, Co-Array Fortran, ...
- how to implement this, e.g. in a standard domain decomposition scheme ?



promising attempts, e.g.:
- D. Pflüger et al. Lecture Notes in Computer Science Volume 8806, 2014, pp 565-576
- A. Md Mohsin et al., HPCS 2015

Software engineering

Standard techniques (cf. softwarecarpentry.org)

- source code management/version control

  git, subversion, …

- static checks, use different compilers!, forc
  (cf. www.polyhedron.org)

- run-time checks (compiler optio
  marmot/must, debuggers, ...)

- unit tests: verification and v
  coverage, performance ver

- code refactoring (e.g.
  Fortran2003, MPI 1 →

  - take advantage of t
    language evolution,

  - hint: check implem
    completeness first!

Towards “continuous integration”

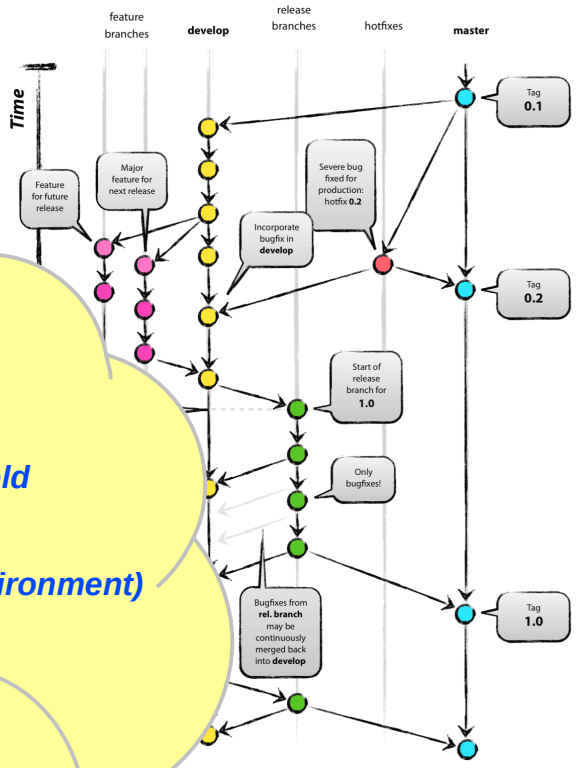
tools and frameworks: Jenkins, build-bot, [illegible]

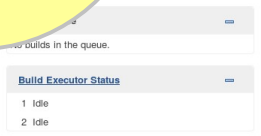
- commit hooks, automated builds, notifications, …
- test-driven development (turn bugs into test cases)
- automated unit tests for V&V and code coverage
- “release” management, issue tracker, ...

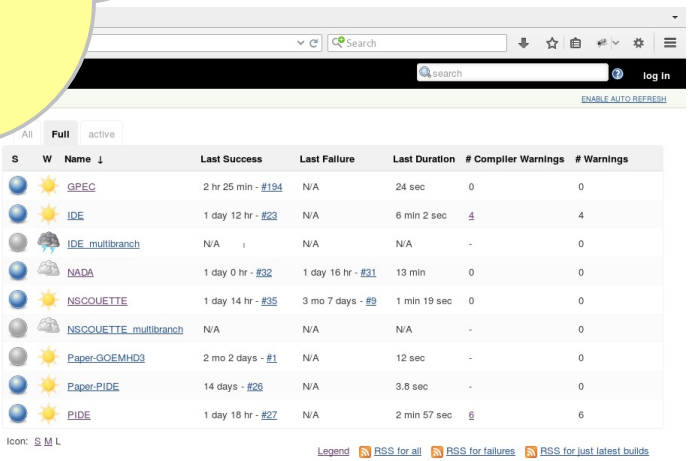
Why bother in this context?

→ (unfortunately) not a matter of course in the field

- ***ever increasing complexity of software/codes***
- ***team development***
- ***multiple platforms (hardware and software environment)***
- ***performance of computers, applications ???***
- ***large runs are expensive***
  OoM estimate for a 24h run @ 1 Petaflop/s:
  (nominal! → 50 Teraflop/s sustained):
  10 M€/5y (hardware) + 1 M€/1y (el. power) =
  = 3 M€/y ~ 10k€/24h
- ***the value of your time/career ?***







Software engineering

Standard techniques (cf. softwarecarpentry.org)

- source code management/version control

  git, subversion, …

- static checks, use different compilers!, forcheck, ... (cf. www.polyhedron.org)

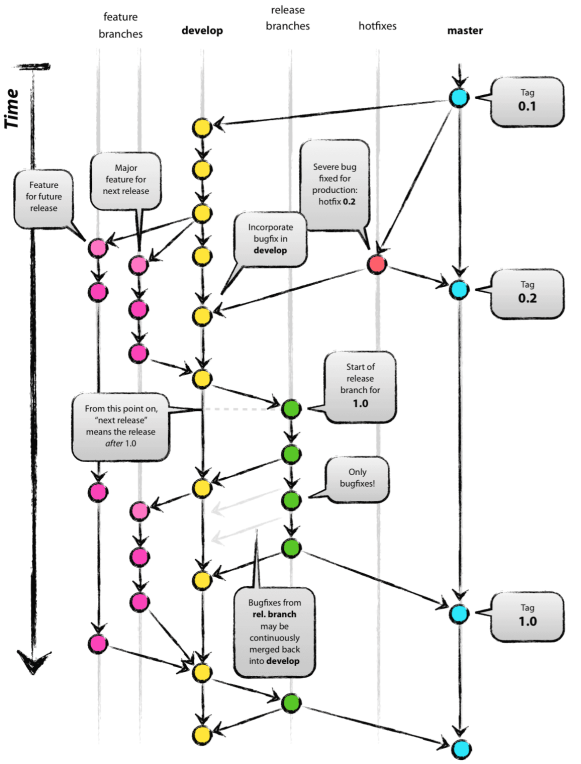
- run-time checks (compiler options, valgrind, marmot/must, debuggers, ...)

- unit tests: verification and validation (of results), code coverage, performance verification

- code refactoring (e.g. Fortran77 → Fortran90 → Fortran2003, MPI 1 → … → MPI 3.1)

  - take advantage of type checking, modularization, language evolution, ...

  - hint: check implementation status, feature completeness first!



Towards “continuous integration”

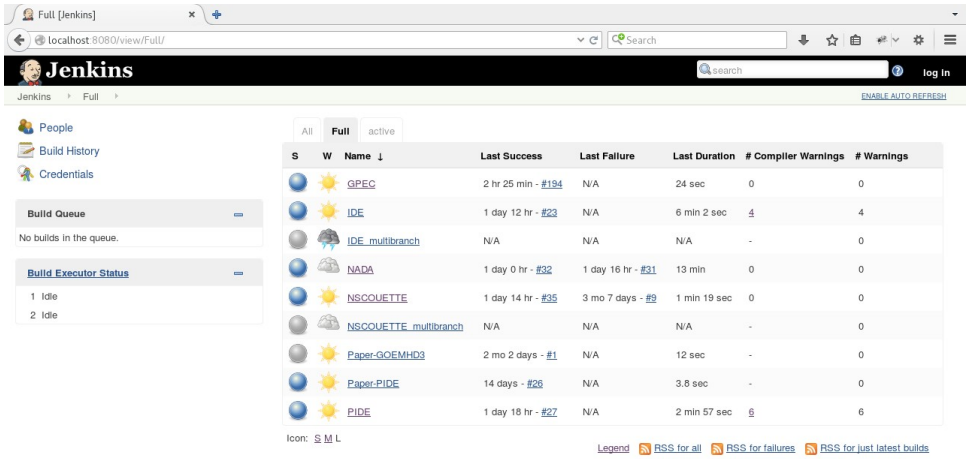
tools and frameworks: Jenkins, build-bot, git, gcov[r], …

- commit hooks, automated builds, notifications, ...

- test-driven development (turn bugs into test cases)

- automated unit tests for V&V and code coverage

- “release” management, issue tracker, ...



Summary & Conclusions

Summary and some predictions (* to be challenged at ISSS 17)

Exaflops systems, *O*(100M $), are on the horizon, technology will “trickle down”

Petaflops, *O*(10M $), is about to become mainstream

Teraflops, *O*(1000 $), is available on a single chip (massive parallelism)

Implications and challenges for scientific application development:

there is no convenient way forward ← ***computing is power limited***

the tools are already around (for the better or worse?)

- parallelization on multiple levels (e.g. SIMD with compiler or OpenMP pragmas)
- hybridization (e.g. with OpenMP within socket or node) ← ***flat MPI will not survive***
- “accelerators” (GPU, MIC) ← ***plain multicore CPUs will not prevail***
- hardware abstraction by OO techniques (C++, FORTRAN 200x, …) ← ***no single HW***

applications have to implement resilience to hardware failures ← ***large, monolithic jobs will crash***

(my own – somewhat pessimistic – view) ← ***HPC depends on mass-market evolution***

- highly application specific, just scratched the surface here

Summary & Conclusions

Summary and some predictions (* to be challenged at ISSS 17)

Exaflops systems, *O*(100M $), are on the horizon, technology will “trickle down”

Petaflops, *O*(10M $), is about to become mainstream

Teraflops, *O*(1000 $), is available on a single chip (massive parallelism)

Implications and challenges for

there is no convenient way forw

the tools are already arou

- parallelization
- hybridization (e.g.
- “accelerators” (C
- hardware abstracti

applications have to implement resili

(my own – somewhat pessimistic – view)

- highly application specific, just scratched the surface here

… don't get obsessed by Flop/s and scaling curves!

“the cheapest (floating-point) operations are those that you don't even execute” (R. Fischer, NEC)

time-to-solution vs. numerical and physical accuracy matters! → clever algorithms

vail

ithout jobs will crash

s on mass-market evolution