Séminaire Fime Polytechnique – IBM

7 décembre 2012

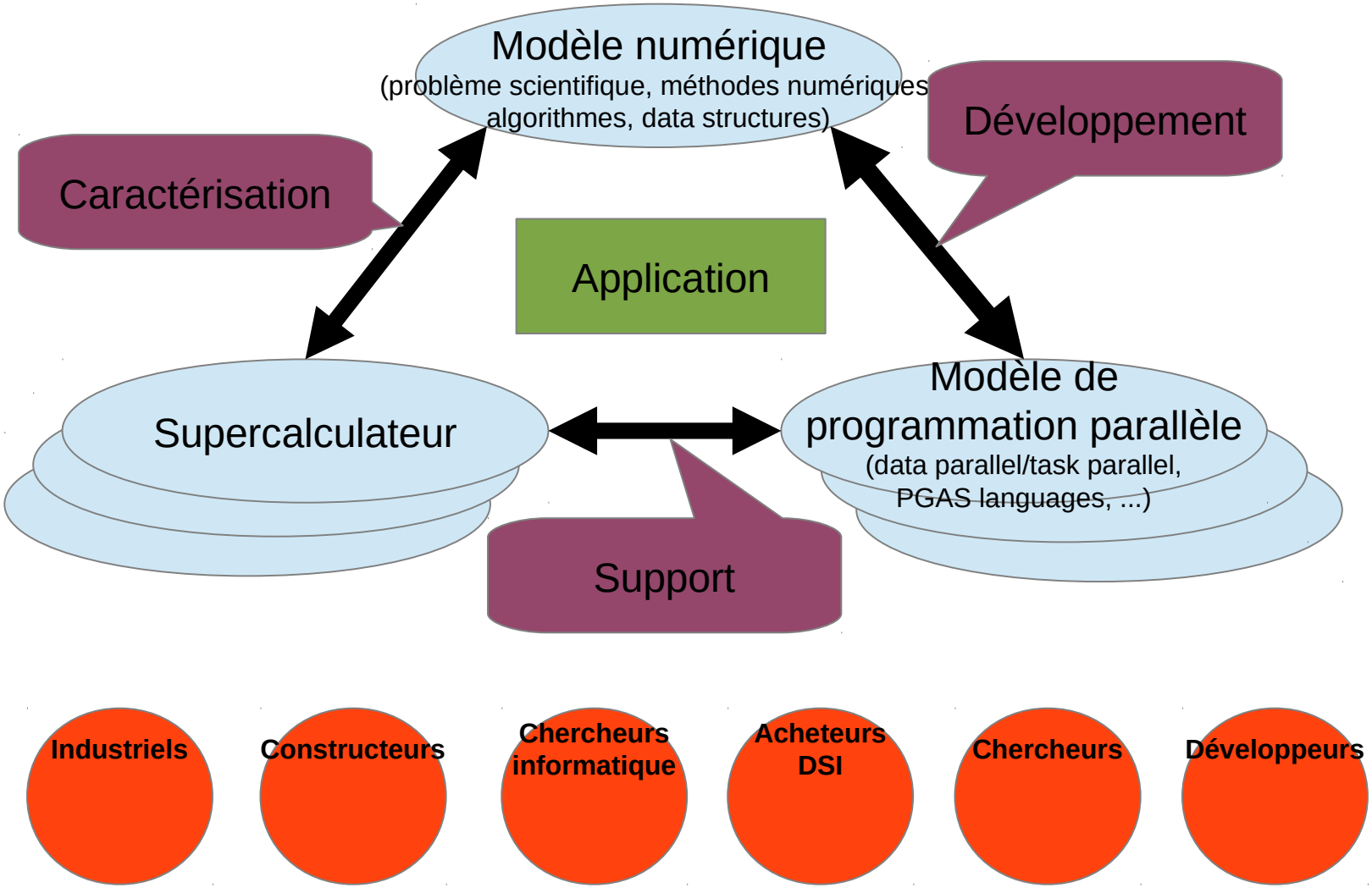
HPC & Simulation/modélisation de l'aléatoire

François Thomas, ft@fr.ibm.com
François Bothorel, Francois.Bothorel@fr.ibm.com
Sylvie Boin, Sylvie_boin@fr.ibm.com
Georges Uzbelger, Georges.Uzbelger@fr.ibm.com

Agenda

- IBM Technical Computing : la vision, les tendances
  - Passage du HPC to Data Centric Deep Computing
  - Les moteurs d'acroissement de puissance en HPC
  - Survol des architectures de microprocesseurs
  - Serveurs et calculs hybrides
- Quel modèle de programmation pour le massivement parallèle ?
  - Diversité des modèles de programmation
  - Critères de choix
- Caractérisation des applications
  - Quelle est l'empreinte de cette application sur ce matériel ?
  - Et quelle sera-t'elle sur le matériel de demain ?
  - Que peut on mesurer ?
  - Cette application est elle performante ?

Schéma



HPC trends and directions

HPC use cases

Discovery

- Huge simulations to discover new things, explain a phenomenon

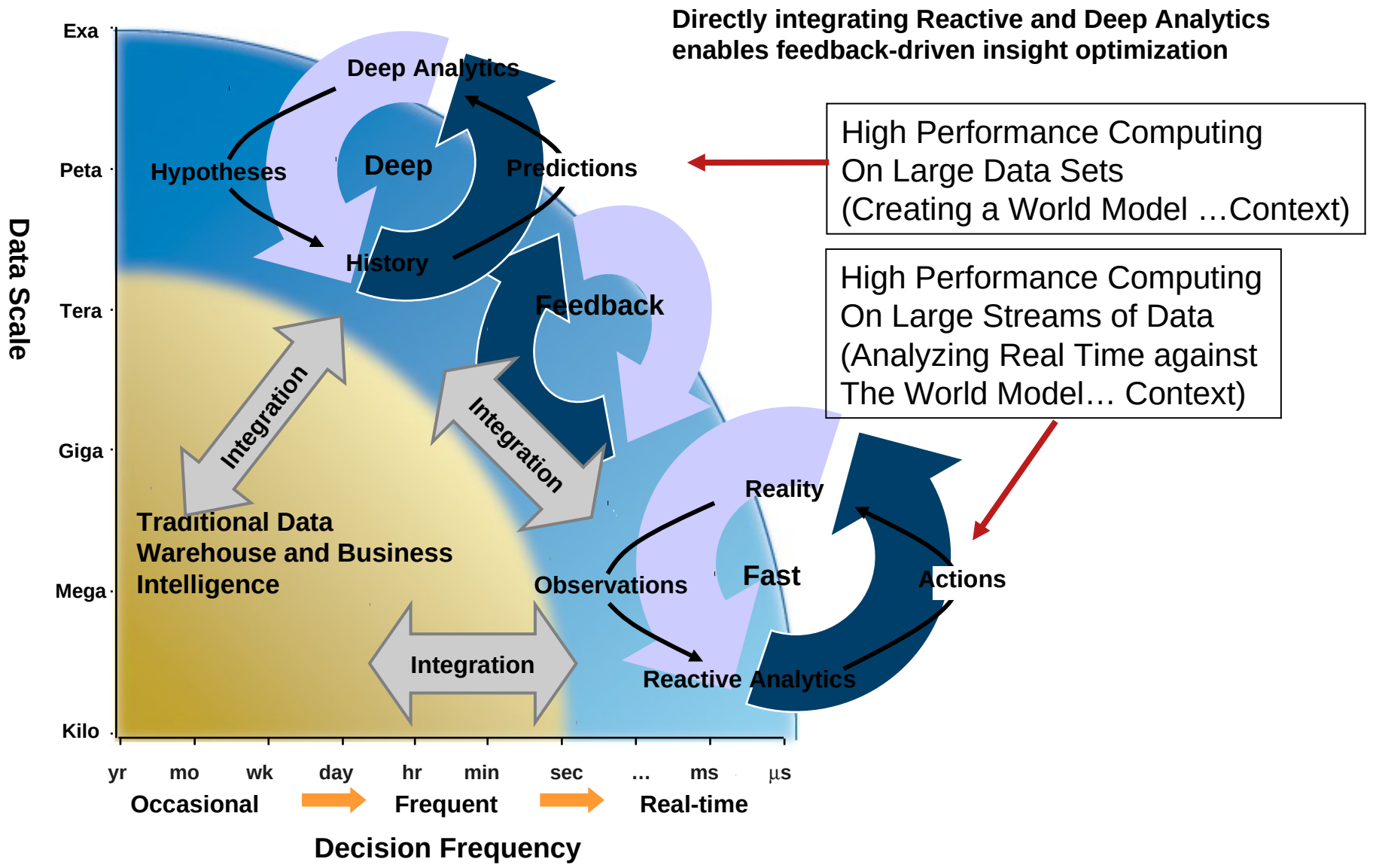
Optimization

- Screen a wide range of design options to choose the “best” one
- Take into account the variability of “things”

Decision

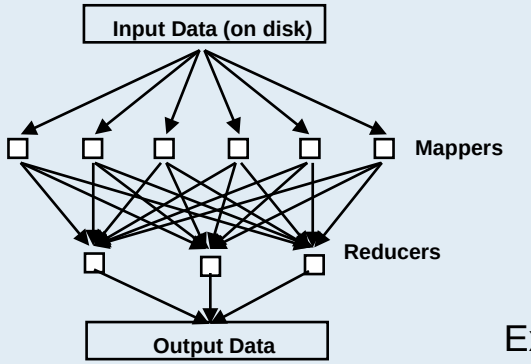
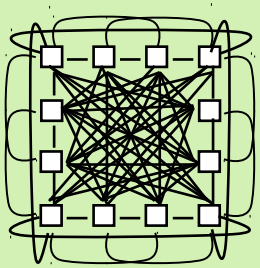
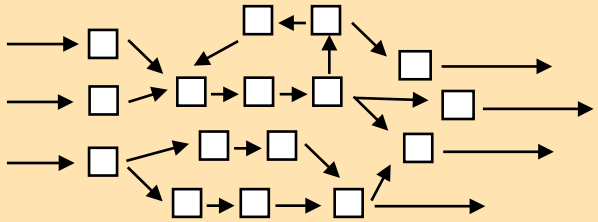
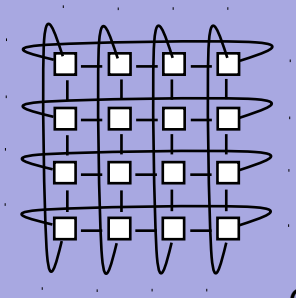
- Assist decision making, with real time constraints

Maximum Insight by Combining Deep and Reactive Analytics



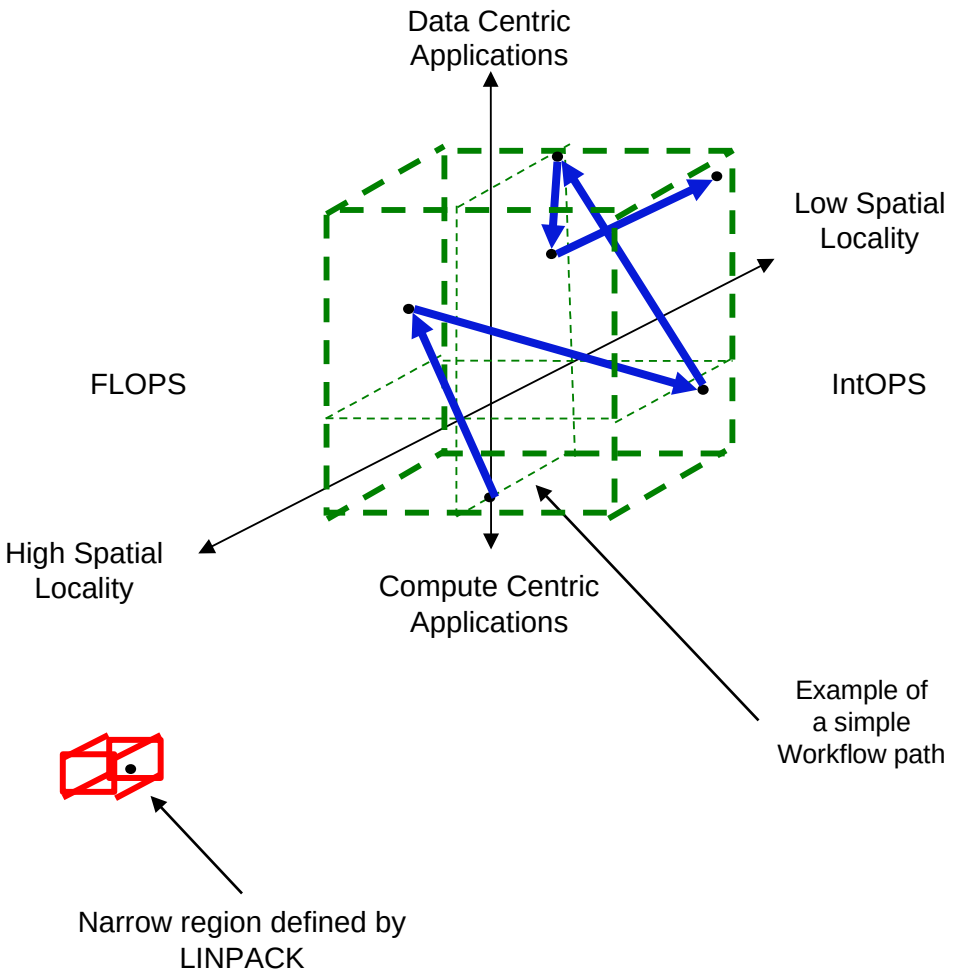
Styles of Massively Parallel Systems

IBM

	Embarrassingly Parallel	Network Dependent
Random Comms	**Data Intensive: Data at Rest** **JAQL, Java**  (*pre-partitioned) **Data at Rest*:** **High Volume** **Mixed Variety** **Low Velocity** Extreme Scale-out	**Data and Compute Intensive (Large Address Space)** **C/C++, Fortran, UPC, SHMEM, MPI, OpenMP**  **Data is Moving:** **Long Running** **All Data View** **Small Messages** Discrete Math Low Spatial Locality
Structured Comms	**Data Intensive: Streaming Data** **SPL, C, Java** **Data in Motion:** **High Velocity** **Mixed Variety** **High Volume***  Reactive Analytics Extreme Ingestion (*over time)	**Compute Intensive (Data Generators)** **C/C++, Fortran, MPI, OpenMP**  **Data is Generated:** **Long Running** **Small Input** **Massive Output** **= compute node** Generative Modeling Extreme Physics

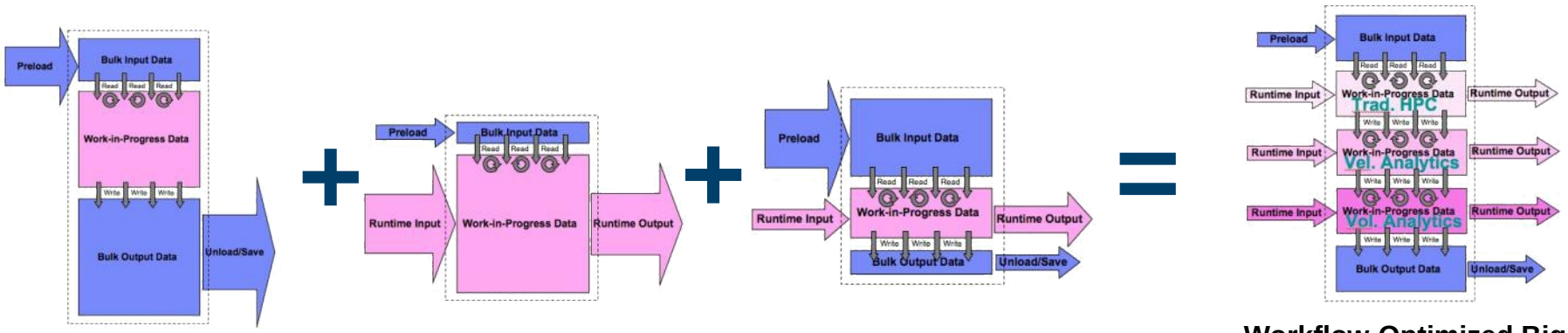
Workflow Characteristics

- Challenge is to determine the best design region boundaries
- Benchmarks are useful but do not tell the entire story – provide measurable dimensions
- Need to examine **workflows** to get the full picture of requirements
- Need to predict how **future** applications will behave across entire workflows
- Need to understand how **multiple** workflows run simultaneously across system(s)

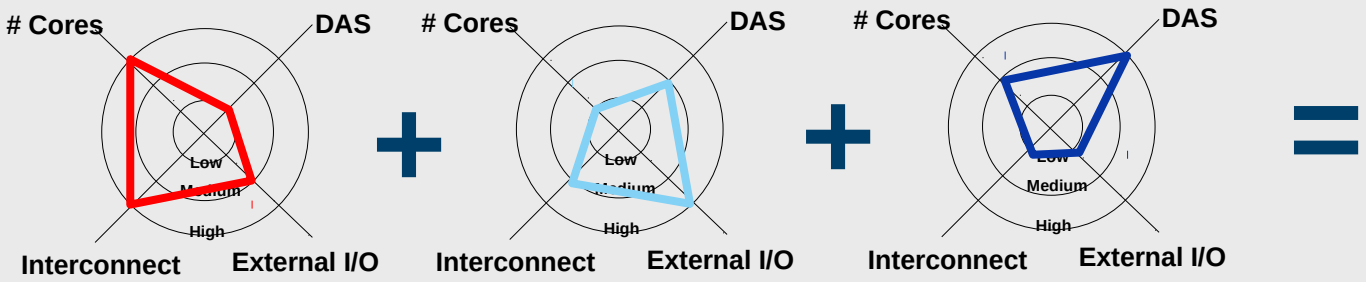


9

Future architecture must handle data-centric locality issues

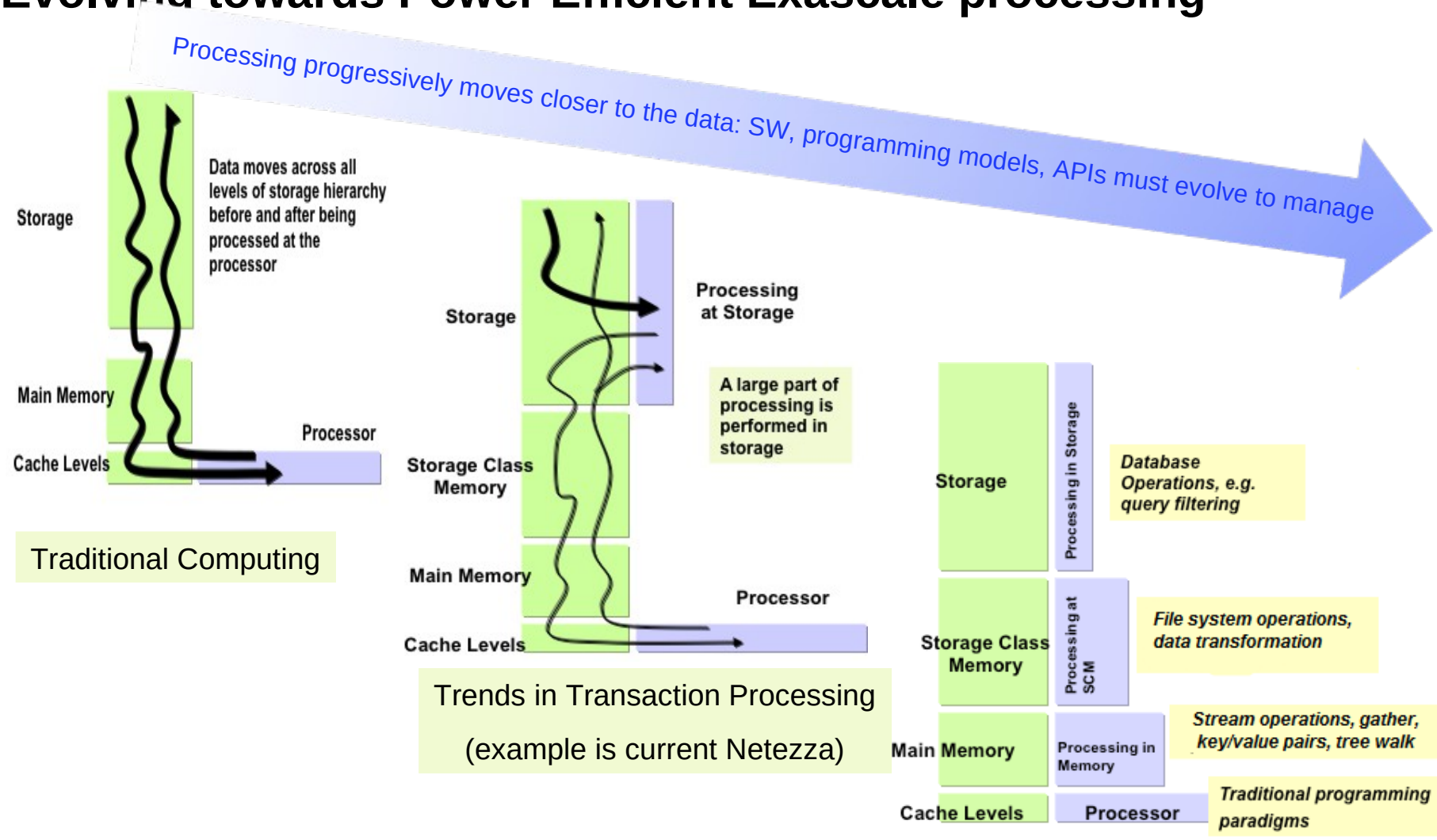


Traditional HPC | **High Velocity Workloads** | **High Volume Workloads** | **Workflow-Optimized Big Data/Big Workloads**



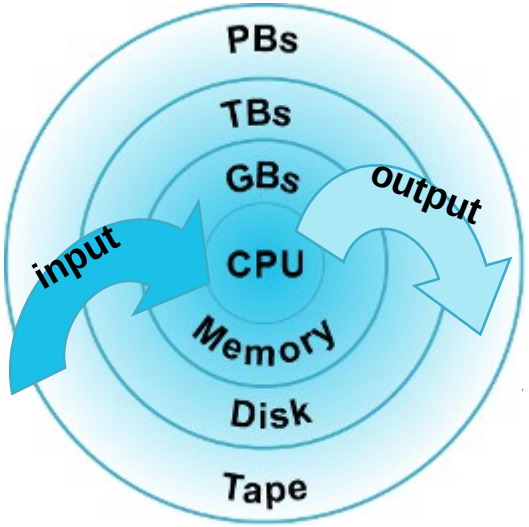
Optimization throughout the hardware and software stack from data ingest, thru run-time and output enables architectural tradeoffs to deliver business value

Evolving towards Power Efficient Exascale processing



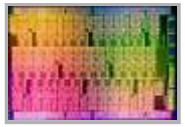
Data-centric Architecture for Performance

Old Compute-Centric Model



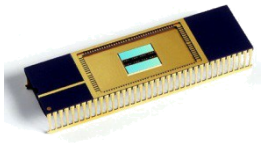
Data lives on disk and tape
Move data to CPU as needed
Deep Storage Hierarchy

Manycore



FPGA

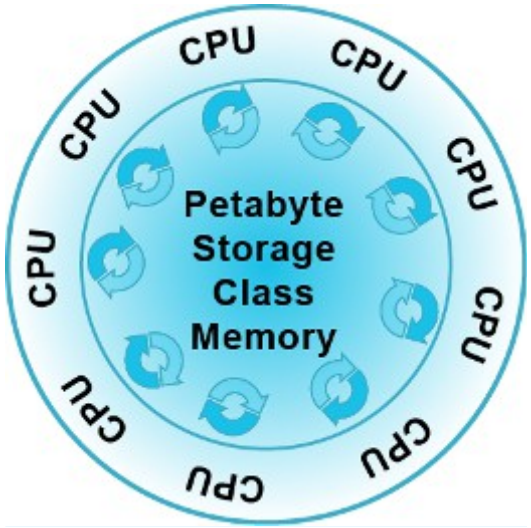
Massive Parallelism
Persistent Memory



Flash

Phase Change

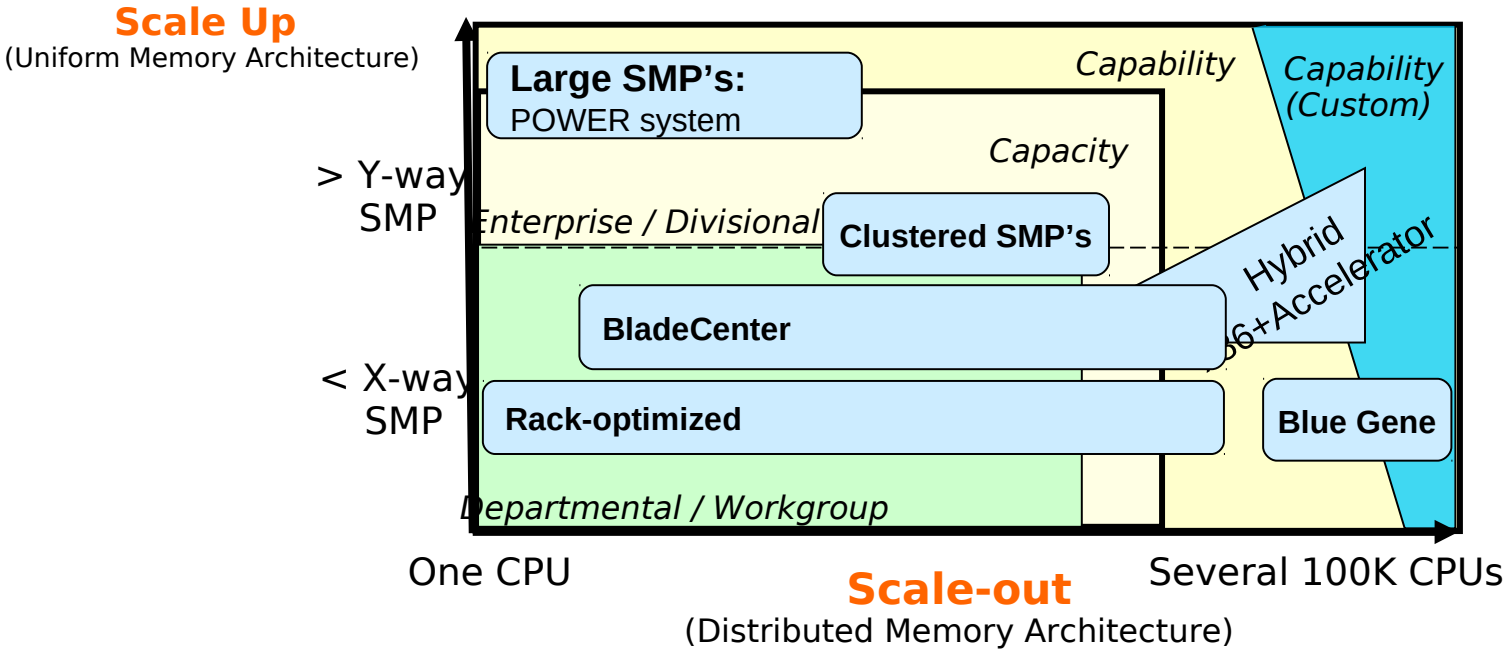
New Data-Centric Model



Data lives in persistent memory
Many CPU's surround and use
Shallow/Flat Storage Hierarchy

HPC system offering

- 2011-15 : clusters of homogeneous SMP nodes (up to 64-128 cores/node)
  - [accelerators: GPGPU/Intel Phi/FPGA]
  - key point: Flops versus Bytes (memory/network), versus Watts/Square Meters, versus RAS, versus programming model cost
- From 2015 : clusters of heterogeneous SMP node introduction
  - Many cores: AMD Fusion, IB, GPGPU and FPGA integration)
  - new memory technologies (data centric)



IBM Technical Computing portfolio

Power Systems™
Engine for faster insights

PureSystems™
Integrated expertise for improved economics

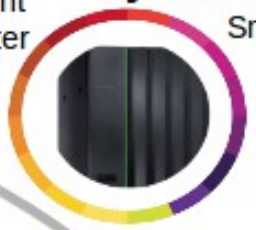
System x®
Redefining x86

Technical Computing for Big Data

System x iDataPlex®
Fast, dense, flexible

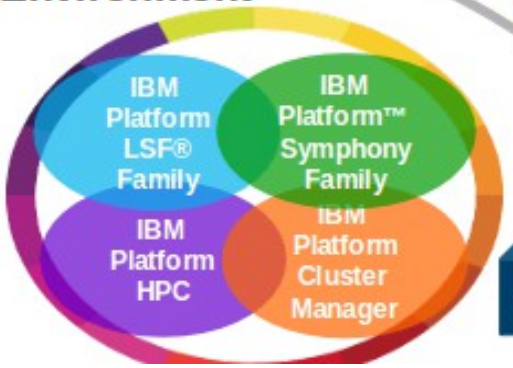
HPC Cloud

Blue Gene®
Extremely fast, energy efficient supercomputer



System Storage®
Smarter storage solutions

Parallel Environment



xCAT

GPFS™

GPFS™ Storage Server

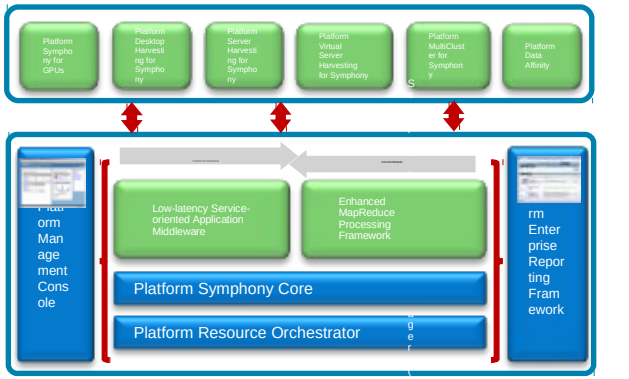
Big data storage

Intelligent Cluster™
Factory-integrated, interoperability-tested **system** with compute, storage, networking and cluster management

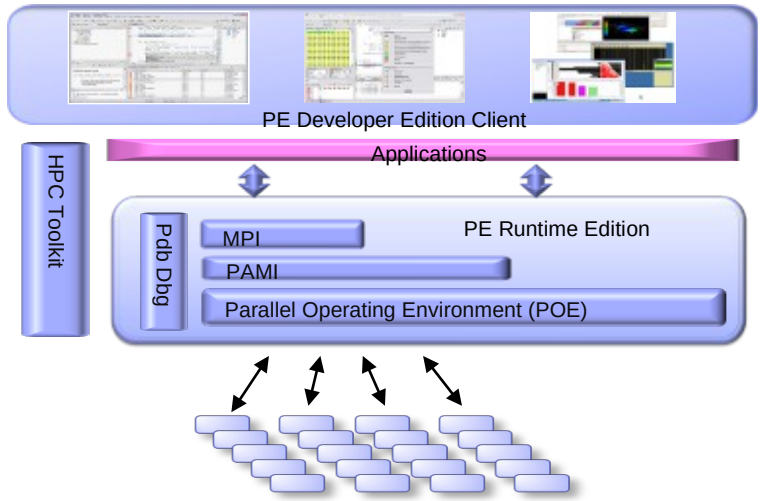
IBM Technical Computing brings a powerful product portfolio for High Performance Computing

Platform LSF
Workload Management

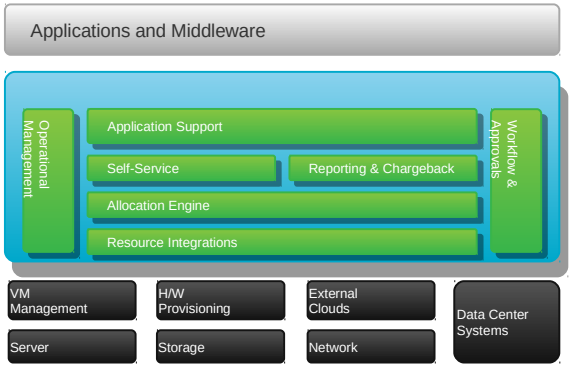
Platform Symphony
HPC Grid Services Management



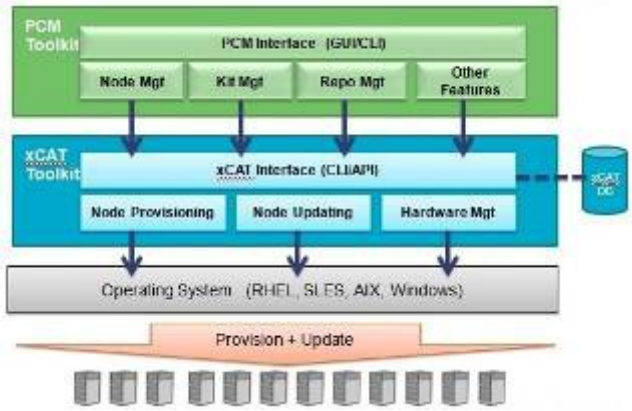
Parallel Environment (PE)



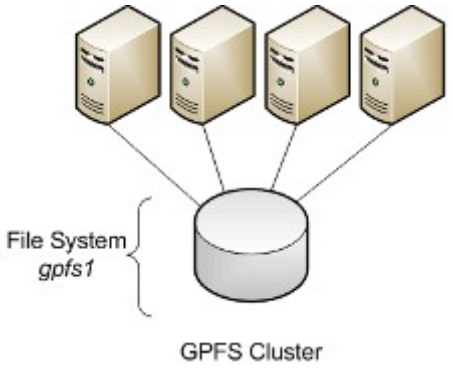
Platform Cluster Manager – Advanced Edition
Dynamic HPC Infrastructure Management



xCAT + Platform Cluster Manager
Systems Management and Provisioning



General Parallel File System (GPFS)
File System Management



Hybrid Systems

Optimizing for Workflows

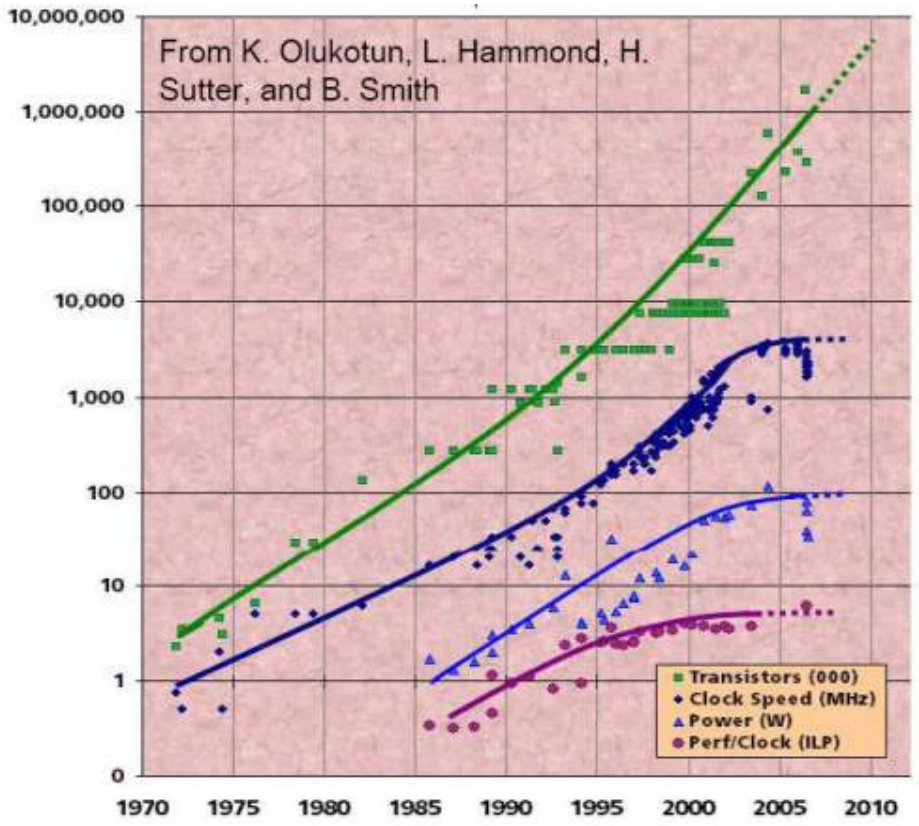
Microprocessor Clock Speed (anti) Trend

- Clock Speed and Power Consumption used to increase with the numbers of transistors and was thus associated with Moore's Law

Clock Speed is not increasing any more because it involves a power consumption and dissipation increase that manufacturers are not able to follow.

Power evolves like frequency^3

Better having many cores running at a lower frequency

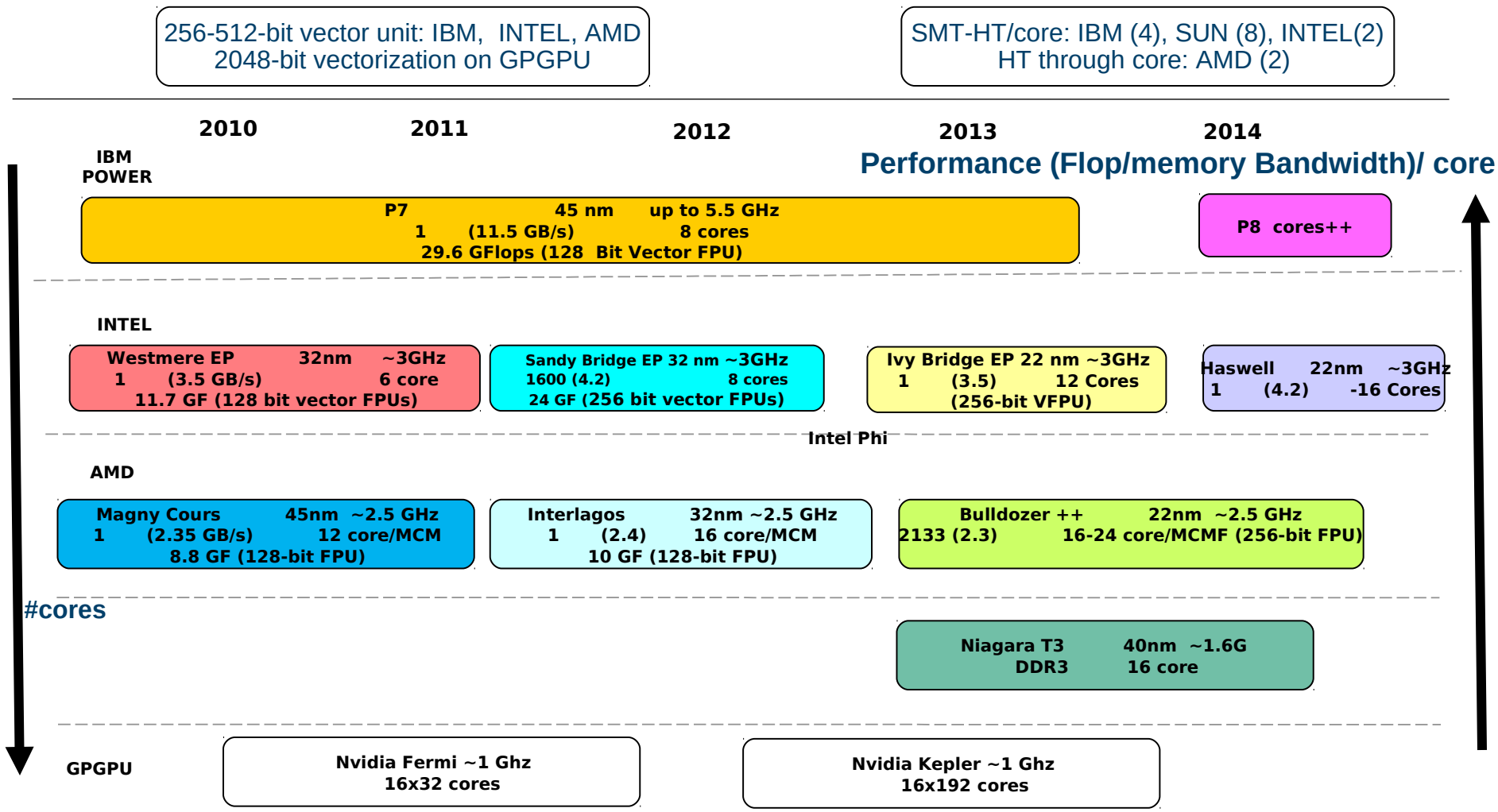


Performance drivers for HPC systems

- Clock speed is flat therefore :
  - more execution units, more cores, more nodes, accelerators
  - While trying to feed the beast and keep improving its IPC
- SIMD
  - 2 way Double Precision (SSE)
  - 4 way Double Precision (AVX)
  - 8 way Double Precision (Xeon Phi)
  - 32 way Double Precision (nVIDIA GPUs)
- Cores
  - 8 cores/Xeon Sandy Bridge
  - Doubling every so many years
- Features
  - Simultaneous Multi Threading
  - Transactional memory
- Accelerators
  - GPU, Xeon Phi, FPGA

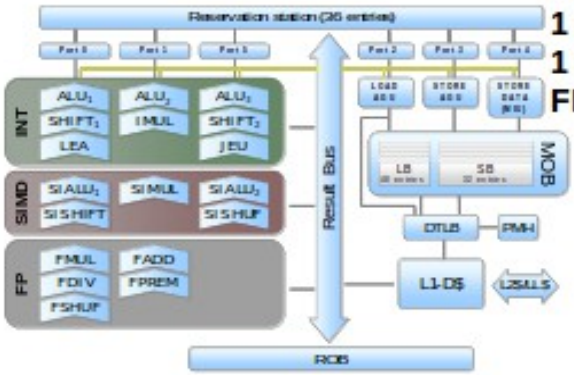
Processor Roadmap Performance Trends

(All the number presented here are estimates)

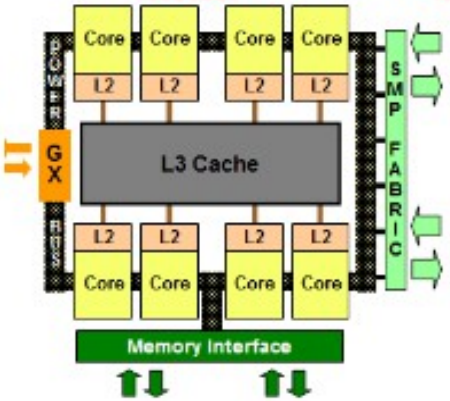


Processor architectures

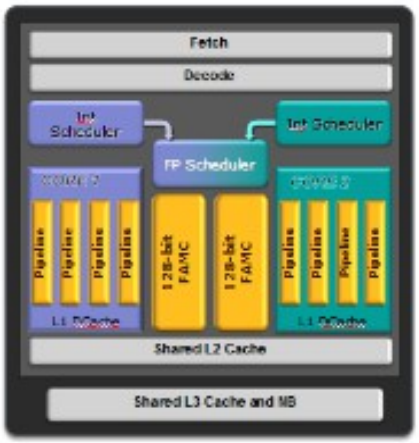
Intel x86 - HT
1 FP 256 bit Mul Unit
1 FP 256 bit Add Unit
FMA in Haswell

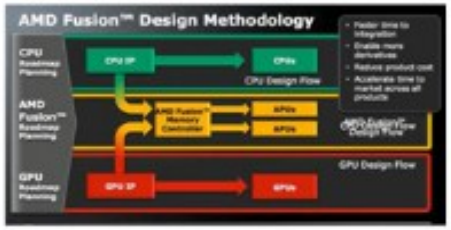


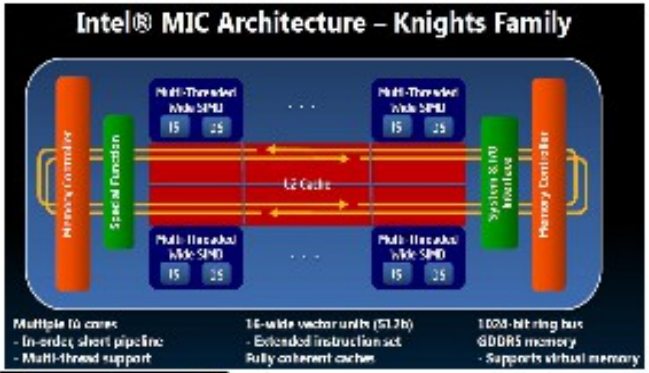
Power 7 – SMT 4
2 vector 128 bit FMA Units



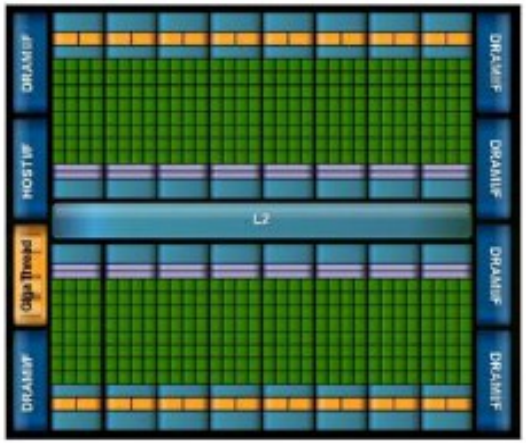
AMD - HT
2 FMA 128b shared Unit





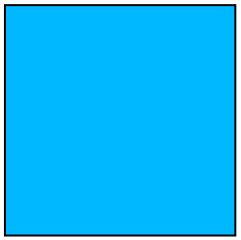


Nvidia Fermi
16 x FMA 32x8bits units

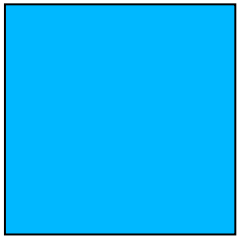


Intel Phi
Real SMT
1 FMA 512b
1 scalar FMA

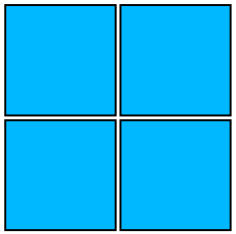
History of supercomputers architecture



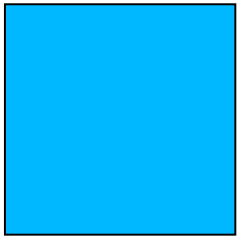
Single core



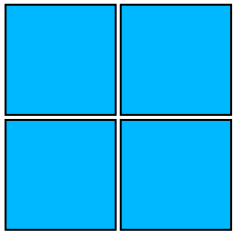
Single core



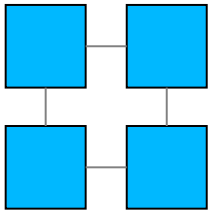
Homogeneous multi-core



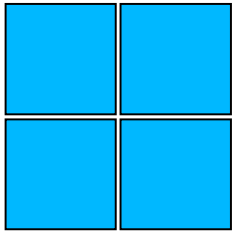
Single core



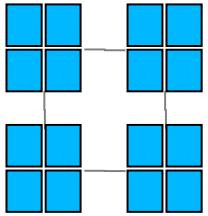
Homogeneous multi-core



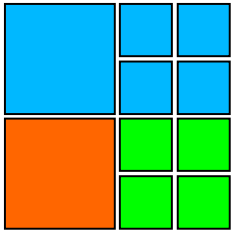
Clusters of Single core nodes



Homogeneous multi-core

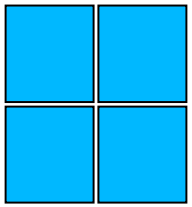


Clusters of Multicore nodes

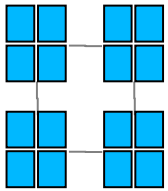


Heterogeneous Many core

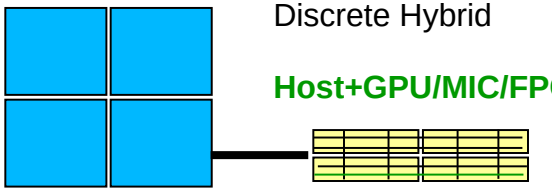
History of supercomputers architectures



Homogeneous multi-core

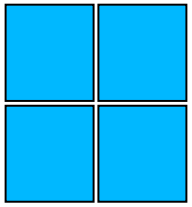


Clusters of homogeneous multi-core nodes

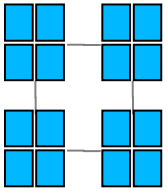


Discrete Hybrid

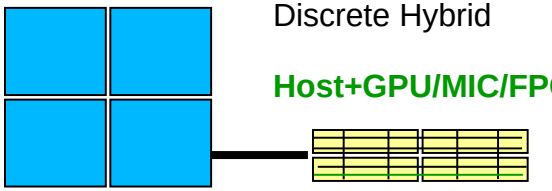
Host+GPU/MIC/FPGA



Homogeneous multi-core

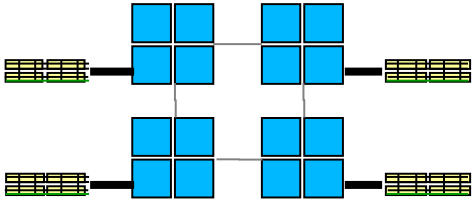


Clusters of homogeneous multi-core nodes

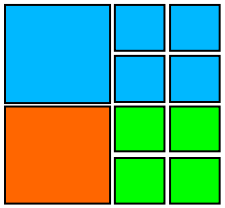


Discrete Hybrid

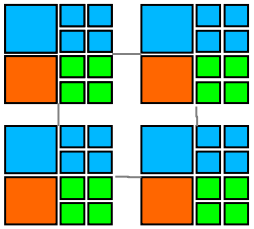
Host+GPU/MIC/FPGA



Clusters of discrete hybrid nodes



Heterogeneous many-core



Clusters of heterogenous many-core nodes

Hybrid computing

Combine general purpose cores/servers with :

(almost) general purpose, small, energy efficient compute cores

special purpose, fixed function cores/chips for better compute thruput

Along the data path:

- Active memory cubes,
- Near storage FPGAs
- IO Space Stream processors

Patterns for Hybrid Computing

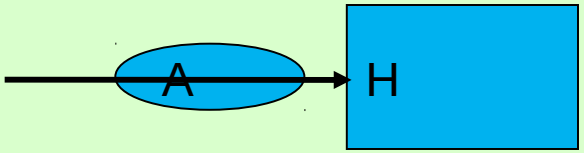
H — Host – where the control code / setup runs

A — Accelerator – specialized core or system

Ingest

- Data flows into the Accelerator first
- Tasks are performed on the Accelerator
- Data is shipped from Accelerator to the Host and will not return to the Accelerator

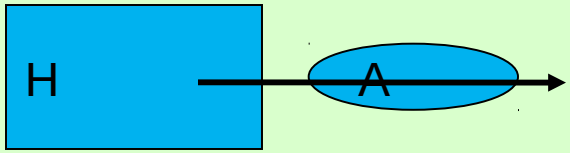
Datapower
WFO
System S



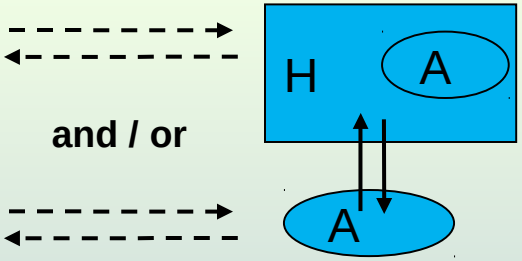
Egest

- Data flows from the Host to the Accelerator
- Tasks are performed on the Accelerator
- Data is shipped onward from Accelerator and does not return to the Host

Crypto (zSeries)



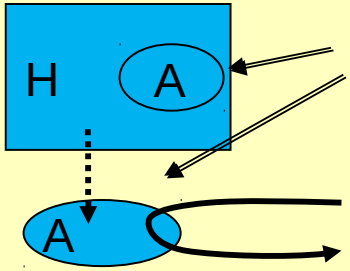
Synchronous Codes
OpenCL
Oil & Gas
Monte Carlo



Collaborative / Roundtrip / Co-processor

- Specific Tasks & Data are offloaded from the Host to Accelerator (or vice versa)
- Tasks are preformed
- Data/Results are returned to the “Off-loader”

*Depends heavily on size of off-load & round-trip communication latency



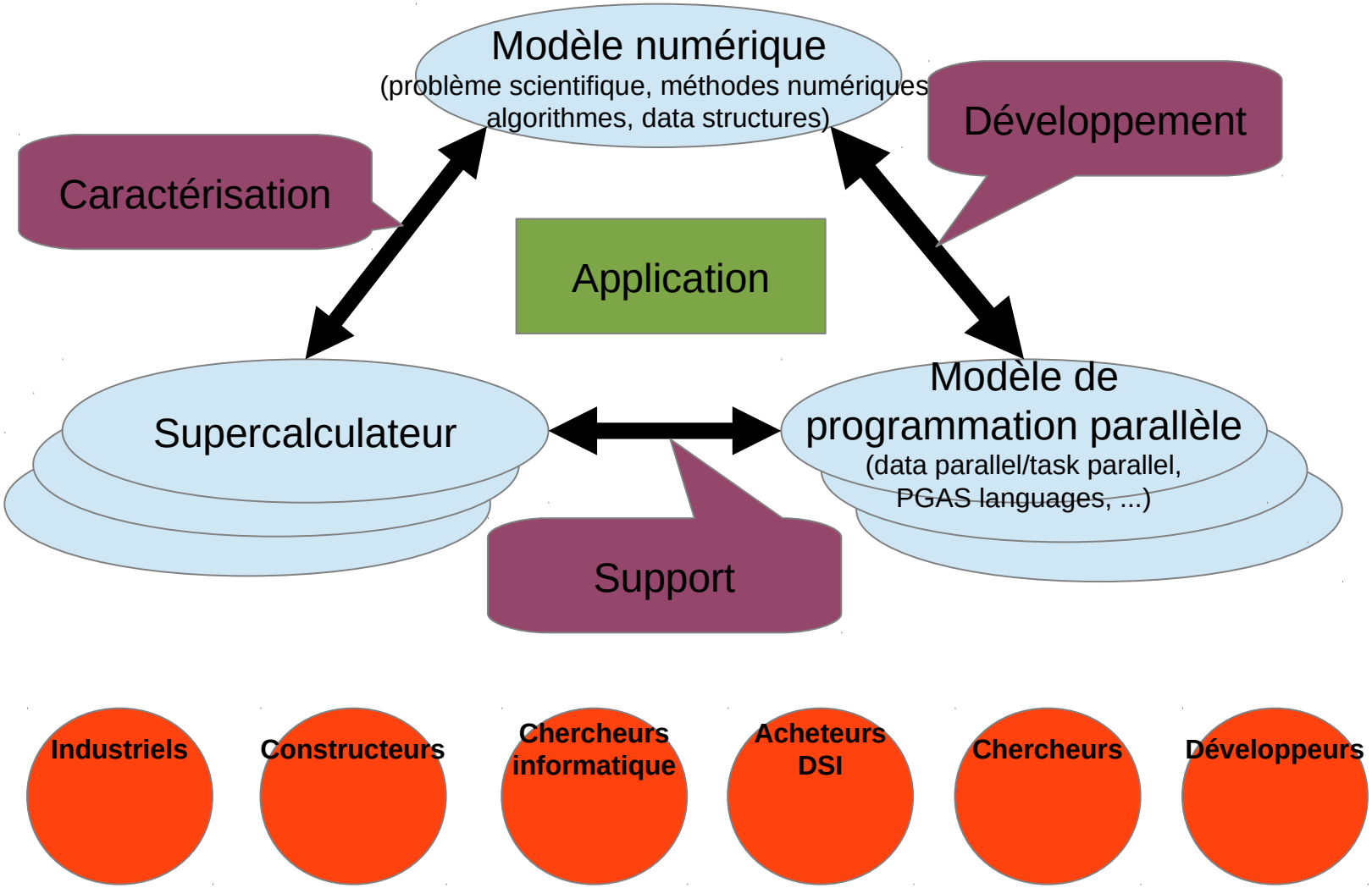
Datapower
Switches
Some Data Mining

Cut-Through

- The host sets up Specific Tasks/Workflows for the Accelerator at startup, and then gets out of the way
- Data flows into the accelerator
- The predefined Tasks/Workflow is preformed on by accelerator
- Data/Results are returned from the accelerator without requiring transfer to the host

Modèles de programmation parallèle

Schéma



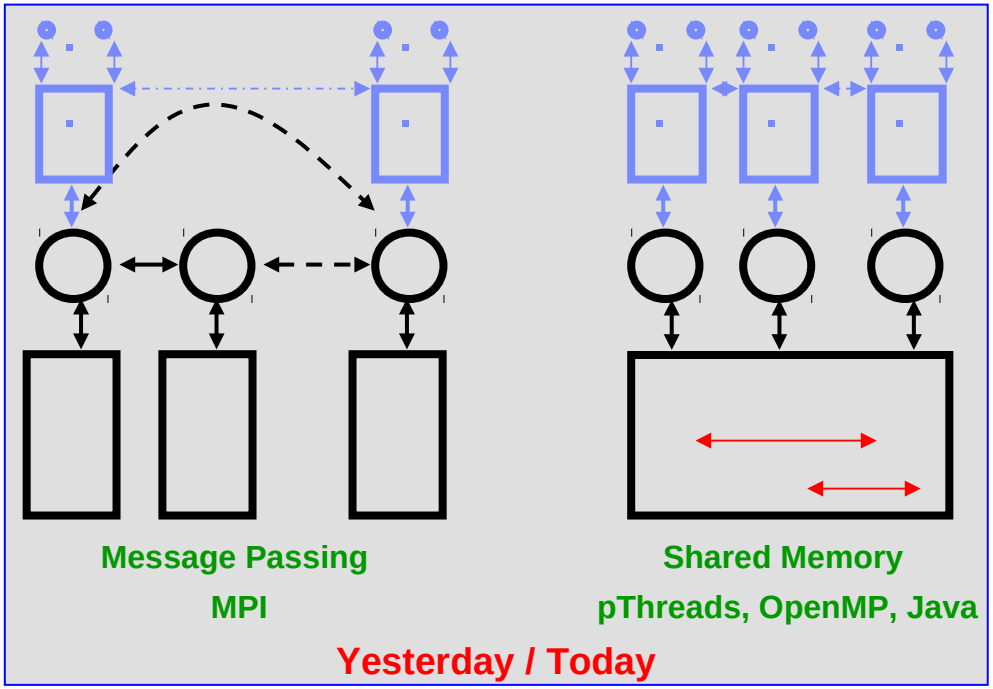
Modèles de programmation parallèle

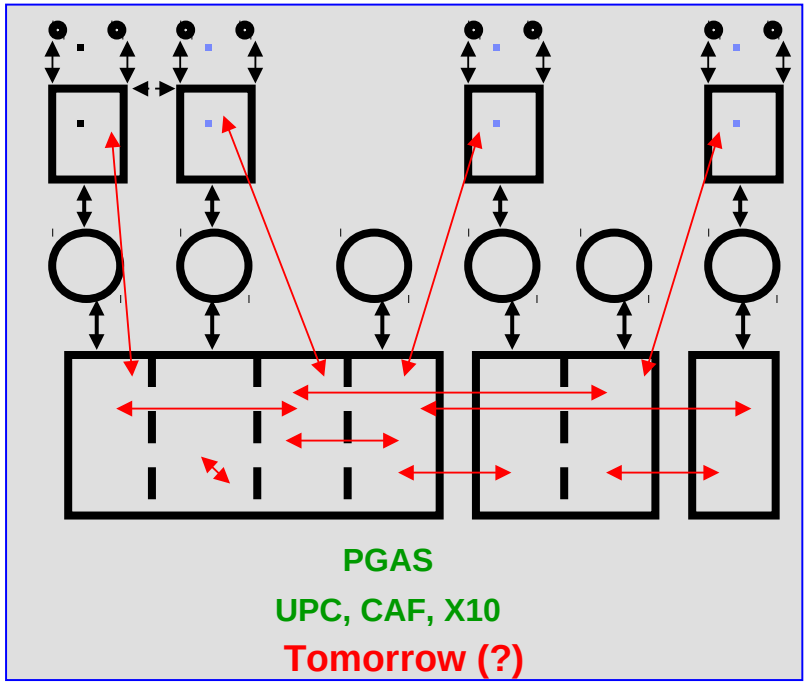
- A quoi ça sert ?
- Quels sont les modèles actuels ?
- Comment choisir ?

Modèles de programmation parallèle

- Idéalisation de la plateforme matérielle
  - Cpus
  - Mémoire
  - Réseau d'interconnection
- Modèle d'exécution
  - Parallélisme de données
  - Parallélisme de tâches
    - Avec dépendences de données
- Modèle mémoire
  - Mémoire partagée
  - Mémoire distribuée
  - Mémoire partagée partitionnée (PGAS)
- Mode d'expression du parallélisme
  - Appels à une API
  - Annotations d'un programme séquentiel (pragma)
  - Language

Programming Models and Languages

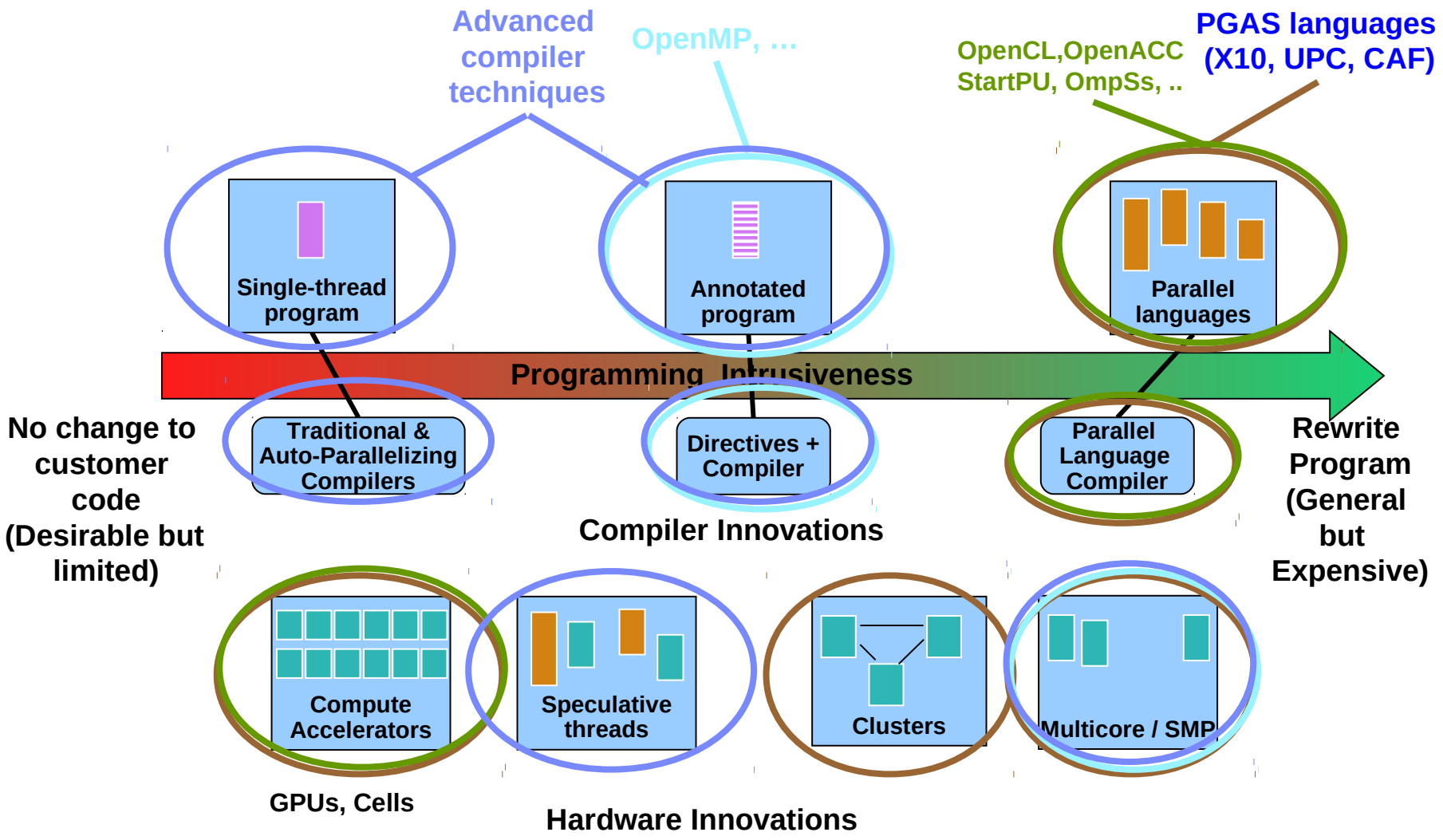




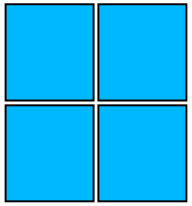
- Computation is performed in multiple places.
- A place contains data that can be operated on remotely.
- Data lives in the place it was created, for its lifetime.

- A datum in one place may reference a datum in another place.
- Data-structures (e.g. arrays) may be distributed across many places.
- Places may have different computational properties

Approaches to programming multicore & hybrid systems

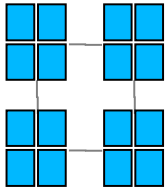


Supercomputers architectures and programming models



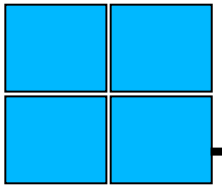
Homogeneous multi-core

OpenMP, Intel TBB, pthreads, Cillk++



Clusters of homogeneous multi-core nodes

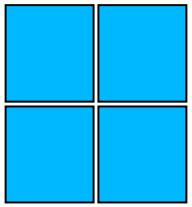
OpenMP, Intel TBB, pthreads, Cillk++, **MPI, UPC**



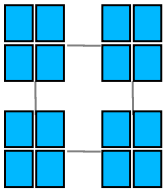
Discrete Hybrid

Host+GPU/MIC/FPGA

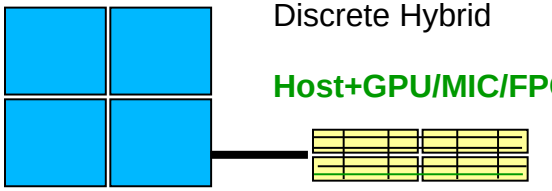
OpenMP, Intel TBB, pthreads, **CUDA, OpenCL, OmpSs**



Homogeneous multi-core

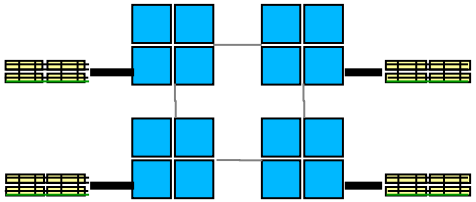


Clusters of homogeneous multi-core nodes



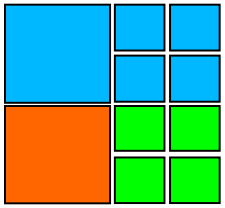
Discrete Hybrid

Host+GPU/MIC/FPGA

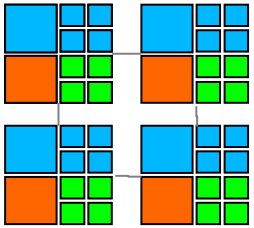


OpenMP, Intel TBB, pthreads, CUDA, OpenCL, **MPI, StarPU, OmpSs, X10**

Clusters of discrete hybrid nodes



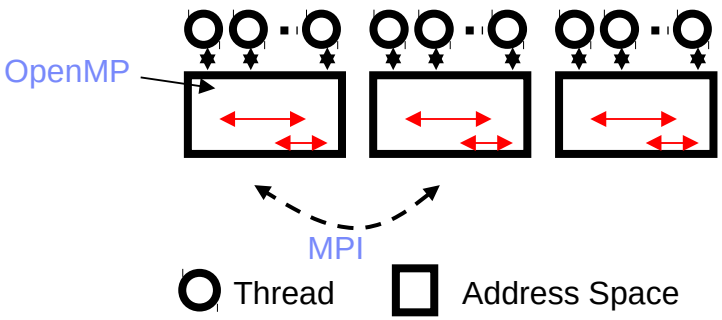
Heterogeneous many-core



MPI, OpenMP ?

Clusters of heterogenous many-core nodes

Standard HPC model: MPI + OpenMP



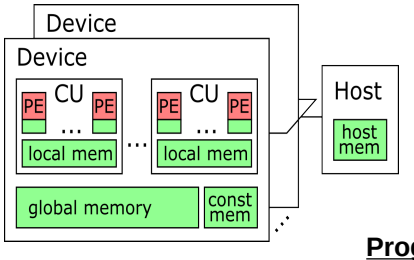
- Paradigm often used for highly scalable applications
  - Typically using domain decomposition for the MPI part
  - And loop level parallelism for OpenMP
- Advantages:
  - Matches current hardware architectures: clusters of homogeneous SMP nodes
    - MPI messages to communicate inside/between nodes
    - OpenMP threads to share memory/work inside nodes
  - Allows to efficiently use high number of cores
    - IBM BG/Q Sequoia : 1572864 cores !
  - I/O operations can be managed by MPI layer, parallel libraries,…
- But…
  - Scalability limits can be reached with hundreds thousands of tasks/threads
  - Data partitioning results in replication of data (inefficient use of memory)
  - Cannot access accelerators (nodes with CPUs + GPGPUs,…)

OpenCL : http://www.khronos.org/opencl

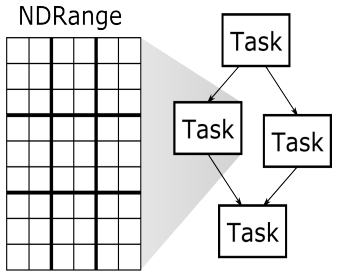
OpenCL = portable data/task parallel language with specification of 4 parts:

- **A platform model**
  - Defines an abstract view of the underlying hardware
- **A programming model**
  - Defines the general approach to express your algorithm
  - „Kernels“ are executed on „devices“
- **A programming language**
  - Specifies the language for the compute kernels
- **A runtime environment**
  - Sets up the compute devices and drives the computations

Platform model



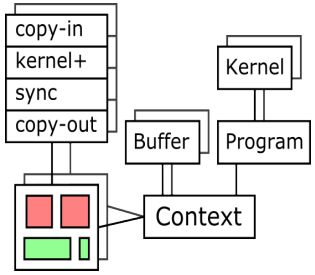
Programming model



Programming language

```
__kernel void k( global int2
*a )
{
    private int i =
        get_global_id(0);
    a[i] = exp( a[i] );
    barrier( /*...*/ );
    // ...
}
```

Runtime environment



StarPU : INRIA, LaBRI (http://runtime.bordeaux.inria.fr/StarPU/)

From http://hal.inria.fr/docs/00/41/15/81/PDF/Aug09RENPAR.pdf

$$f = \begin{cases} f_{cpu} \\ f_{gpu} \\ f_{spu} \\ \dots \end{cases}$$

FIG. 1: Une codelet et ses différentes implémentations

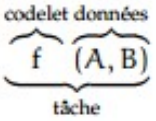


FIG. 2: Une tâche : une codelet appliquée à des données

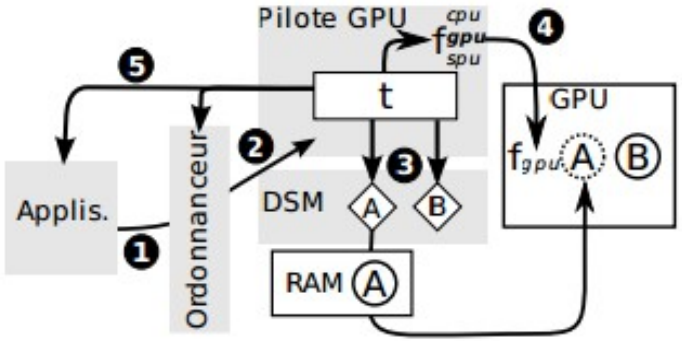


FIG. 3: Cheminement d'une tâche t calculant f(A,B)

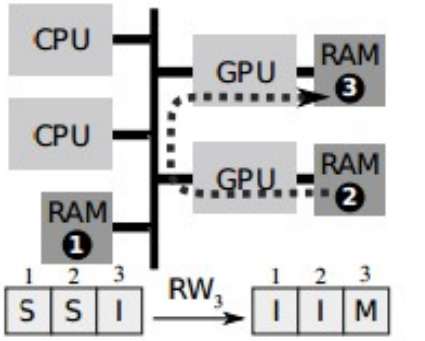


FIG. 4: Protocole de cohérence MSI

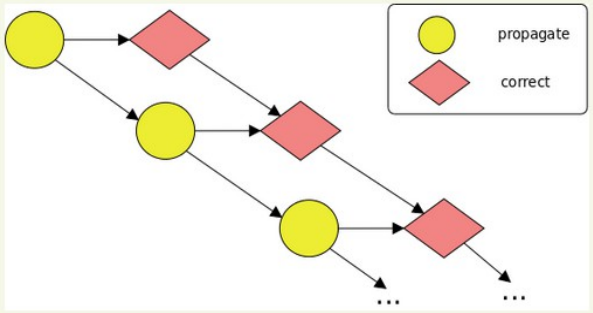
OmpSs : BSC, Barcelona Supercomputing Center

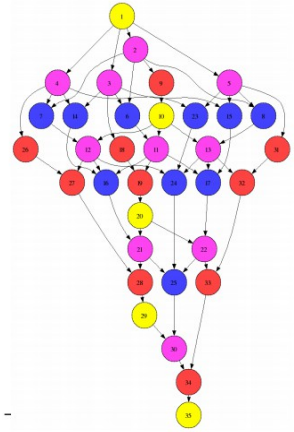
From http://pm.bsc.es/ompss

```
void foo ( int *a, int *b )
{
   for ( i  = 1; i < N; i++ ) {
      #pragma omp task input(a[i-1]) inout(a[i]) output(b[i])
         propagate(&a[i-1],&a[i],&b[i]);

      #pragma omp task input(b[i-1]) inout(b[i])
         correct(&b[i-1],&b[i]);
   }
}
```

This code as generate the following task graph as the loop unfolds:





```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]) ;
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         ssyrk (A[k*NT+i], A[i*NT+i]);
   }
```

```
#pragma omp task inout ([TS][TS]A)
void spotrf (float *A);
#pragma omp task input ([TS][TS]A) inout ([TS][TS]C)
void ssyrk (float *A, float *C);
#pragma omp task input ([TS][TS]A,[TS][TS]B) inout ([TS][TS]C )
void sgemm (float *A, float *B, float *C);
#pragma omp task input ([TS][TS]T) inout ([TS][TS]B)
void strsm (float *T, float *B);
```

Cilk+ : MIT, Intel http://software.intel.com/en-us/intel-cilk-plus

From https://www.research.ibm.com/haifa/Workshops/padtad2009/present/PADTAD.pptx

```
int fib(int n)
{
   if (n < 2) return n;
   int x, y;
   x = cilk_spawn fib(n-1);
   y = fib(n-2);
   cilk_sync;
   return x+y;
}
```

The named ***child*** function may execute in parallel with the ***parent*** caller.

Control cannot pass this ***point*** until all spawned ***children*** have returned.

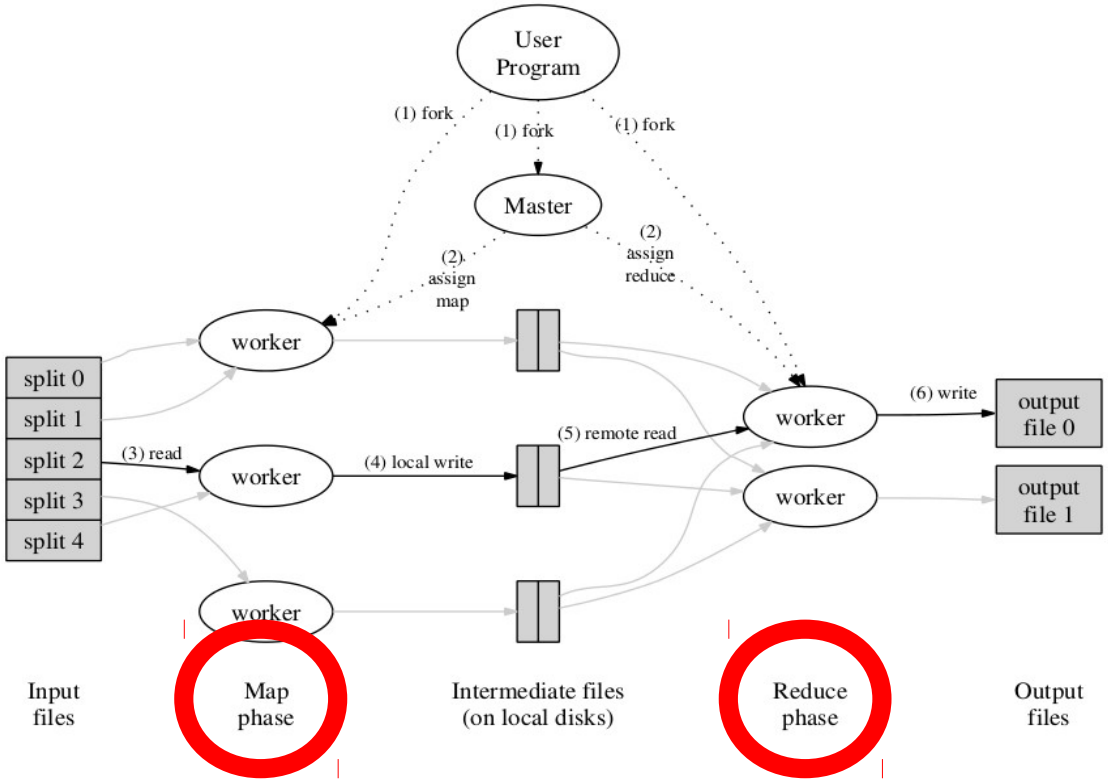
Simple ***cilk_for*** construct for loop parallelism.

```
// indices run from 0, not 1
cilk_for (int i=1; i<n; ++i) {
    cilk_for (int j=0; j<i; ++j)
{
        double temp = A[i][j];
        A[i][j] = A[j][i];
        A[j][i] = temp;
    }
}
```

Google/Apache/Platform Symphony : MapReduce

From http://research.google.com/archive/mapreduce-osdi04.pdf

The programmer provides the (Java) map() and reduce() functions

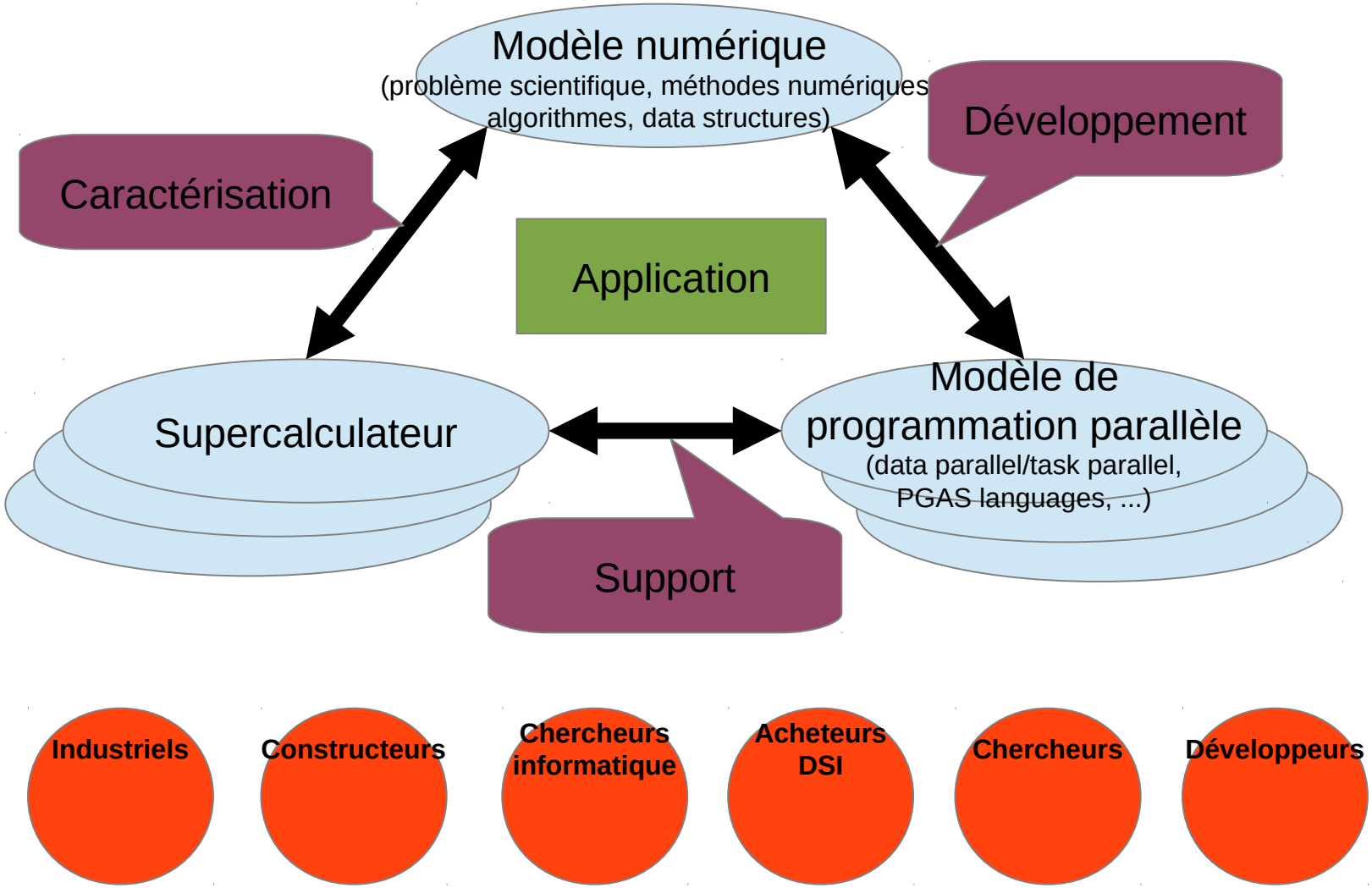


Critères de choix d'un modèle de programmation parallèle

- Durée de vie
  - prévue pour l'application : <1 an ou >10 ans ?
  - Du modèle lui même
    - Soutenu par qui ?
- Coûts
  - Entrée
  - Sortie
  - Apprentissage
- Performances
  - scalabilité
- Facilité
  - Apprentissage
    - Développement initial
    - Développeurs et développements futurs
  - Extension d'un langage existant ou nouveau langage ?
- Diffusion
  - Disponible sur un grand nombre de plateformes ?
  - Portabilité, ex: CUDA=nVIDIA seulement
  - Portabilité des performances

Caractérisation des applications

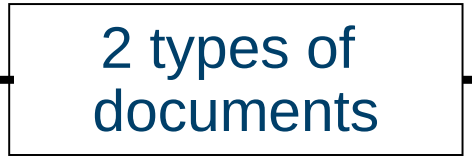
Schéma



Application characterization

- Why is it needed ?
  - Check that the application is working well
    - Today : exploits the current hardware
      - Identify bottlenecks
      - Know your expectations
    - Tomorrow : make an educated guess
  - Give feedback about either the programming model used and/or the efficiency of the numerical model on real hardware
  - Efficiency can be the energy efficiency : how many kWh per run ?
  - Estimate the benefits of changing the application
  - And the workload to do so
- Method
  - Classification
  - Benchmark
  - Profile

HPC Application Characterization



Application Summary

- audience
  - all people:
- contents
  - General information: application, workload, targets, needs, challenges, …
  - Technical overview
    - technical contact
    - systems already used
    - numerical and programming models overview
    - classifications
  - links towards deeper technical details
    - application characterization document
    - benchmark results
    - scientific documentations
    - …
- author
  - a HPC person by not necessary by an expert, no deeper knowledge is requested
- workload
  - ~ 1-2 days
- format
  - short (less than 2 pages), format to be defined: .ppt, doc, table, …

Application Characterization

- audience
  - Engineers/Users/Developer/production people
- contents
  - all the technical details
    - Assessment details
    - General information
      - segment, contacts, etc
      - current and future workload
    - technical Characteristics
      - Source code
      - Addressing & Computation
      - I/O
      - Libraries
      - …
    - Packaging
    - Compilation and execution env
    - Test Cases
    - Optimizations, tricks, etc …
    - Numerical models and algorithms
    - development roadmap
    - other similar applications
    - …
  - performance benchmarks, analysis, profile, ...
- author
  - Developer or users expert in Computer Science (or collaboration)
- workload
  - several weeks
- format
  - large, format to be defined: .doc

The Landscape of Parallel Computing Research: A View From Berkeley http://view.eecs.berkeley.edu/wiki/Main_Page

Dense Linear Algebra	Data are dense matrices or vectors. General applications use unit-stride memory accesses
Sparse Linear Algebra	Data sets include many zero values. Data is usually stored in compressed data structures and bandwidth requirements to access all of the nonzero values; indexed loads and stores.
Spectral Method / FFT	Data are in the frequency domain, as opposed to time or spatial domains. Typically, spectral methods use multiple butterfly stages, which combine multiply-add ops and a specific pattern of data permutations, with all-to-all communications for some stages
N-Body Methods	Depends on interactions between many discrete points. Variations include particle-particle methods, where every point depends on all others, leading to an O(N2) calculation, and hierarchical particle methods.
Structure Grids or Lattice- Boltzman	Represented by a regular grid; points on grids are conceptually udpated together. It has high spatial locality. Updates may be in place or between 2 versions of the grid. The grid may be subdivised into finer grids (Adaptive Mesh Refinement); and the tran
Unstructured grid	An irregular grid where data locations are selected, usially by underlying characteristics of the app. Data point location and connectivity of neighboring points must be explicit. The points on the grid are conceptually updated
MonteCarlo (Quantum MonteCarlo)	Calculations depend on statistical results of repeated random trials. Considered embarrassingly parrallel.
Combinational Logic	parallelism. For example, computing Cyclic Redundancy Codes (CRC) is critical to ensure integrity and RSA encryption for data security
Graph Traversal	applications must traverse a number of objects and examine characteristics of those objects such as would be used for search. It typically involves indirect table lookups and little computation
Graphical Models	applications involve graphs that represent random variables as nodes and conditional dependencies as edges. Examples include Bayesian networks and Hidden Markov Models.
Finite State Machines	represent an interconnected set of states, such as would be used for parsing. Some state machines can decompose into multiple simultaneously active state machines that can act in parallel.

HPC 13 dwarfs – pattern of computation/communication

Dwarf	*Embedded Computing*	*General Purpose Computing*	*Machine Learning*	*Graphics / Games*	*Databases*	*Intel RMS*
1. Dense Linear Algebra (e.g., BLAS or MATLAB)	*EEMBC Automotive*: iDCT, FIR, IIR, Matrix Arith; *EEMBC Consumer*: JPEG, RGB to CMYK, RGB to YIQ; *EEMBC Digital Entertainment*: RSA MP3 Decode, MPEG-2 Decode, MPEG-2 Encode, MPEG-4 Decode; MPEG-4 Encode; *EEMBC Networking*: IP Packet; *EEMBC Office Automation*: Image Rotation; *EEMBC Telecom*: Convolution Encode; *EEMBC Java:* PNG	*SPEC Integer*: Quantum computer simulation (libquantum), video compression (h264avc) *SPEC Fl. Pt.*: Hidden Markov models (sphinx3)	Support vector machines, princpal component analysis, independent component analysis		Database hash accesses large contiguous sections of memory	Body Tracking, media synthesis linear programming, K-means, support vector machines, quadratic programming, *PDE*: Face, *PDE*: Cloth*
2. Sparse Linear Algebra (e.g., SpMV, OSKI, or SuperLU)	*EEMBC Automotive*: Basic Int + FP, Bit Manip, CAN Remote Data, Table Lookup, Tooth to Spark; *EEMBC Telecom*: Bit Allocation; *EEMBC Java:* PNG	*SPEC Fl. Pt.*: Fluid dynamics (bwaves), quantum chemistry (gamess; tonto), linear program solver (soplex)	Support vector machines, principal component analysis, independent component analysis	Reverse kinematics; Spring models		Support vector machines, quadratic programming, *PDE*: Face, *PDE*: Cloth* *PDE:* Computational fluid dynamics
3. Spectral Methods (e.g., FFT)	*EEMBC Automotive*: FFT, iFFT, iDCT; *EEMBC Consumer*: JPEG; *EEMBC Entertainment*: MP3 Decode		Spectral clustering	Texture maps		*PDE*: Computational fluid dynamics *PDE:* Cloth
4. N-Body Methods (e.g., Barnes-Hut, Fast Multipole Method)		*SPEC Fl. Pt.*: Molecular dynamics (gromacs, 32-bit; namd, 64-bit)				
5. Structured Grids (e.g., Cactus or Lattice-Boltzmann Magneto-	*EEMBC Automotive*: FIR, IIR; *EEMBC Consumer*: HP Gray-Scale; *EEMBC Consumer*: JPEG; *EEMBC Digital Entertainment*: MP3 Decode, MPEG-2 Decode, MPEG-2 Encode, MPEG-4 Decode; MPEG-4 Encode; *EEMBC Office Automation*:	*SPEC Fl. Pt.*: Quantum chromodynamics (milc),magneto hydrodynamics (zeusmp), general relativity (cactusADM), fluid dynamics (leslie3d-AMR; lbm), finite element methods (dealII-AMR; calculix), Maxwell's E&M		Smoothing; interpolation		

HPC 13 dwarfs - continued

Dwarf	*Embedded Computing*	*General Purpose Computing*	*Machine Learning*	*Graphics / Games*	*Databases*	*Intel RMS*
hydrodynamics)	Dithering; *EEMBC Telecom*: Autocorrelation	eqns solver (GemsFDTD), quantum crystallography (tonto), weather modeling (wrf2-AMR)				
6. Unstructured Grids (e.g., ABAQUS, FIDAP)			Belief propagation			Global illumination
7. MapReduce (e.g., Monte Carlo)		*SPEC Fl. Pt.*: Ray tracer (povray)	Expectation maximization		MapReduce	
8. Combinational Logic	*EEMBC Digital Entertainment*: AES, DES, IP Packet, IP NAT, Route Lookup; *EEMBC Office Automation*: Image Rotation; *EEMBC Telecom*: Convolution Encode		Hashing		Hashing	
9. Graph Traversal	*EEMBC Automotive*: Pointer Chasing, Tooth to Spark; *EEMBC Networking*: IP NAT, OSPF, Route Lookup; *EEMBC Office Automation*: Text Processing; *EEMBC Java:* Chess, XML Parsing		Bayesian networks, decision trees	Reverse kinematics, collision detection, hidden surface removal	Transitive closure	Natural language processing
10. Dynamic Programming	*EEMBC Telecom*: Viterbi Decode	*SPEC Integer:* Go (gobmk)	Forward-backward, inside-outside, variable elimination, value iteration		Query optimization	
11. Backtrack and Branch +Bound		*SPEC Integer:* Chess (sjeng), network simplex algorithm (astar)	Kernel regression, constraint satisfaction,			
12. Graphical Models	*EEMBC Telecom*: Viterbi Decode	*SPEC Integer:* Hidden Markov models (hmmer)	Hidden Markov models			
13. Finite State Machine	*EEMBC Automotive*: Angle To Time, Cache "Buster", CAN Remote Data, PWM, Road Speed, Tooth to Spark; *EEMBC Consumer*: JPEG; *EEMBC Digital Entertainment*: Huffman Decode, MP3 Decode, MPEG-2 Decode, MPEG-2 Encode, MPEG-4 Decode; MPEG-4 Encode; *EEMBC Networking*: QoS, TCP; *EEMBC Office Automation*: Text Processing; *EEMBC Telecom*: Bit Allocation; *EEMBC Java:* PNG	*SPEC Integer*: Text processing (perlbench), compression (bzip2), compiler (gcc), video compression (h264avc), network discrete event simulation (omnetpp), XML transformation (xalancbmk)		Response to collisions		

The Performance Analysis Method

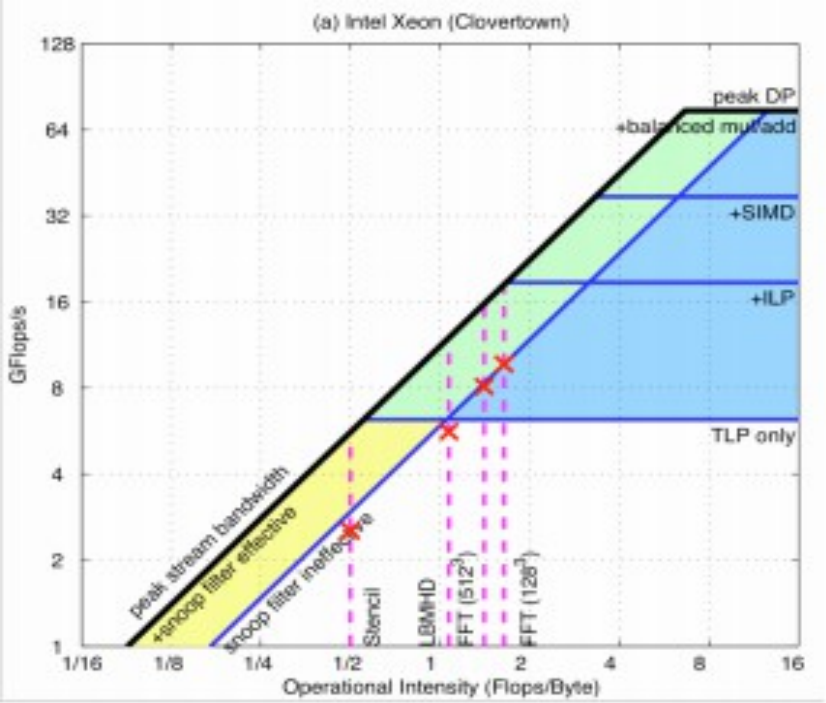
- You need a method and tools to assess the application's performance
- We have our own
  - Instrument the application
  - Split the execution time in IO, communication, computation
  - For the compute part
    - For all hot routines, work out the CPI metric (cycles/instructions)
    - Compare it to a threshold
    - If CPI > threshold try to fix the code
    - NB : threshold is empirical
- There are better ones
  - The roofline model
  - David Levinthal's cycle accounting model

The Roofline model

Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures

Samuel Webb Williams, Andrew Waterman and David A. Patterson

http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-134.html



The cycle accounting methodology

Description

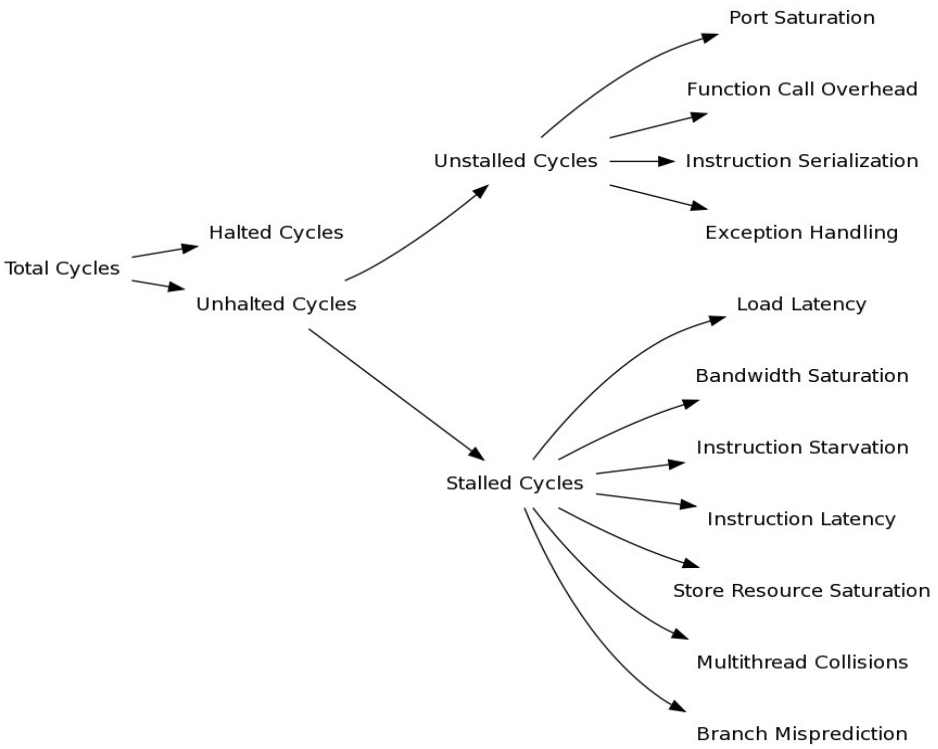
http://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

Where do the cycles go ?

Uses Intel Vtune

Will use

http://code.google.com/p/gooda-visualizer/
https://code.google.com/p/gooda



Know your expectations

- I am running at 9.13 Gflop/s, my cache hit ratio is 92 % !
- Great !
- But am I doing well ?
- Can I do better ?
- What should I expect ?

The Tools

From IBM

- HPC Toolkit from IBM Parallel Environment Developer Edition

From Intel

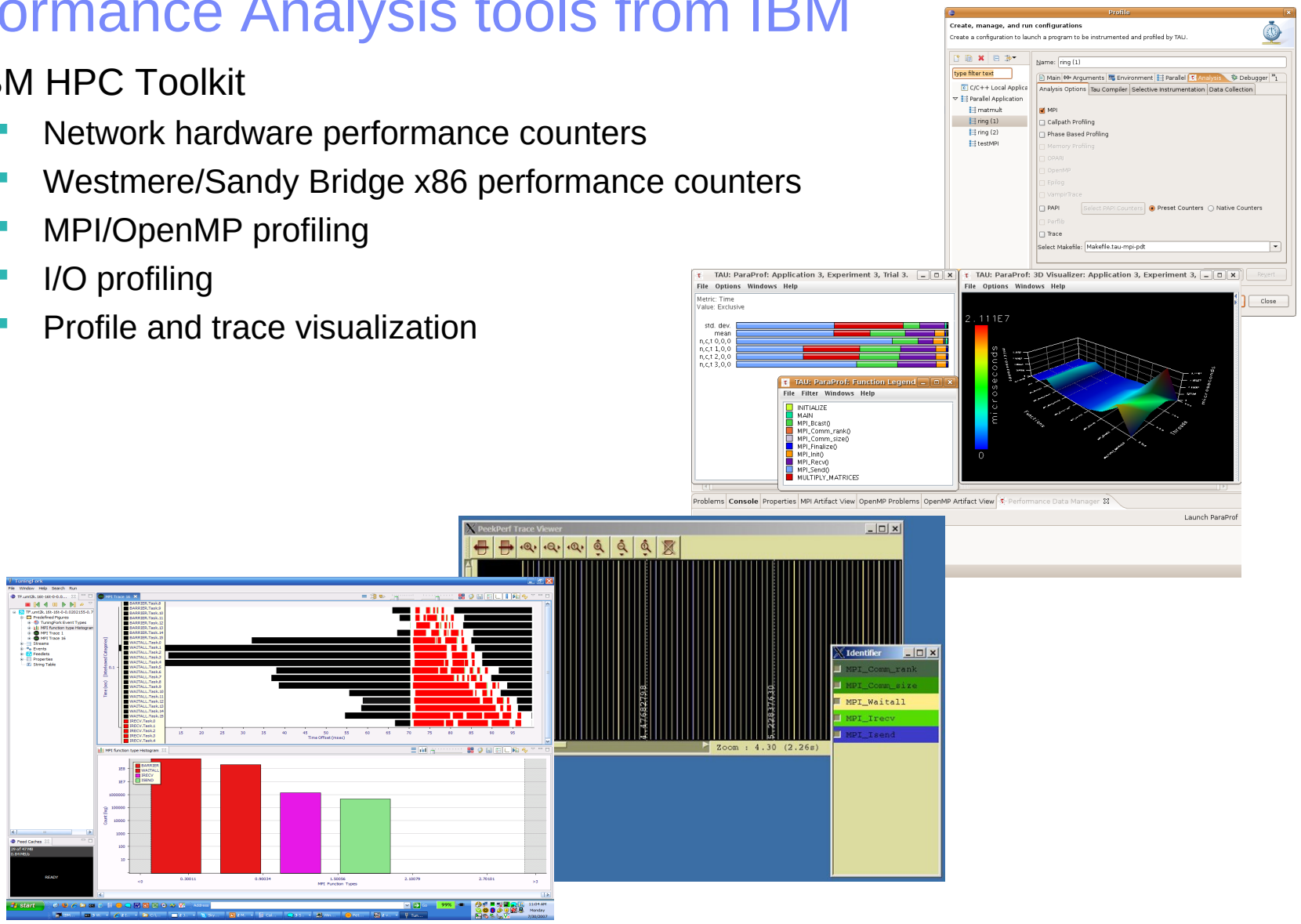
- Intel Vtune
- Intel PCM
- Intel ITAC (for MPI applications)

From Open Source

- perf command (Linux)
- Oprofile (Linux)
- PAPI, libpfm4
- TAU
- Scalasca
- Others

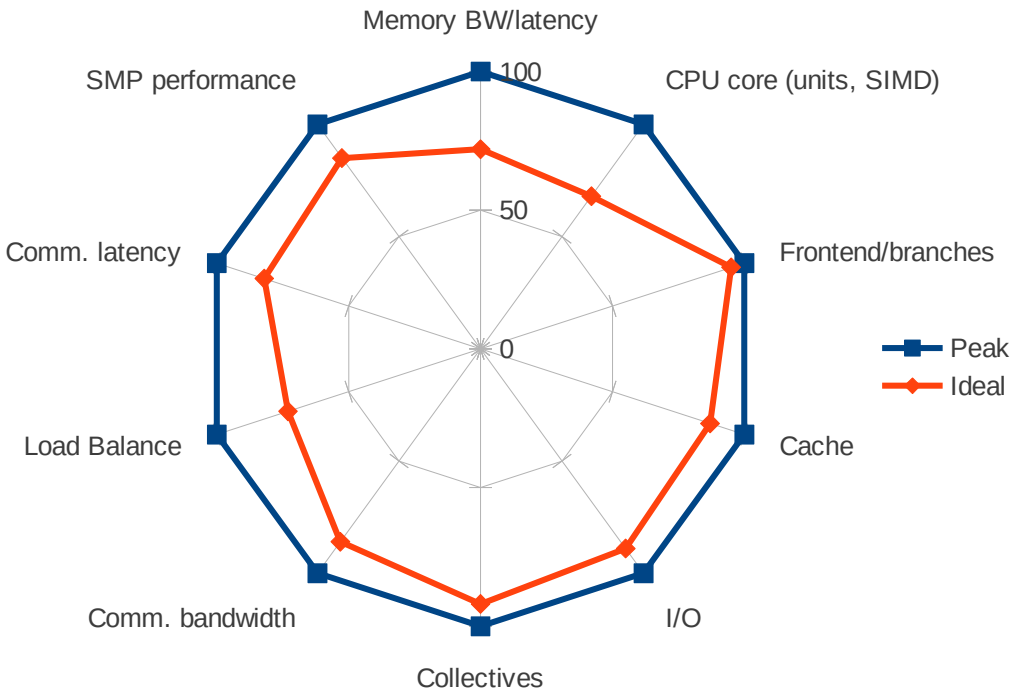
Performance Analysis tools from IBM

- IBM HPC Toolkit
  - Network hardware performance counters
  - Westmere/Sandy Bridge x86 performance counters
  - MPI/OpenMP profiling
  - I/O profiling
  - Profile and trace visualization



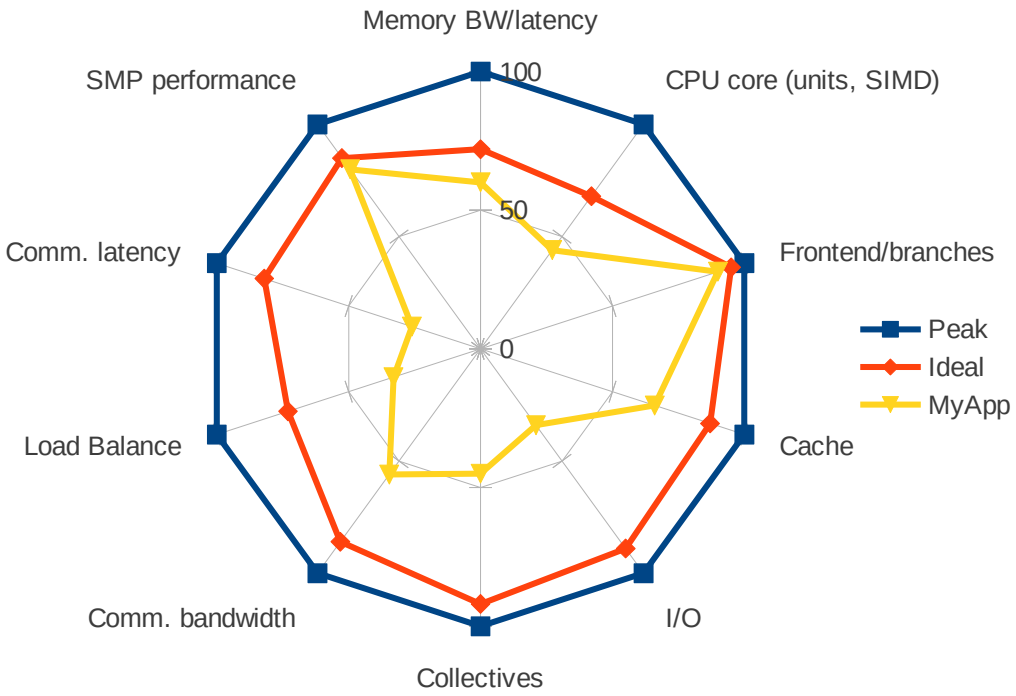
Benchmark the system

Main metrics

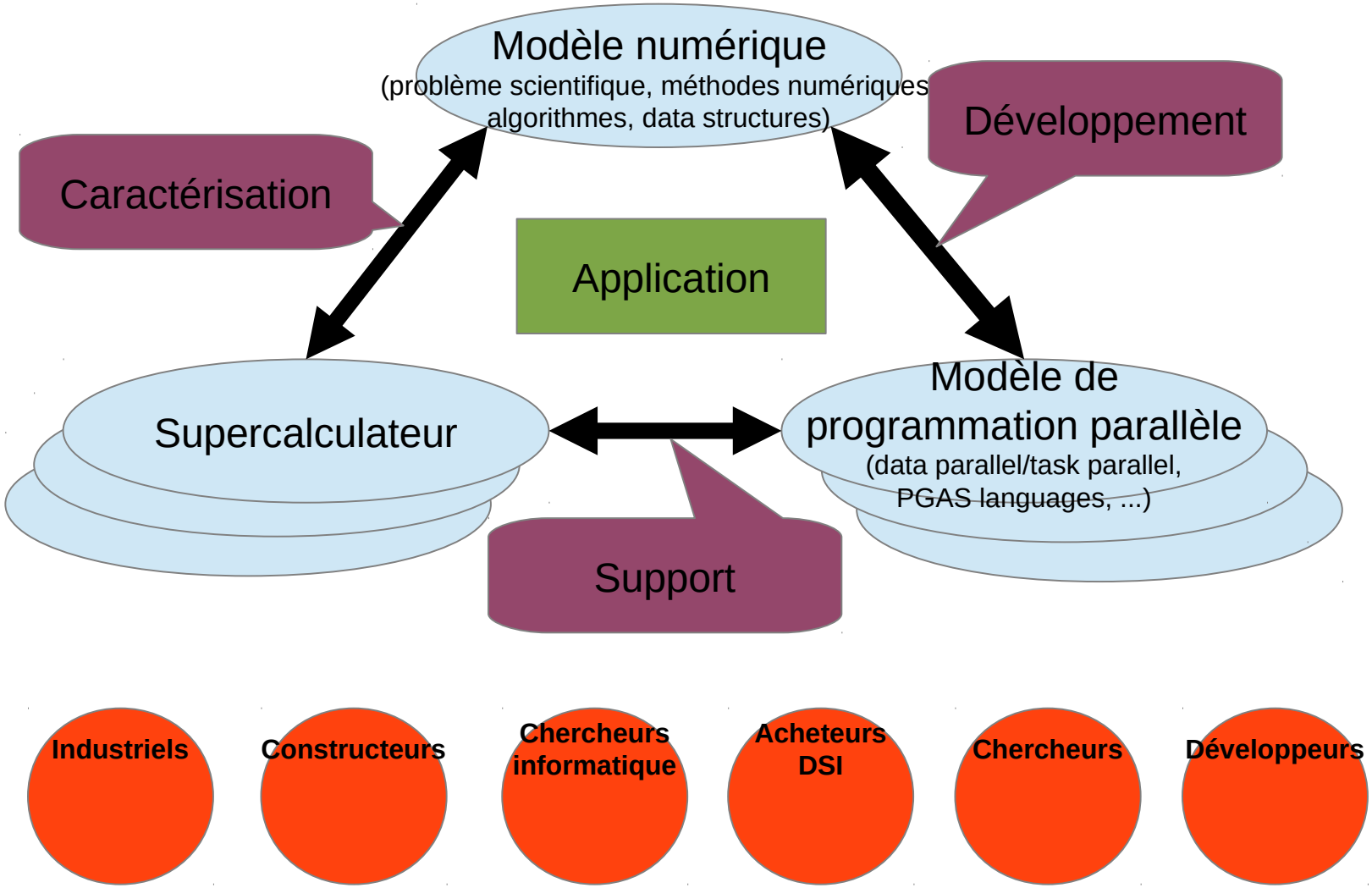


Benchmark the application

MyApp vs System



Schéma



Extrait du dernier livre blanc sur le calcul intensif au CNRS

7. La préparation des codes face aux évolutions actuelles (passage à l'échelle, programmation, hétérogénéité des architectures avec les processeurs multi-cœurs, apparition récente d'accélérateurs de type GPU) est devenue cruciale. Constituer des communautés autour de ces codes pour assurer leur pérennité est un élément important. Bien évaluer les coûts de conception/développement des applications reste essentiel. La collaboration entre spécialistes des applications, informaticiens, mathématiciens et numériciens est indispensable, le « co-design » étant l'un des moyens d'exploiter au mieux les architectures novatrices.

Questions ?