www.bsc.es

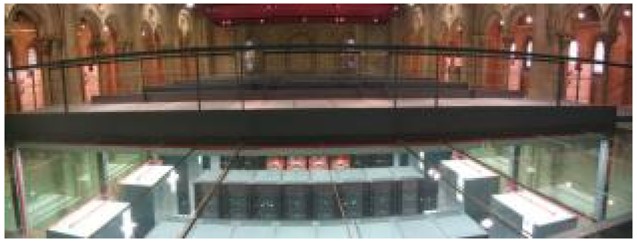
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Parallel Scalable Algorithms

Vassil Alexandrov, ICREA - Barcelona Supercomputing Center, Spain

BSC-CNS

- Barcelona Supercomputing Center – Centro Nacional de Supercomputación (BSC-CNS) is the Spanish National Supercomputing Center.



- The BSC mission:
  – To investigate, develop and manage technology to facilitate the advancement of science.

- The BSC objectives:
  – To perform R&D in Computer Sciences and e-Sciences
  – To provide Supercomputing support to external research.

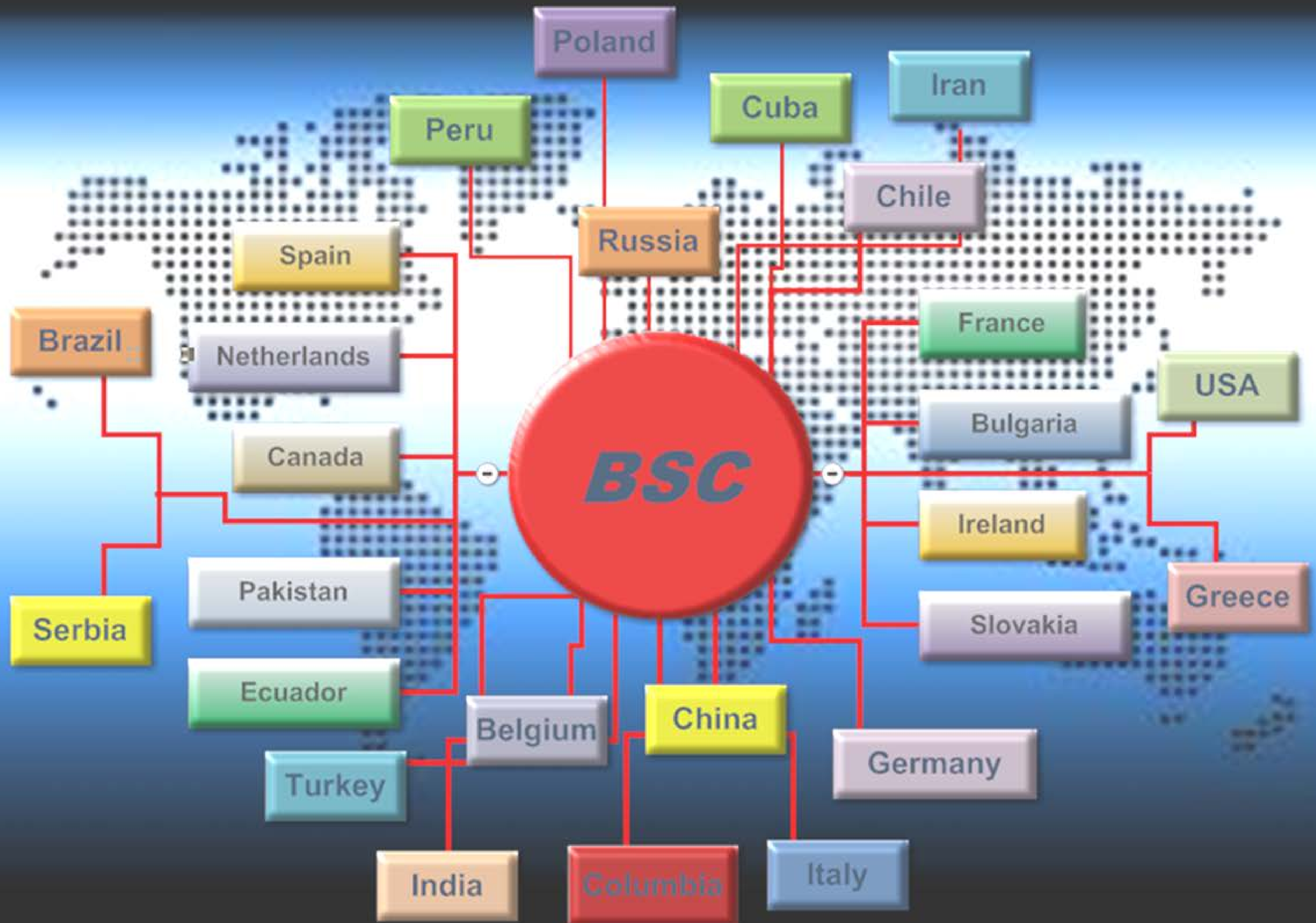
- BSC is a consortium that includes:
  – the Spanish Government – 51%
  – the Catalan Government – 37%
  – the Technical University of Catalonia – 12%

GOBIERNO DE ESPAÑA MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

Generalitat de Catalunya **Departament d'Economia i Coneixement**

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Barcelona Supercomputing Center
Centro Nacional de Supercomputación


Poland
Peru
Cuba
Iran
Chile
Russia
Spain
Brazil
Netherlands
France
USA
Bulgaria
Canada
BSC
Ireland
Greece
Pakistan
Serbia
Slovakia
Ecuador
Belgium
China
Germany
Turkey
India
Columbia
Italy

BSC profile in Education and Training

- Unique role as HPC provider and R&D Center
- Leading Expertise in Computer, Life, Earth & Physical Sciences
- Internally developed technologies
- International prestige
- Severo Ochoa recognition
- Link to large Spanish industries
- Multicultural and multidisciplinary young and motivated team
- Training skills
- Location

Professional Training

- Awarded Advanced Training Centre by PRACE
  - 12 events per academic year
  - Core, Specialised, Scientific Community specific and Industry focused courses
- BSC leads the Spanish SC Network training through RES
  - Workshops, tutorials and lectures
- Severo Ochoa Research Seminar Lectures
  - Monthly event
  - BSC researchers and invited speakers
  - Topics covering the research from all 4 departments
- Severo Ochoa Doctoral Symposium

Focus on the Existing Skills Gap Relevant to HPC

- Computational Scientists (Scientists with HPC capabilities and multidisciplinary skills)
- Programmers for heterogeneous systems
- Parallel programmers
- Algorithm developers for computational science
- HPC systems administration
- Managers with expertise in Computational Science

PRACE Research Infrastructure

- Establishment of the legal framework
  - PRACE AISBL created with seat in Brussels in Apri
    (Association Internationale Sans But Lucratif)
  - 24 members representing 20 European countries
    - Hosting members: France, Germany, Italy, Spain
  - Inaugurated in Barcelona on June 9, 2010

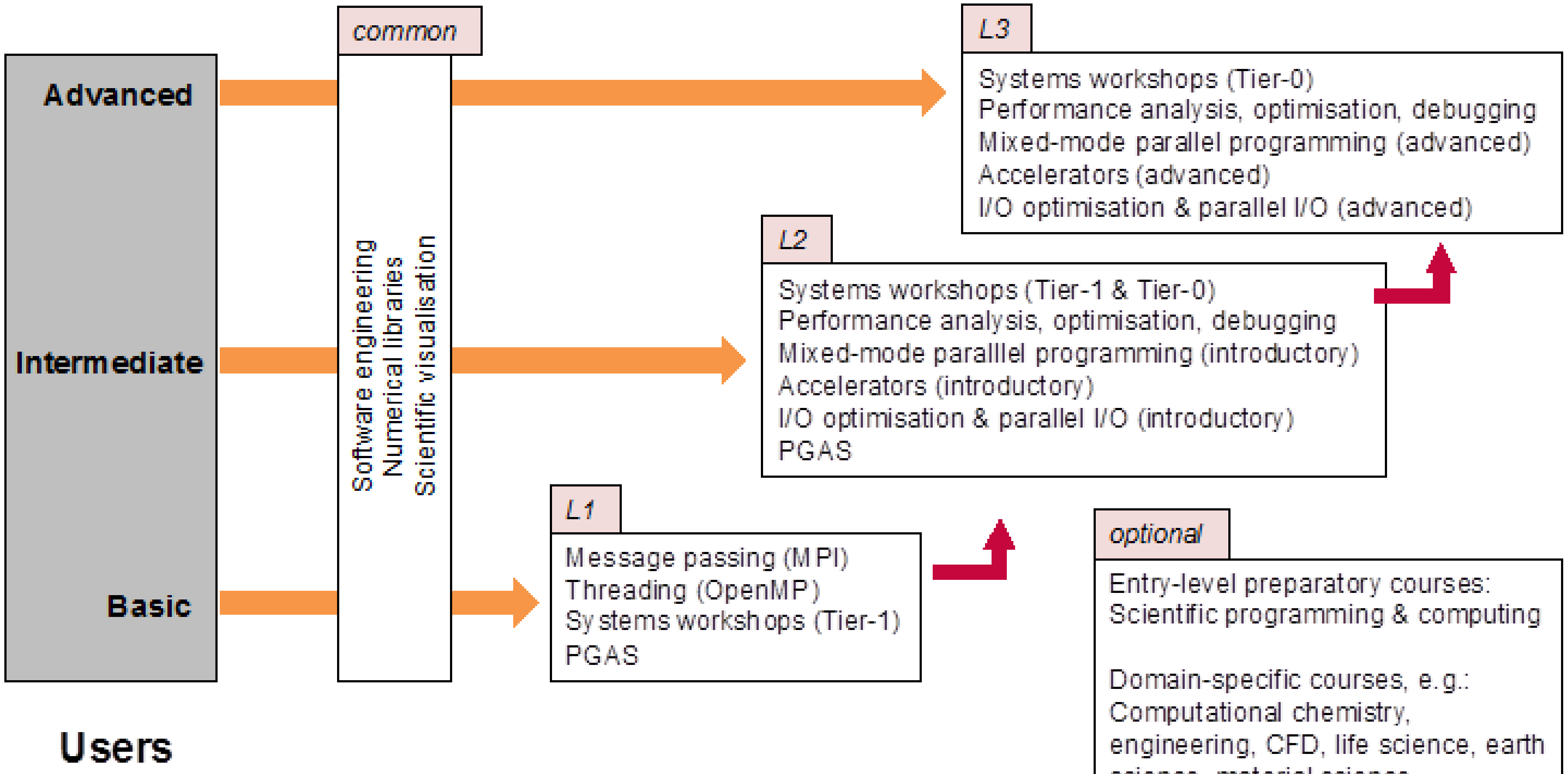
- Funding secured for 2010 - 2015
  - 400 Million € from France (GENCI), Germany (GCS), Italy (CINECA), Spain (BSC)
    Provided as Tier-0 services on TCO basis
  - 70+ Million € from EC FP7 for preparatory and implementat
    Grants INFSO-RI-211528 and 261557
    Complemented by ~ 60 Million € from PRACE members

PRACE model of professional training (1/2)

- Centres of Excellence in Professional Training:
  - Barcelona Supercomputing Centre (Spain),
  - CINECA - Consorzio Interuniversitario (Italy),
  - CSC - IT Centre for Science Ltd (Finland),
  - EPCC at the University of Edinburgh (UK),
  - Gauss Centre for Supercomputing (Germany)
  - Maison de la Simulation (France)

PRACE model of professional training (2/2)

Common Curricula Paths with Localized Syllabus


Advanced
Intermediate
Basic
Users
common
Software engineering
Numerical libraries
Scientific visualisation
L3
Systems workshops (Tier-0)
Performance analysis, optimisation, debugging
Mixed-mode parallel programming (advanced)
Accelerators (advanced)
I/O optimisation & parallel I/O (advanced)
L2
Systems workshops (Tier-1 & Tier-0)
Performance analysis, optimisation, debugging
Mixed-mode paralllel programming (introductory)
Accelerators (introductory)
I/O optimisation & parallel I/O (introductory)
PGAS
L1
Message passing (MPI)
Threading (OpenMP)
Systems workshops (Tier-1)
PGAS
optional
Entry-level preparatory courses:
Scientific programming & computing
Domain-specific courses, e.g.:
Computational chemistry, engineering, CFD, life science, earth science, material science

Professional Training Courses at BSC (2013-14)

Code	Course Title	Level / days	Dates
BSC10	Parallel Programming Workshop	L1 -1; L2 - 4	14 -18.10. 2013
BSC09	Introduction to simulation environment for Earth Sciences	C/C- 2	12 -13.12. 2013
BSC11i	System Administration on a Petaflop System, MareNostrum III	L3 - 2	27, 28.01.2014
BSC13i	13th VI - HPC Tuning Workshop	L2/3	Feb 2014
BSC14	Programming Distributed Computing Platforms with COMPSs	L2/3	Feb 2014
BSC07	Engineering simulation tools: ALYA, FALL3D & PANDORA	C/C - 3	05 -07.02. 2014
BSC08	Simulation environment for Life Sciences	C/C - 2	13 -14.03. 2014
BSC06	Systems Workshop: Programming MareNostrum III	L2 - 2	10 -11.04. 2014
BSC01	Performance Analysis and Tools	L2 -1; L3 - 1	12 -13.05. 2014
BSC02	Heterogeneous Programming on GPUs with MPI + OmpSs	L2 -1; L3 - 1	14 -15.05. 2014
BSC03	Programming ARM based prototypes	L3 - 1	16.05. 2014
BSC04	Introduction to CUDA Programming (with CCOE)	L2 - 5	02 -06.05. 2014
BSC12i	Alya System as a Computational Mechanics Environment	C/C - 2	June 2014
BSC05	PUMPS Summer School (with CCOE)	L2 -1; L3 - 4	July 2014

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

THE COURSE PROGRAMME

Lecture Plan:

Day 1

Session 1 / 10:00 am – 1:00 pm

1. Introduction to parallel architectures, parallel algorithms design and performance metrics.
2. Introduction to the MPI programming model
3. Practical: How to compile and run MPI applications

Session 2 / 2:00pm – 5:00 pm

1. Introduction to Paraver tool: tool to analyze and understand performance
2. Practical: Trace generation and trace analysis

Lecture Plan:

Day 2

Session 1 / 10:00 am - 1:00 pm

1. Tareador: understanding and predicting the potential of task decomposition strategies
2. MPI: Point-to-point communication, collective communication
3. Practical: Simple matrix computations

Session 2 / 2:00 pm - 5:00 pm

1. MPI: Blocking and non-blocking communications
2. MPI: Communicators, Topologies
3. Practical: Heat equation example

Lecture Plan:

Day 3

Session 1 / 10:00 am - 1:00 pm

1. Dimemas: Scalability simulation for MPI applications
2. Practical: Scalability simulations using Dimemas

Session 2 / 2:00 pm - 5:00 pm

1. xSim: Online scalability simulations for MPI applications
2. Practical: Scalability simulations using xSim
3. Additional MPI features: Error handling, parallel libraries, I/O and fault tolerance

Lecture Plan:

Day 4

Session 1 / 10:00am – 1:00 pm

1. Shared-memory programming models, OpenMP fundamentals
2. Parallel regions and work sharing constructs
3. Synchronization mechanisms in OpenMP
4. Practical: heat diffusion in OpenMP

Session 2 / 2:00pm – 5:00 pm

1. Programming using a hybrid MPI/OpenMP approach
2. Practical: heat diffusion in hybrid MPI/OpenMP

Lecture Plan:

Day 5

Session 1 / 10:00 am – 1:00 pm

1. Tasking in OpenMP 3.0 and 4.0
2. Introduction to the OmpSs programming model
3. Practical: heat equation example and divide-and-conquer

Session 2 / 2:00pm – 5:00 pm

1. Programming using a hybrid MPI/OmpSs approach
2. Practical: heat equation example and divide-and-conquer

END of COURSE

1st Lecture Outline:

- Introduction
- Computer Architectures Overview
- Parallel Algorithms and Parallelisation Techniques
- Performance Evaluation and Performance Metrics

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

INTRODUCTION

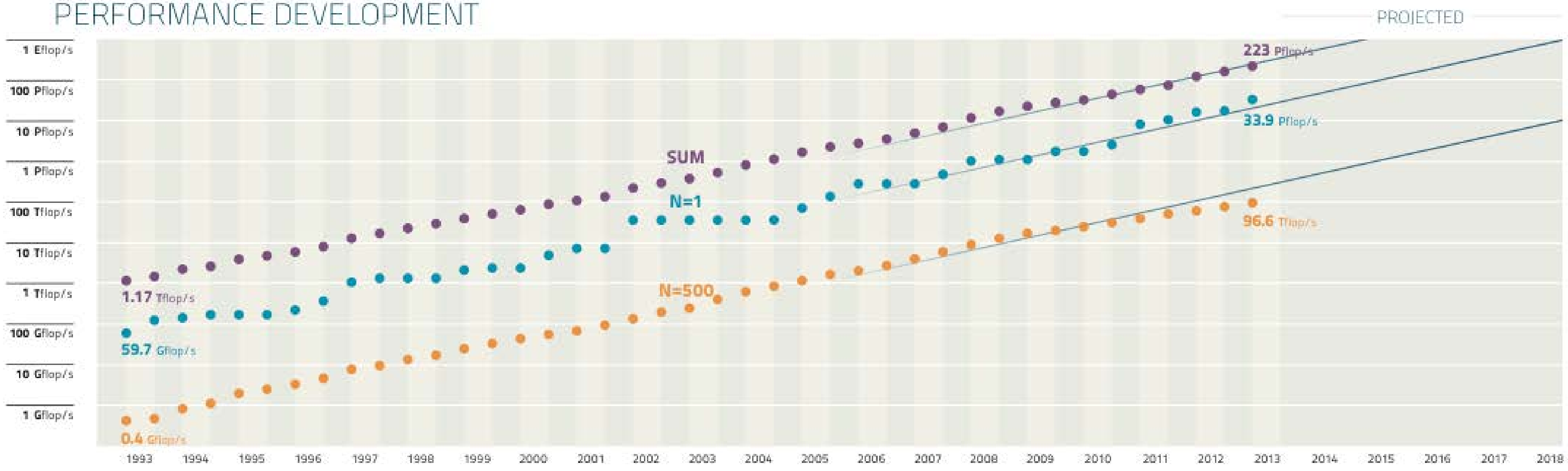
TOP500® JUNE 2013

PRESENTED BY UNIVERSITY OF MANNHEIM · ICL INNOVATIVE COMPUTING LABORATORY THE UNIVERSITY of TENNESSEE · BERKELEY LAB Lawrence Berkeley National Laboratory

FIND OUT MORE AT www.top500.org

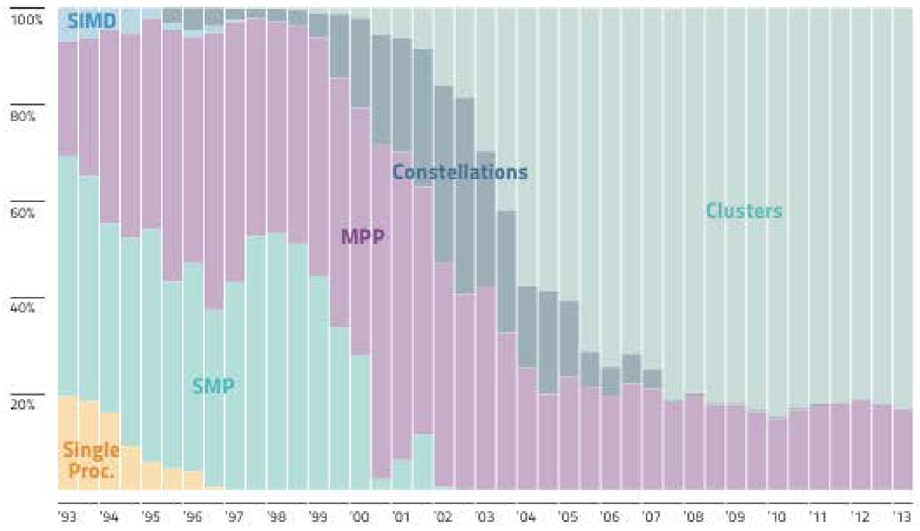
	NAME	SPECS	SITE	COUNTRY	CORES	R_{MAX} PFLOP/S	POWER MW
1	**Tianhe-2 (Milkyway-2)**	NUDT, Intel Ivy Bridge (12C, 2.2 GHz) & Xeon Phi (57C, 1.1 GHz), Custom interconnect	NUDT	China	3,120,000	**33.9**	17.8
2	**Titan**	Cray XK7, Opteron 6274 (16C, 2.2 GHz) + Nvidia Kepler (14C, .732 GHz), Custom interconnect	DOE/SC/ORNL	USA	560,640	**17.6**	8.3
3	**Sequoia**	IBM BlueGene/Q, Power BQC (16C, 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	**17.2**	7.9
4	**K computer**	Fujitsu SPARC64 VIIIfx (8C, 2.0GHz), Custom interconnect	RIKEN AICS	Japan	705,024	**10.5**	12.7
5	**Mira**	IBM BlueGene/Q, Power BQC (16C, 1.60 GHz), Custom interconnect	DOE/SC/ANL	USA	786,432	**8.16**	3.95

PERFORMANCE DEVELOPMENT

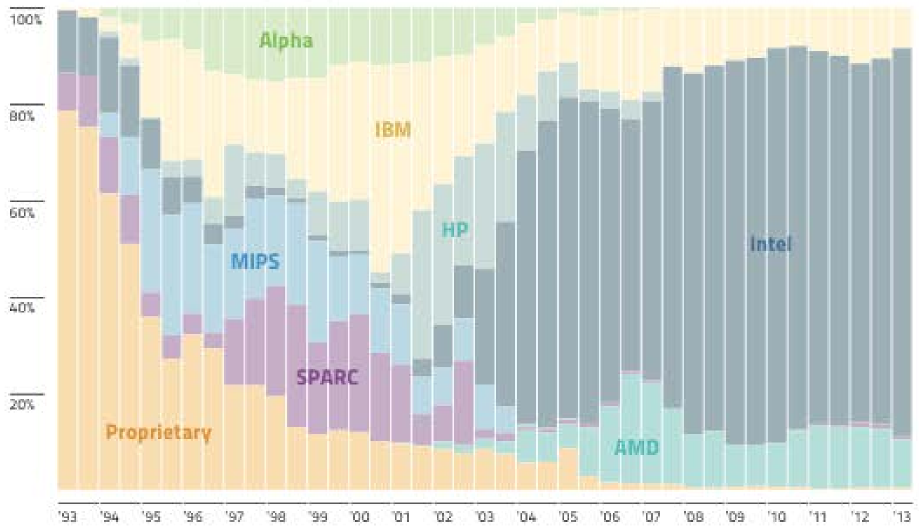


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

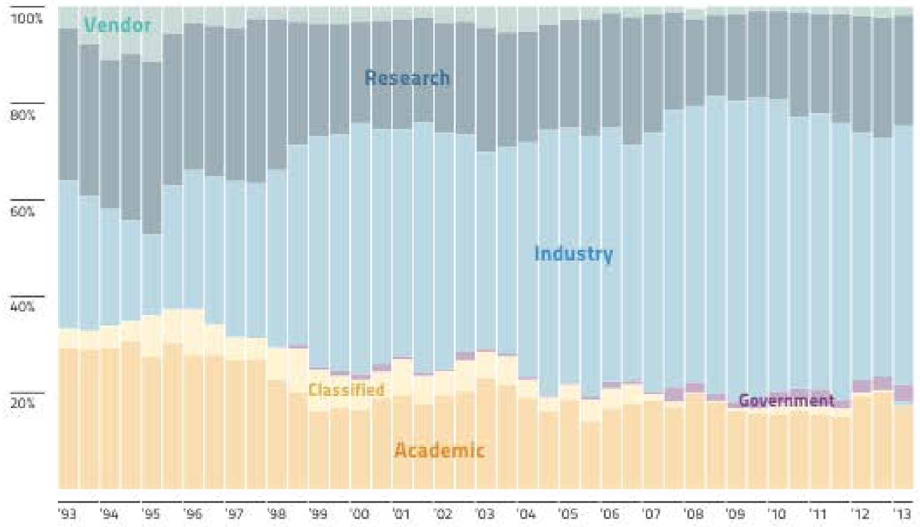
ARCHITECTURES



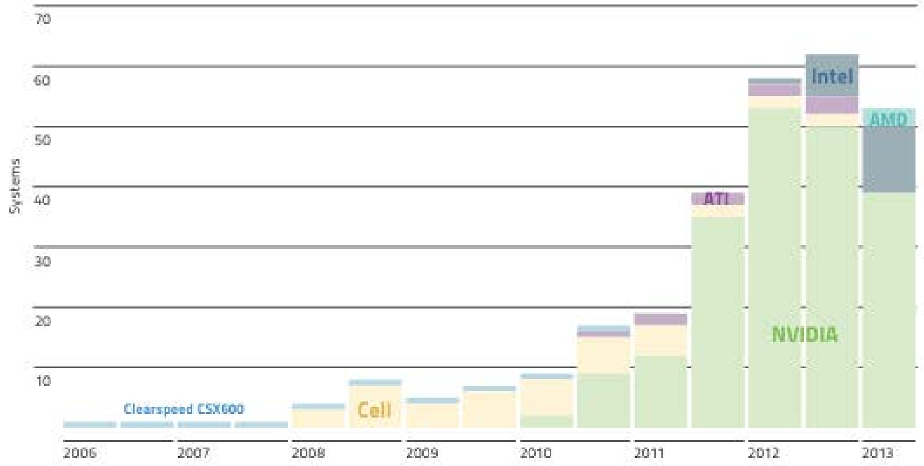
CHIP TECHNOLOGY



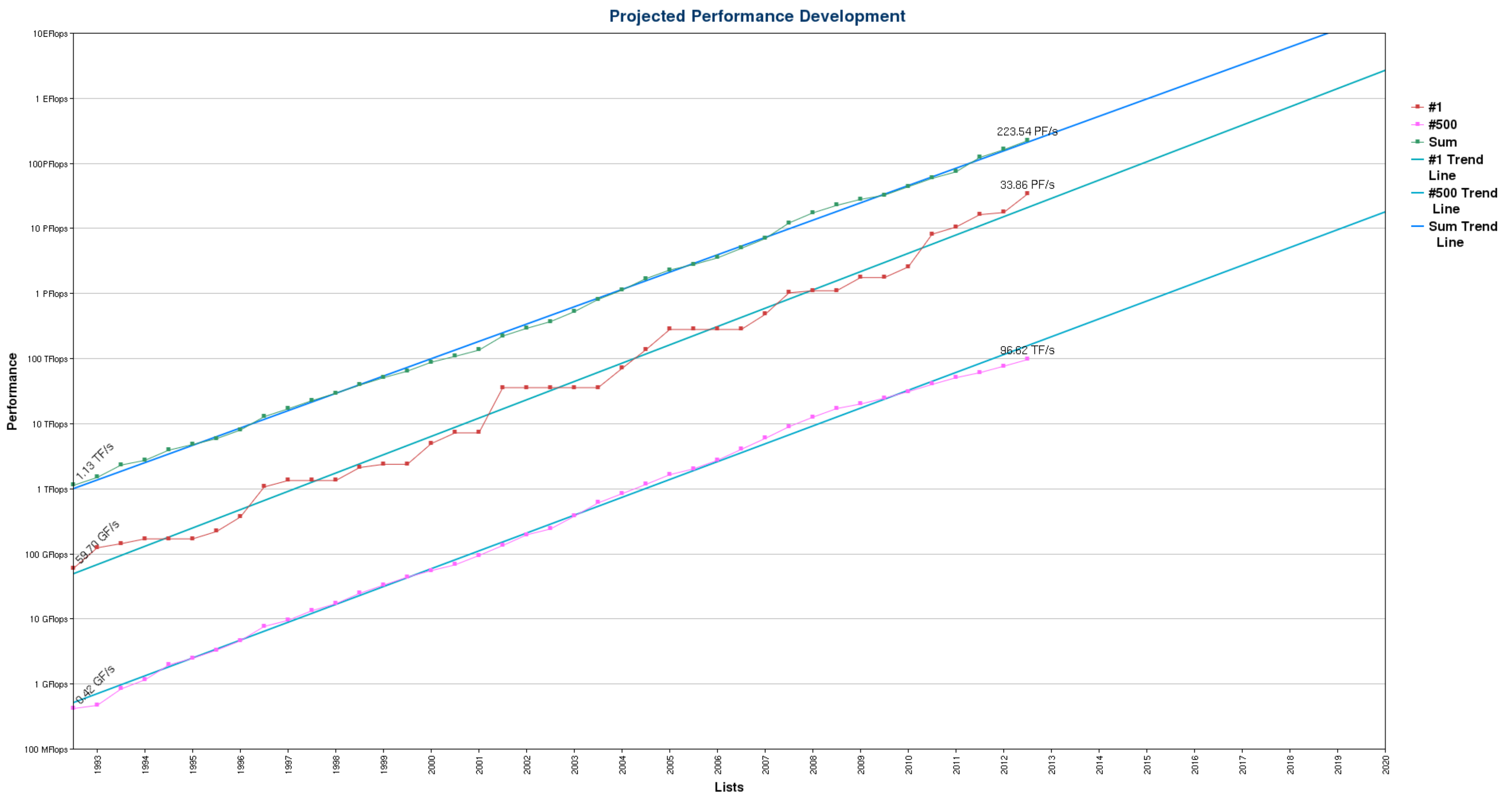
INSTALLATION TYPE



ACCELERATORS / CO-PROCESSORS

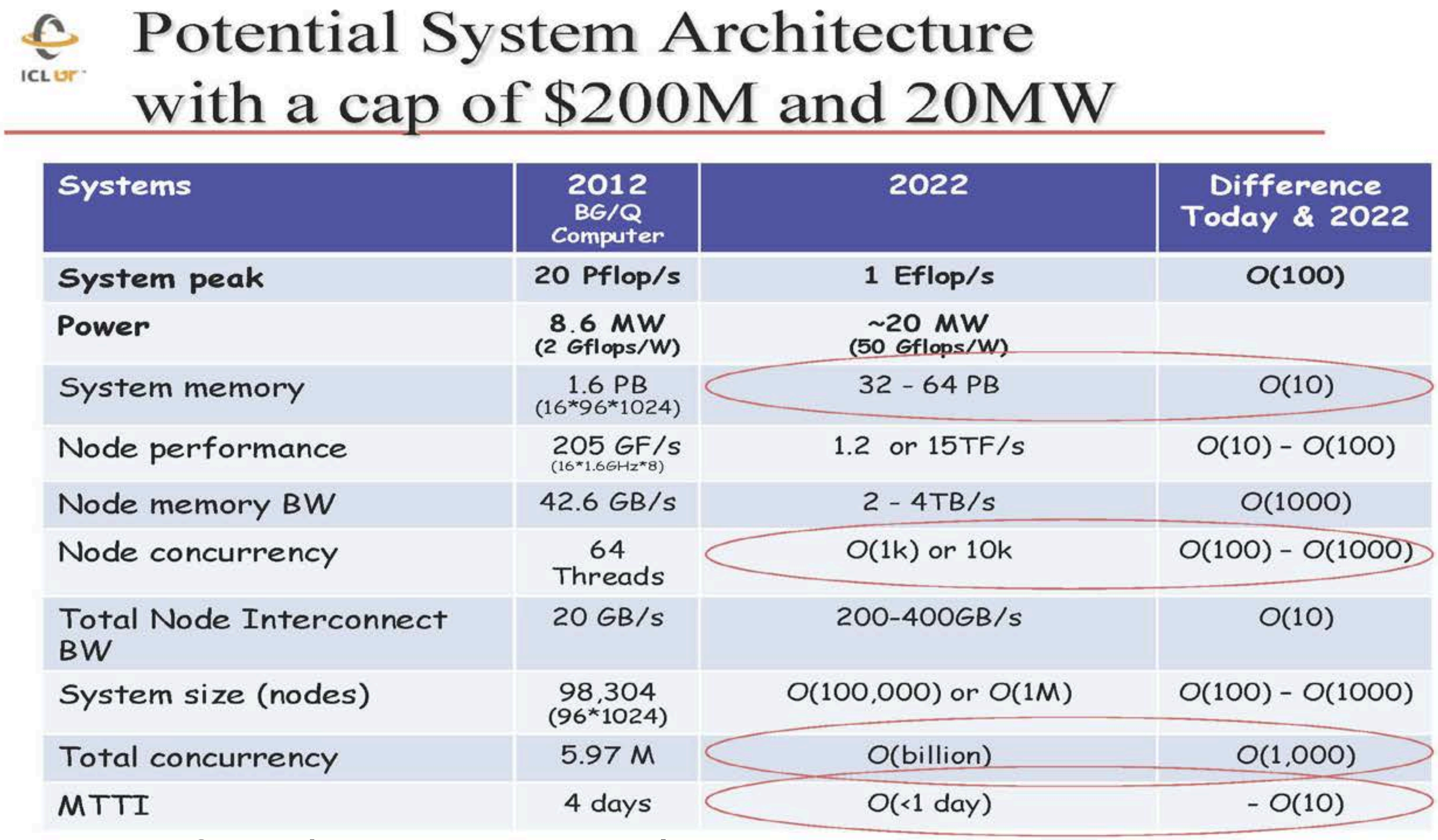


Projected Performance Development



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Road to Exascale



Potential System Architecture with a cap of $200M and 20MW

Systems	2012 BG/Q Computer	2022	Difference Today & 2022
System peak	20 Pflop/s	1 Eflop/s	O(100)
Power	8.6 MW (2 Gflops/W)	~20 MW (50 Gflops/W)	
System memory	1.6 PB (16*96*1024)	32 - 64 PB	O(10)
Node performance	205 GF/s (16*1.6GHz*8)	1.2 or 15TF/s	O(10) – O(100)
Node memory BW	42.6 GB/s	2 - 4TB/s	O(1000)
Node concurrency	64 Threads	O(1k) or 10k	O(100) – O(1000)
Total Node Interconnect BW	20 GB/s	200-400GB/s	O(10)
System size (nodes)	98,304 (96*1024)	O(100,000) or O(1M)	O(100) – O(1000)
Total concurrency	5.97 M	O(billion)	O(1,000)
MTTI	4 days	O(<1 day)	- O(10)

Prof. Jack Dongarra, ScalA12, SLC, USA

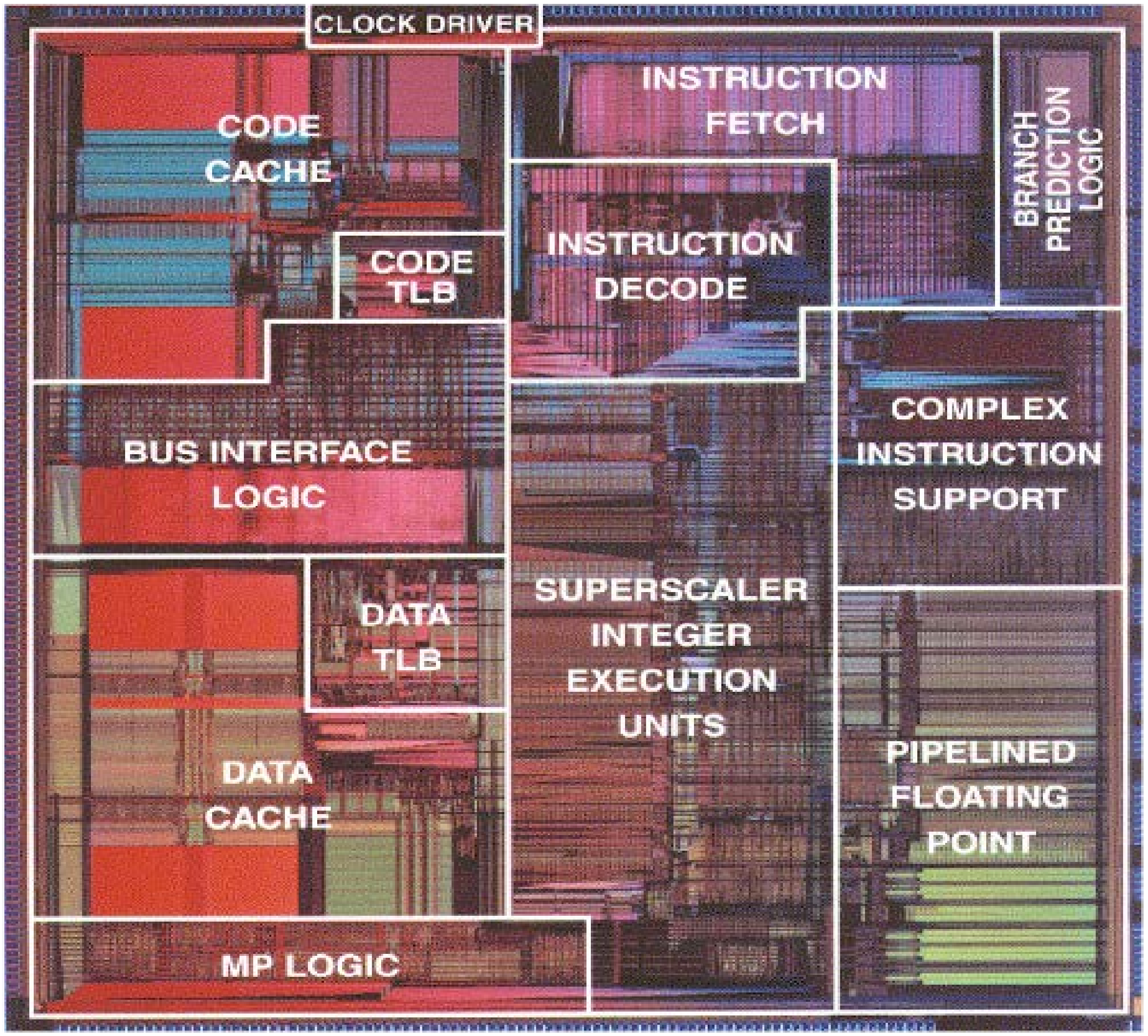
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

COMPUTER ARCHITECTURES

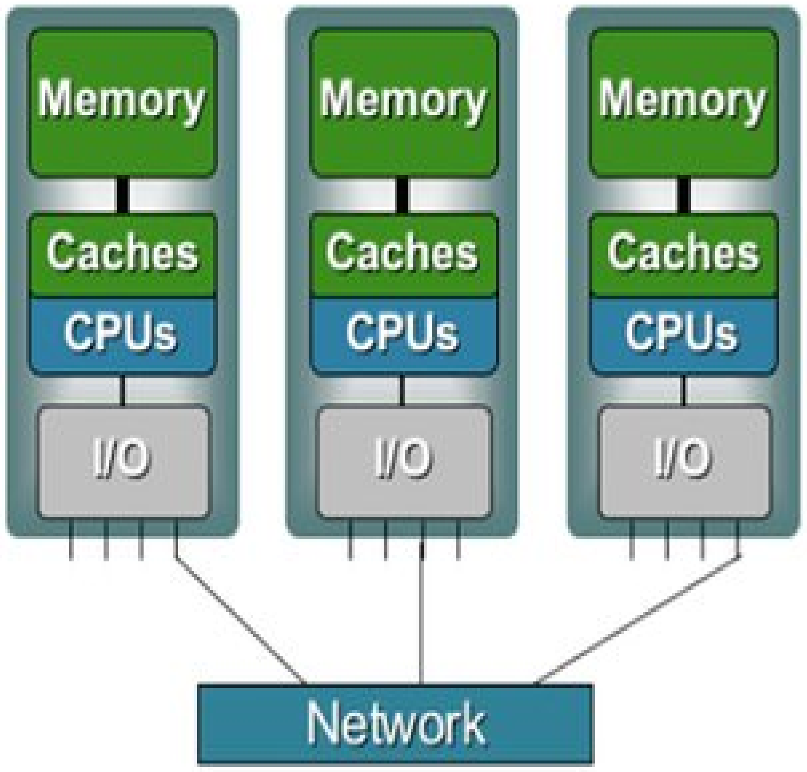
Flynn's Taxonomy

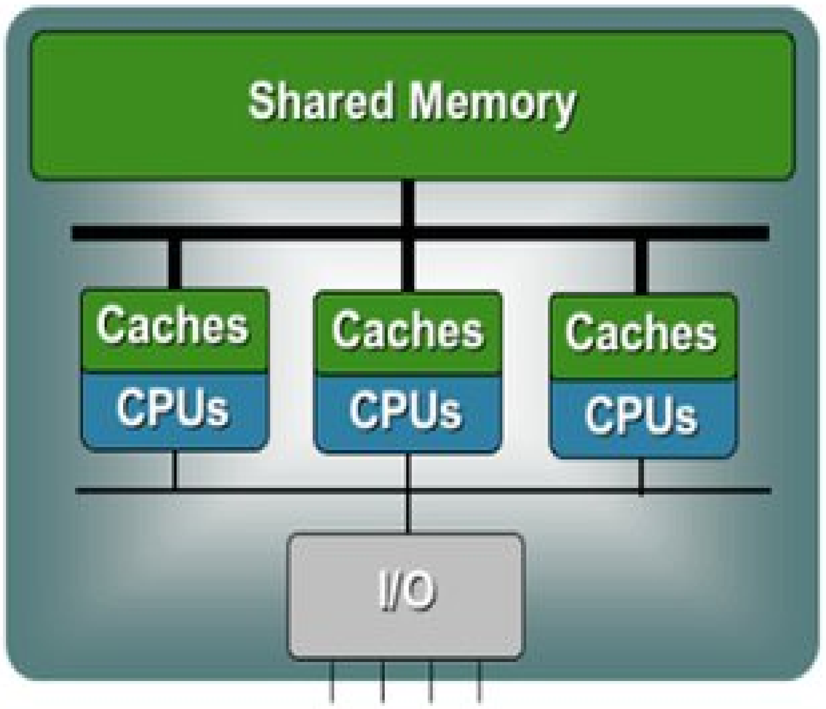
- SISD – Single Instruction/ Single Data Stream
- SIMD – Single Instruction/Multiple Data Stream
- MISD - Multiple Instruction/Single Data Stream
- MIMD - Multiple Instruction/Multiple Data Stream

SISD - Traditional x86



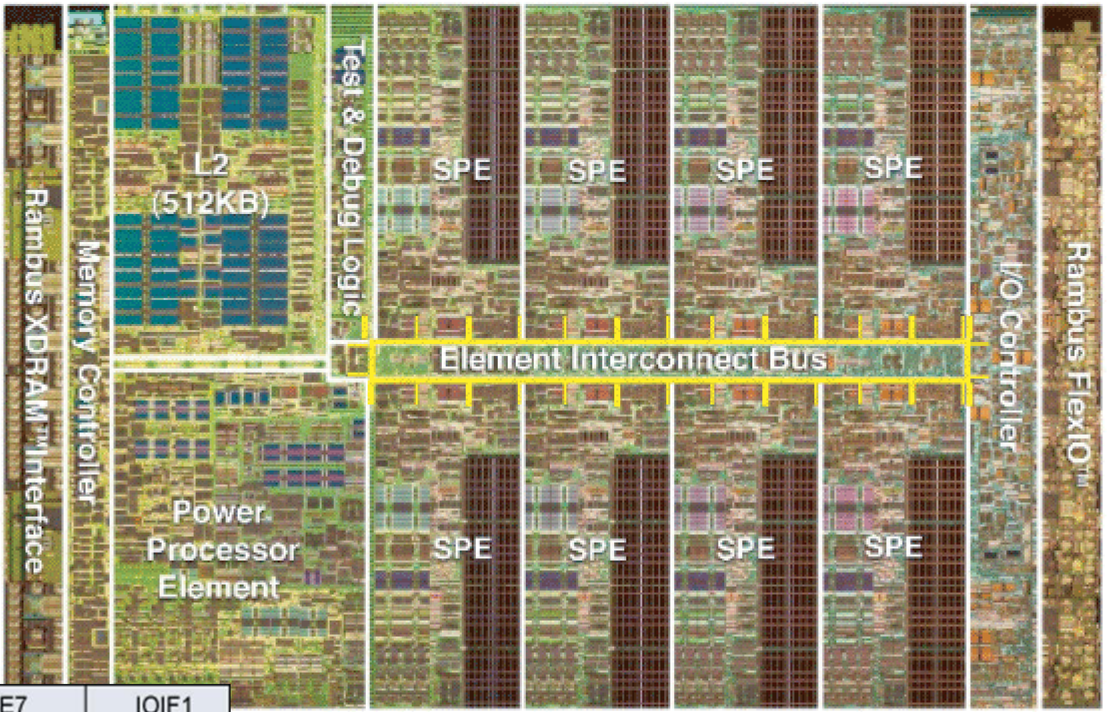
MIMD - Distributed Memory vs. Shared Memory

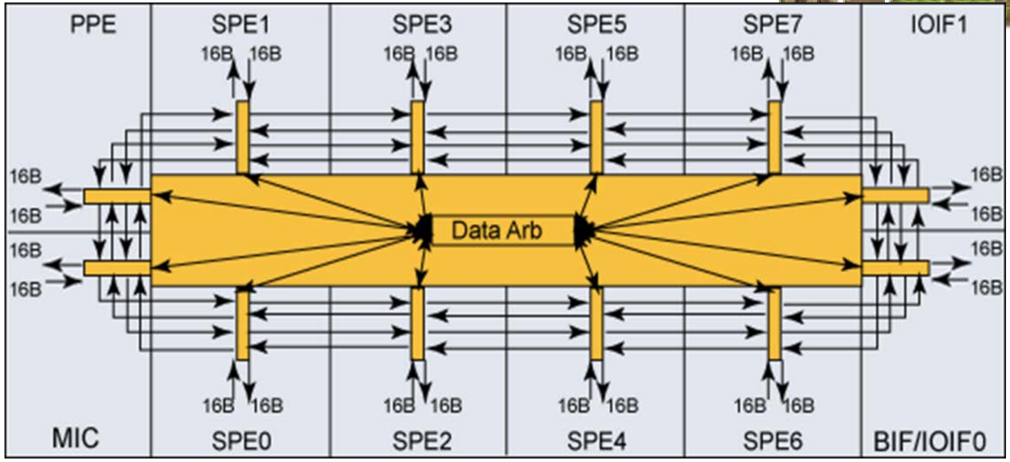




Accelerator type architectures: IBM Cell architechture

- 8 separate computational units SPE
- Data needs to be transferred on a special bus between main PowerPC CPU and SPEs





GPU Accelerators

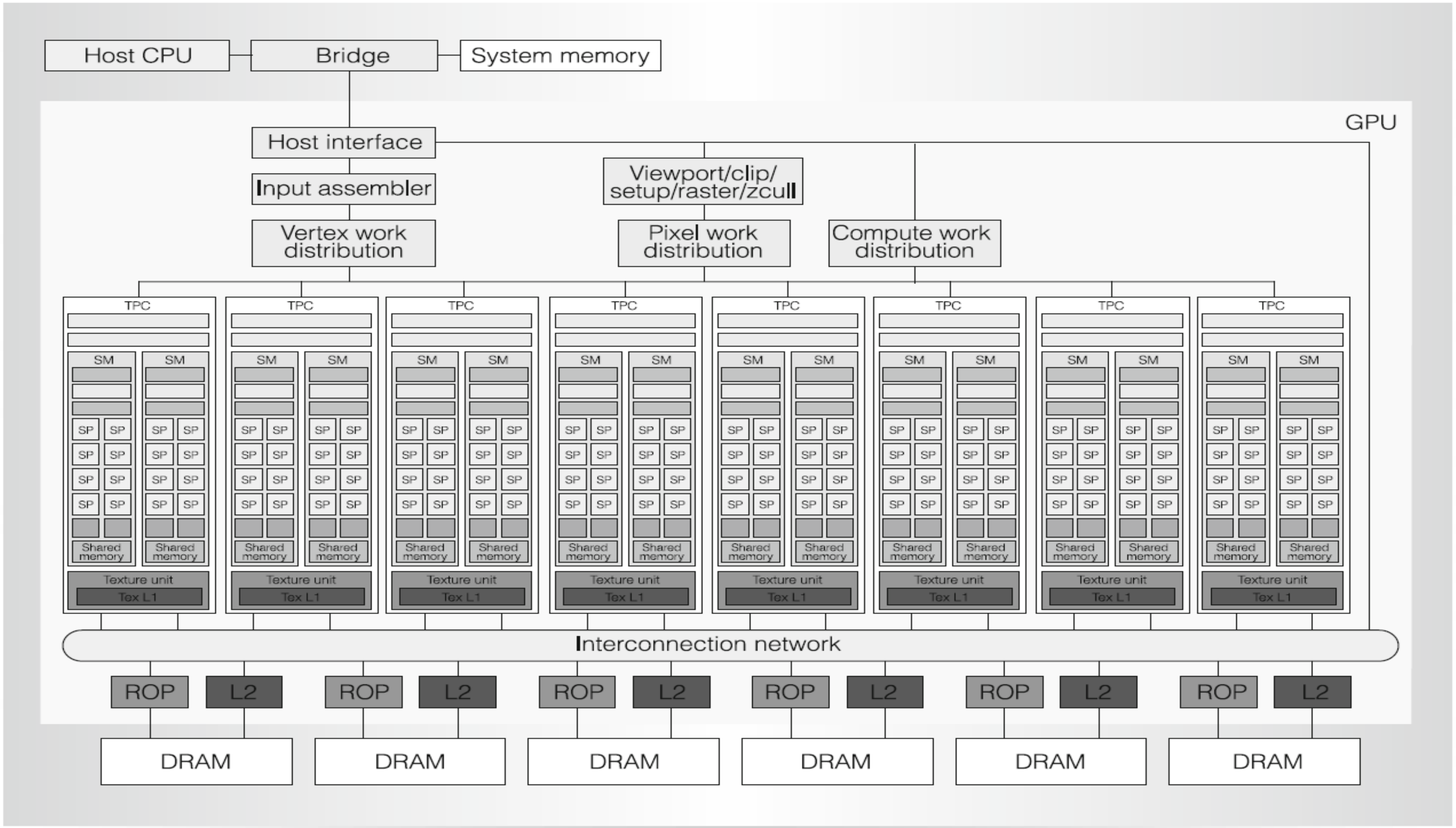
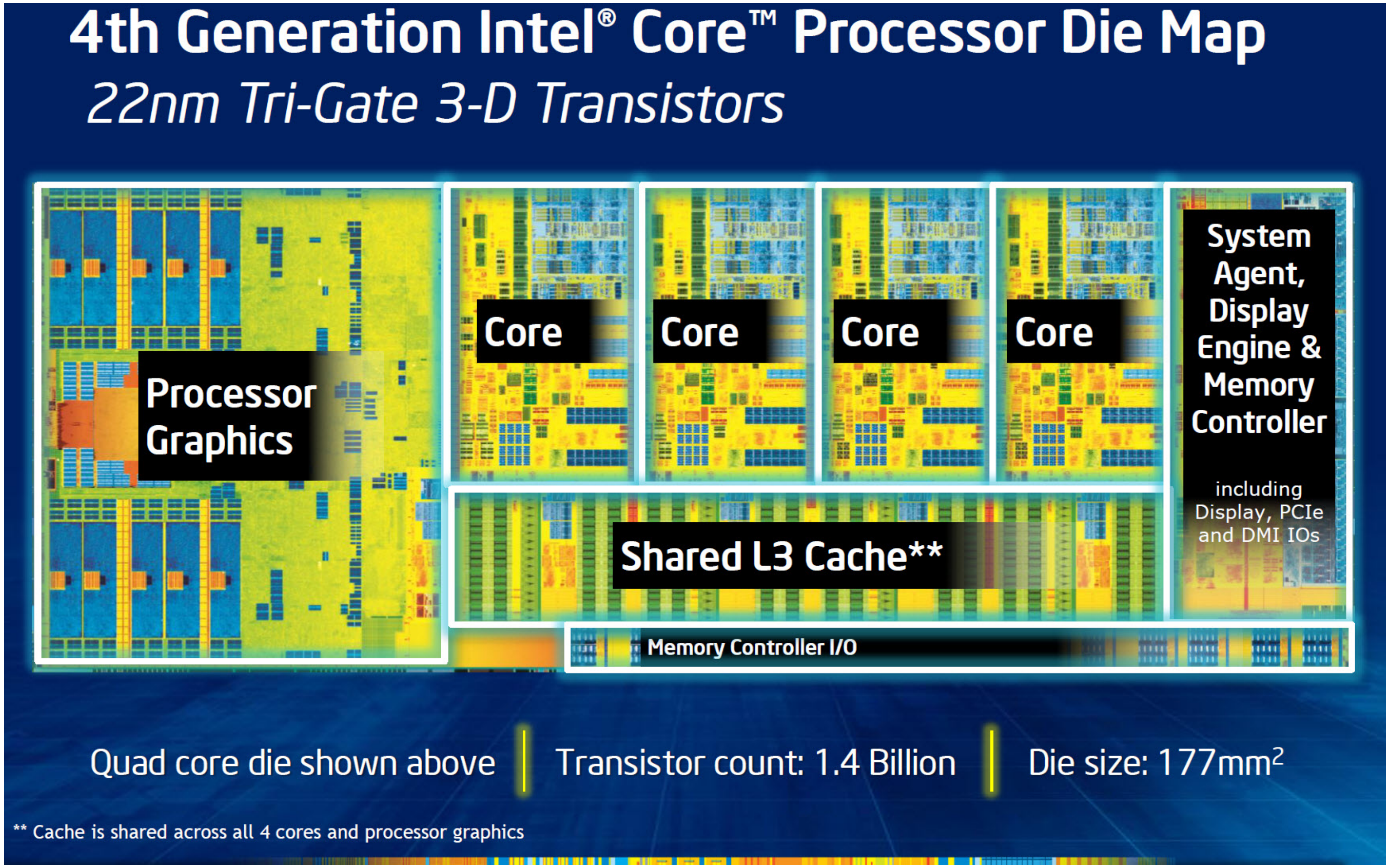
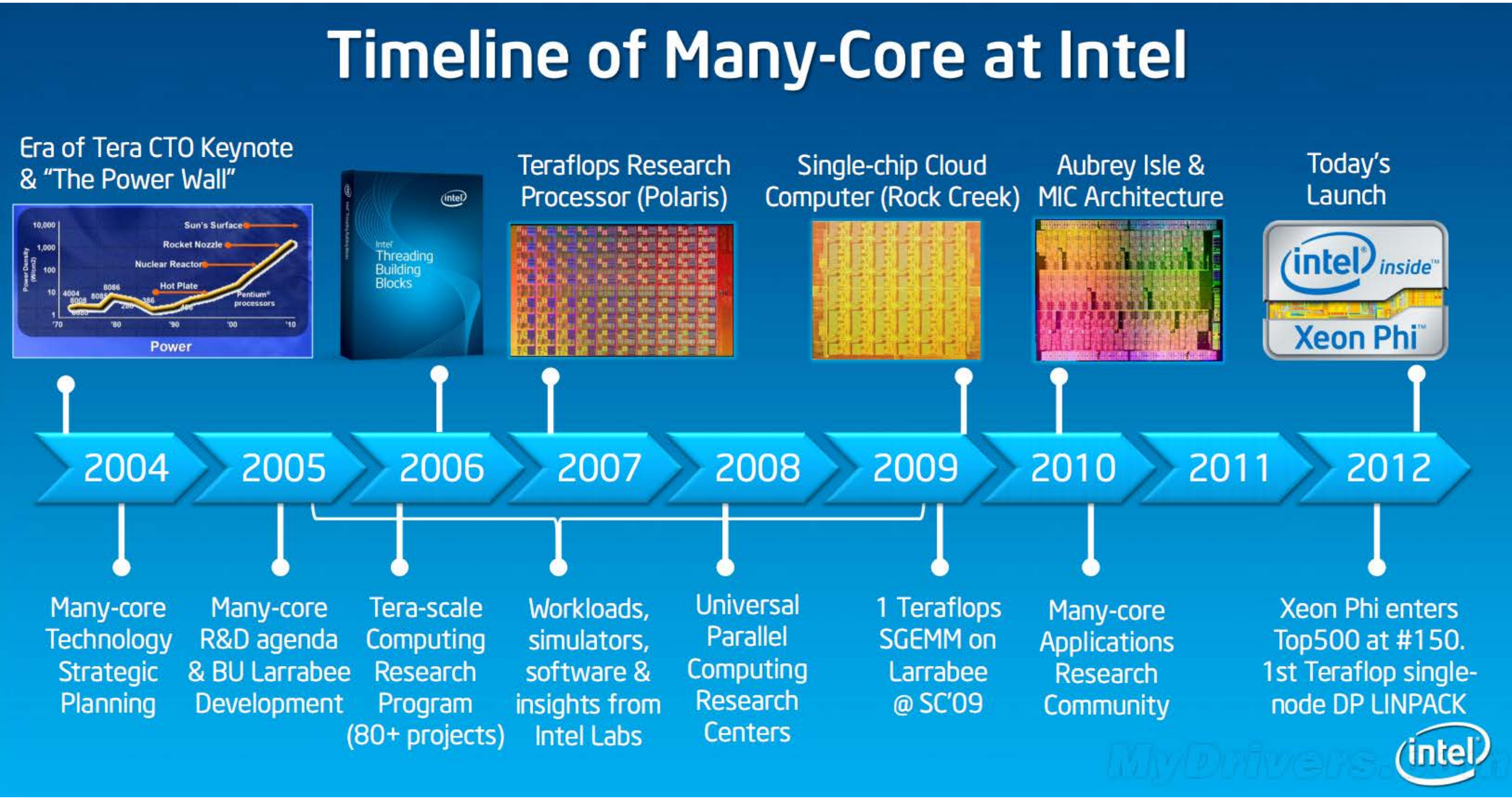


Figure 1. Tesla unified graphics and computing GPU architecture. TPC: texture/processor cluster; SM: streaming multiprocessor; SP: streaming processor; Tex: texture, ROP: raster operation processor.

Modern Multi-Cores


4th Generation Intel® Core™ Processor Die Map
22nm Tri-Gate 3-D Transistors
Processor Graphics
Core
Core
Core
Core
System Agent, Display Engine & Memory Controller
including Display, PCIe and DMI IOs
Shared L3 Cache**
Memory Controller I/O
Quad core die shown above
Transistor count: 1.4 Billion
Die size: 177mm²
** Cache is shared across all 4 cores and processor graphics

Multi- & Many-Core Architectures



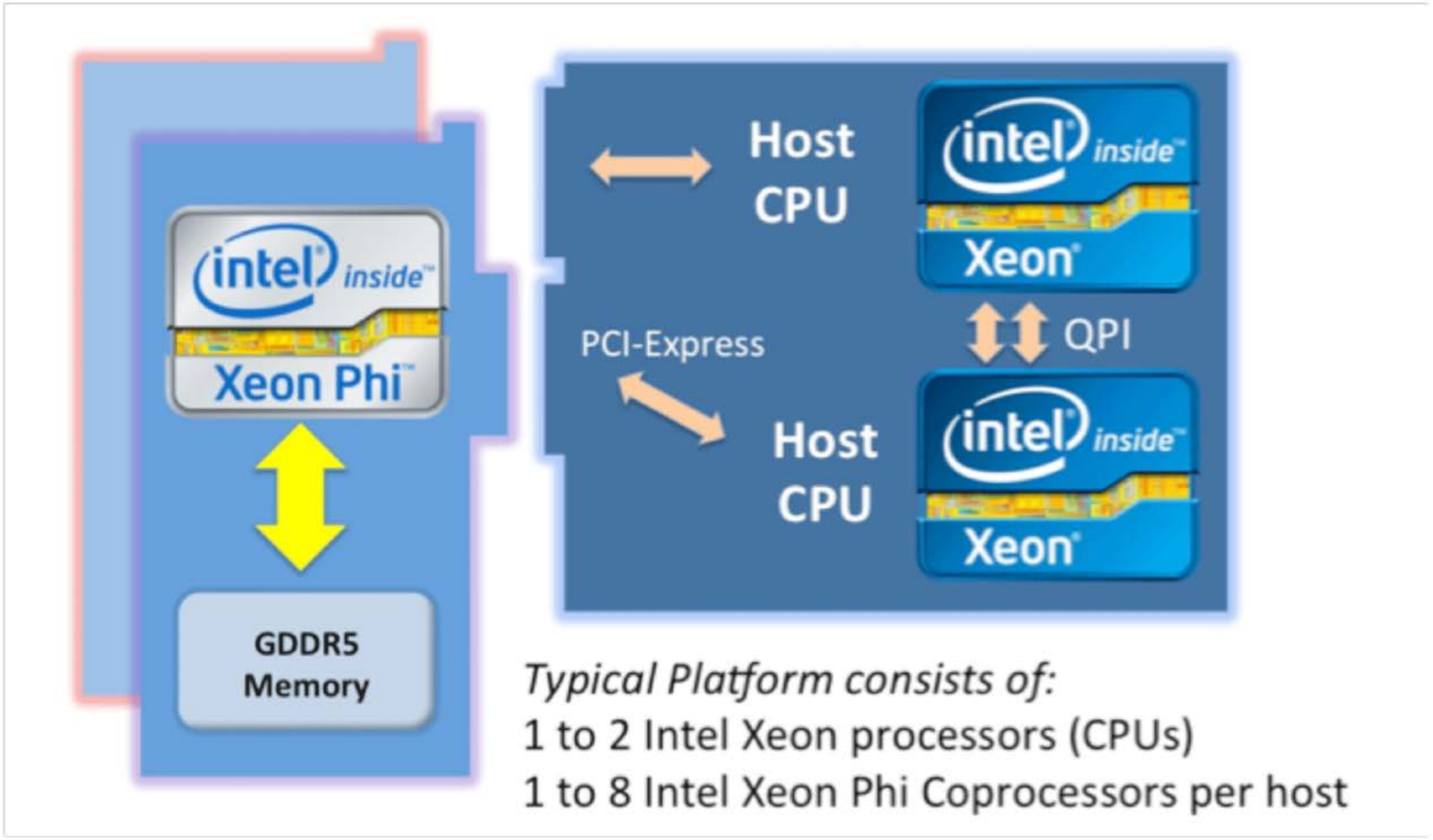
Timeline of Many-Core at Intel

Year	Event
2004	Era of Tera CTO Keynote & "The Power Wall"
2006	Intel Threading Building Blocks
2007	Teraflops Research Processor (Polaris)
2009	Single-chip Cloud Computer (Rock Creek)
2010	Aubrey Isle & MIC Architecture
2012	Today's Launch

2004 – 2005 – 2006 – 2007 – 2008 – 2009 – 2010 – 2011 – 2012

Year	Milestone
2004	Many-core Technology Strategic Planning
2005	Many-core R&D agenda & BU Larrabee Development
2006	Tera-scale Computing Research Program (80+ projects)
2007	Workloads, simulators, software & insights from Intel Labs
2008	Universal Parallel Computing Research Centers
2009	1 Teraflops SGEMM on Larrabee @ SC'09
2010	Many-core Applications Research Community
2012	Xeon Phi enters Top500 at #150. 1st Teraflop single-node DP LINPACK

Adding External Accelerators



Typical Platform consists of:
1 to 2 Intel Xeon processors (CPUs)
1 to 8 Intel Xeon Phi Coprocessors per host

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Intel® Xeon Phi™ Coprocessor Family Reference Table

SKU #	Form Factor, Thermal	Peak Double Precision	Max # of Cores	Clock Speed (GHz)	GDDR5 Memory Speeds (GT/s)	Peak Memory BW	Memory Capacity (GB)	Total Cache (MB)	Board TDP (Watts)	Process
SE10P (special edition)	PCIe Card, Passively Cooled	1073 GF	61	1.1	5.5	352	8	30.5	300	22nm
SE10X (special edition)	PCIe Card, No Thermal Solution	1073 GF	61	1.1	5.5	352	8	30.5	300	
5110P	PCIe Card, Passively Cooled	1011 GF	60	1.053	5.0	320	8	30	225	
3100 Series	PCIe Card, Actively Cooled	>1 TF	Disclosed at 3100 series launch (H1'13)		5.0	240	6	28.5	300	
	PCIe Card, Passively Cooled	> 1 TF			5.0	240	6	28.5	300	

PCIe Card, Actively Cooled

PCIe Card, Passively Cooled

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

BSC SUPERCOMPUTERS

MareNostrum III

IBM iDataPlex cluster with 3028 compute nodes

- Peak Performance of 1 Petaflops
- 48,448 Intel SandyBridge-EP E5-2670 cores at 2.6 GHz
- Two 8 core CPUs per node (16 cores/node)
- 94.625 TB of main memory (32 GB/node)
- 1.9 PB of disk storage
- Interconnection networks:
  - Infiniband
  - Gigabit Ethernet
- Operating System: Linux - SuSe Distribution
- Consisting of 36 racks
- Footprint:120m^2

Completed system - 48,448 cores and predicted to be in the top 25

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

MinoTauro

NVIDIA GPU cluster with 128 Bull B505 blades

- 2 Intel E5649 6-Core processors at 2.53 GHz per node; in total 5544 cores
- 2 M2090 NVIDIA GPU Cards
- 24 GB of Main memory
- Peak Performance: 185.78 TFlops
- 250 GB SSD (Solid State Disk) as local storage
- 2 Infiniband QDR (40 Gbit each) to a non-blocking network
- RedHat Linux
- 14 links of 10 GbitEth to connect to BSC GPFS Storage

The Green 500 list November 2012: #36 with 1266 Mflops/Watt, 81.5 kW total Power

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

PARALLEL SCALABLE ALGORITHMS- PARALLELIZATION TECHNIQUES

Scalable Algorithms: Motivation/Drivers

- Bridging the Performance Gap while dealing with Hybrid Architectures
- Increased Scalability
- Highly fault-tolerant and fault-resilient algorithms
- Need to calculate with higher precision without restart
- Need to tackle efficiently Grand Challenges problems

Challenges

To achieve excellent results scalability at all levels would be required:

- Mathematical models level
- Algorithmic level
- Systems level

Parallel Algorithms and Concurrency

- Parallel Algorithms
  - Tasks and Decomposition
  - Processes and Mapping
  - Processes Versus Processors
- Decomposition Techniques
  - Recursive Decomposition
  - Recursive Decomposition
  - Exploratory Decomposition
  - Hybrid Decomposition
- Characteristics of Tasks and Interactions
  - Task Generation, Granularity, and Context
  - Characteristics of Task Interactions.

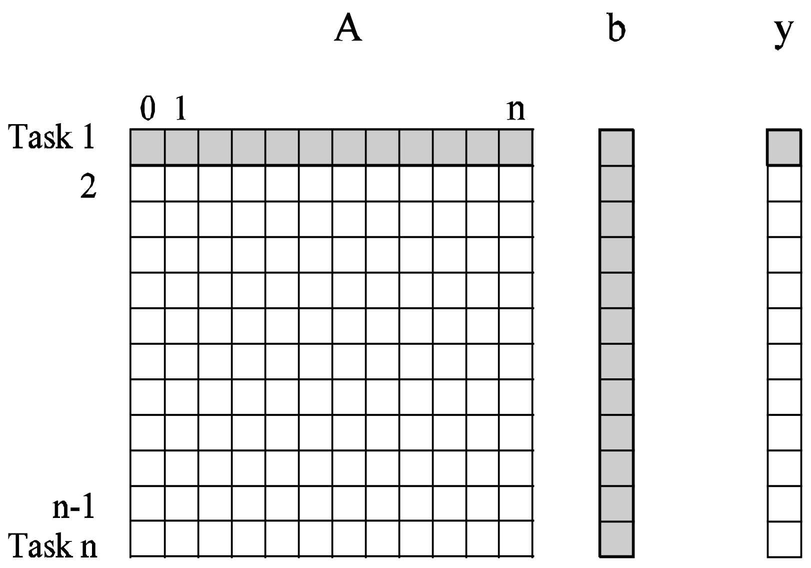
Mapping

- Mapping Techniques for Load Balancing
- Methods for Minimizing Interaction Overheads
- Parallel Algorithm Design Models

Decomposition, Tasks, and Dependency Graphs

- The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently
- A given problem may be decomposed into tasks in many different ways.
- Tasks may be of same, or different sizes.
- A decomposition can be illustrated in the form of a directed graph with nodes corresponding to tasks and edges indicating that the result of one task is required for processing the next. Such a graph is called a task dependency graph.

Multiplying a Dense Matrix with a Vector

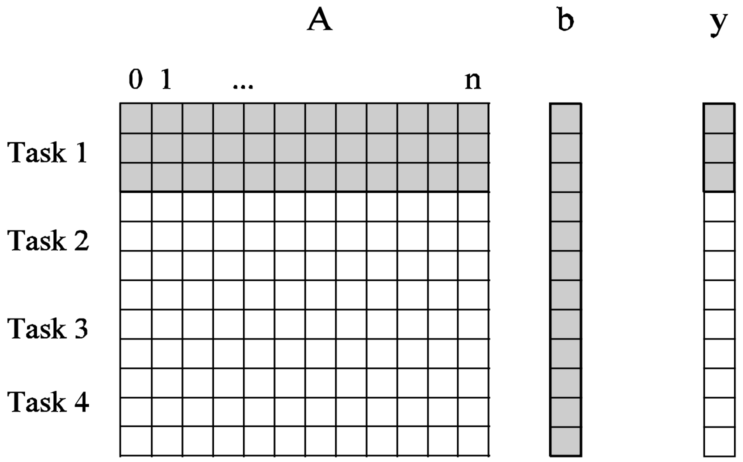


Computation of each element of output vector y is independent of other elements. Based on this, a dense matrix-vector product can be decomposed into n tasks. The figure highlights the portion of the matrix and vector accessed by Task 1.

Observations: While tasks share data (namely, the vector b), they do not have any control dependencies - i.e., no task needs to wait for the (partial) completion of any other. All tasks are of the same size in terms of number of operations. Is this the maximum number of tasks we could decompose this problem into?

Granularity of Task Decompositions

- The number of tasks into which a problem is decomposed determines its granularity.
- Decomposition into a large number of tasks results in fine-grained decomposition and that into a small number of tasks results in a coarse grained decomposition.



A coarse grained version of the dense matrix-vector product example. Each task in this example corresponds to the computation of p=3 elements of the result vector.

Degree of Concurrency

- The number of tasks that can be executed in parallel is the degree of concurrency of a decomposition.
- Since the number of tasks that can be executed in parallel may change over program execution, the maximum degree of concurrency is the maximum number of such tasks at any point during execution. What is the maximum degree of concurrency of the database query examples?
- The average degree of concurrency is the average number of tasks that can be processed in parallel over the execution of the program. Assuming that each tasks in the database example takes identical processing time, what is the average degree of concurrency in each decomposition?
- The degree of concurrency increases as the decomposition becomes finer in granularity and vice versa.

Limits on Parallel Performance

- It would appear that the parallel time can be made arbitrarily small by making the decomposition finer in granularity.
- There is an inherent bound on how fine the granularity of a computation can be. For example, in the case of multiplying a dense matrix with a vector, there can be no more than (n2) concurrent tasks.
- Concurrent tasks may also have to exchange data with other tasks. This results in communication overhead. The tradeoff between the granularity of a decomposition and associated overheads often determines performance bounds.

Processes and Mapping

- In general, the number of tasks in a decomposition exceeds the number of processing elements available.

- For this reason, a parallel algorithm must also provide a mapping of tasks to processes.

Processes and Mapping

- Appropriate mapping of tasks to processes is critical to the parallel performance of an algorithm.
- Mappings are determined by both the task dependency and task interaction graphs.
- Task dependency graphs can be used to ensure that work is equally spread across all processes at any point (minimum idling and optimal load balance).
- Task interaction graphs can be used to make sure that processes need minimum interaction with other processes (minimum communication).

Processes and Mapping

An appropriate mapping must minimize parallel execution time by:

- Mapping independent tasks to different processes.
- Assigning tasks on critical path to processes as soon as they become available.
- Minimizing interaction between processes by mapping tasks with dense interactions to the same process.

Note: These criteria often conflict with each other. For example, a decomposition into one task (or no decomposition at all) minimizes interaction but does not result in a speedup at all!

Decomposition Techniques

So how does one decompose a task into various subtasks?

While there is no single recipe that works for all problems, we present a set of commonly used techniques that apply to broad classes of problems. These include:

- recursive decomposition
- data decomposition
- exploratory decomposition
- hybrid decomposition

Data Decomposition: Example

Consider the problem of multiplying two n x n matrices A and B to yield matrix C. The output matrix C can be partitioned into four tasks as follows:

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Task 1: $C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$

Task 2: $C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$

Task 3: $C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$

Task 4: $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$

Data Decomposition: Example

A partitioning of output data does not result in a unique decomposition into tasks. For example, for the same problem as in previous slide, with identical output data distribution, we can derive the following two (different) decompositions:

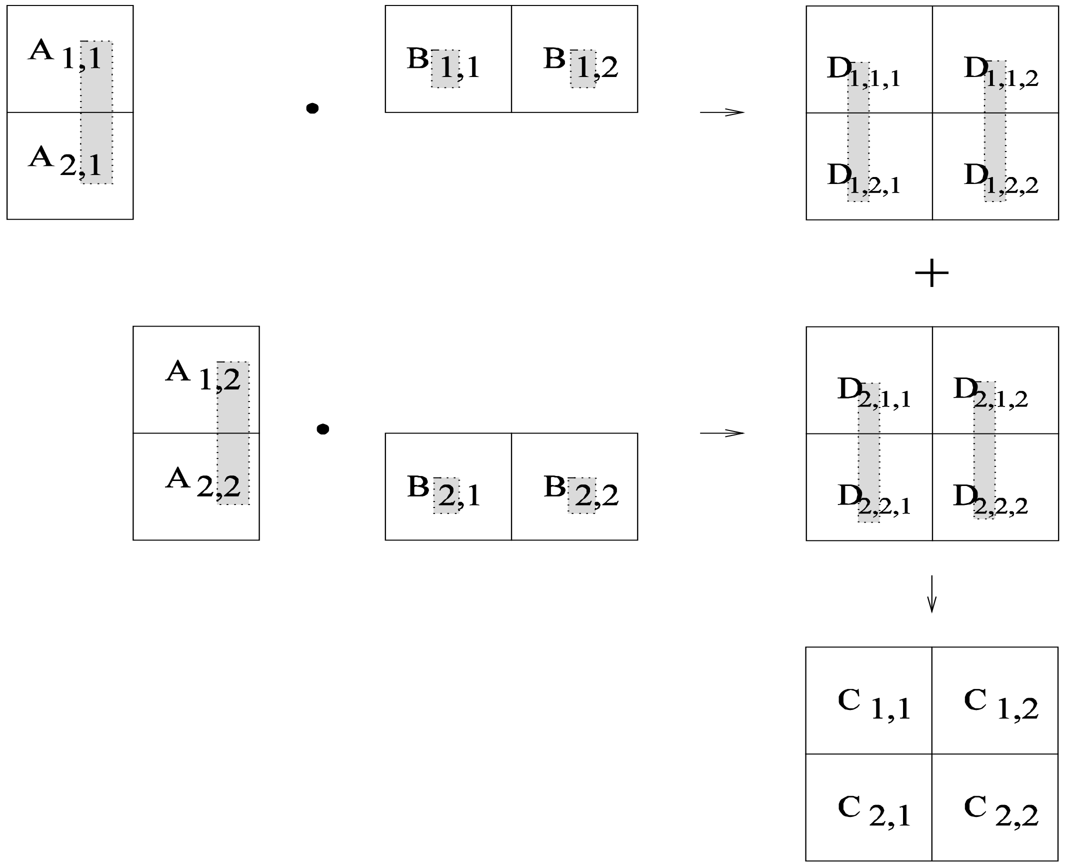
Decomposition I	**Decomposition II**
Task 1: $C_{1,1} = A_{1,1} B_{1,1}$	Task 1: $C_{1,1} = A_{1,1} B_{1,1}$
Task 2: $C_{1,1} = C_{1,1} + A_{1,2} B_{2,1}$	Task 2: $C_{1,1} = C_{1,1} + A_{1,2} B_{2,1}$
Task 3: $C_{1,2} = A_{1,1} B_{1,2}$	Task 3: $C_{1,2} = A_{1,2} B_{2,2}$
Task 4: $C_{1,2} = C_{1,2} + A_{1,2} B_{2,2}$	Task 4: $C_{1,2} = C_{1,2} + A_{1,1} B_{1,2}$
Task 5: $C_{2,1} = A_{2,1} B_{1,1}$	Task 5: $C_{2,1} = A_{2,2} B_{2,1}$
Task 6: $C_{2,1} = C_{2,1} + A_{2,2} B_{2,1}$	Task 6: $C_{2,1} = C_{2,1} + A_{2,1} B_{1,1}$
Task 7: $C_{2,2} = A_{2,1} B_{1,2}$	Task 7: $C_{2,2} = A_{2,1} B_{1,2}$
Task 8: $C_{2,2} = C_{2,2} + A_{2,2} B_{2,2}$	Task 8: $C_{2,2} = C_{2,2} + A_{2,2} B_{2,2}$

Intermediate Data Partitioning

- Computation can often be viewed as a sequence of transformation from the input to the output data.
- In these cases, it is often beneficial to use one of the intermediate stages as a basis for decomposition.

Intermediate Data Partitioning: Example

Let us revisit the example of dense matrix multiplication. We first show how we can visualize this computation in terms of intermediate matrices D.



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Intermediate Data Partitioning

A decomposition of intermediate data structure leads to the following decomposition into 8 + 4 tasks:

Stage I

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \left(\begin{array}{c} \begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} \\ \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \end{array} \right)$$

Stage II

$$\begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} + \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Task 01: $\boldsymbol{D}_{1,1,1} = \boldsymbol{A}_{1,1}\,\boldsymbol{B}_{1,1}$

Task 02: $\boldsymbol{D}_{2,1,1} = \boldsymbol{A}_{1,2}\,\boldsymbol{B}_{2,1}$

Task 03: $\boldsymbol{D}_{1,1,2} = \boldsymbol{A}_{1,1}\,\boldsymbol{B}_{1,2}$

Task 04: $\boldsymbol{D}_{2,1,2} = \boldsymbol{A}_{1,2}\,\boldsymbol{B}_{2,2}$

Task 05: $\boldsymbol{D}_{1,2,1} = \boldsymbol{A}_{2,1}\,\boldsymbol{B}_{1,1}$

Task 06: $\boldsymbol{D}_{2,2,1} = \boldsymbol{A}_{2,2}\,\boldsymbol{B}_{2,1}$

Task 07: $\boldsymbol{D}_{1,2,2} = \boldsymbol{A}_{2,1}\,\boldsymbol{B}_{1,2}$

Task 08: $\boldsymbol{D}_{2,2,2} = \boldsymbol{A}_{2,2}\,\boldsymbol{B}_{2,2}$

Task 09: $\boldsymbol{C}_{1,1} = \boldsymbol{D}_{1,1,1} + \boldsymbol{D}_{2,1,1}$

Task 10: $\boldsymbol{C}_{1,2} = \boldsymbol{D}_{1,1,2} + \boldsymbol{D}_{2,1,2}$

Task 11: $\boldsymbol{C}_{2,1} = \boldsymbol{D}_{1,2,1} + \boldsymbol{D}_{2,2,1}$

Task 12: $\boldsymbol{C}_{2,,2} = \boldsymbol{D}_{1,2,2} + \boldsymbol{D}_{2,2,2}$

Exploratory Decomposition

- In many cases, the decomposition of the problem goes hand-in-hand with its execution.
- These problems typically involve the exploration (search) of a state space of solutions.
- Problems in this class include a variety of discrete optimization problems (0/1 integer programmin, etc.), theorem proving, game playing, etc.

Speculative Decomposition

- In some applications, dependencies between tasks are not known a-priori.
- For such applications, it is impossible to identify independent tasks.
- There are generally two approaches to dealing with such applications: conservative approaches, which identify independent tasks only when they are guaranteed to not have dependencies, and, optimistic approaches, which schedule tasks even when they may potentially be erroneous.
- Conservative approaches may yield little concurrency and optimistic approaches may require roll-back mechanism in the case of an error.

Characteristics of Tasks

Once a problem has been decomposed into independent tasks, the characteristics of these tasks critically impact choice and performance of parallel algorithms. Relevant task characteristics include:

- Task generation.
- Task sizes.
- Size of data associated with tasks.

Task Generation

- Static task generation: Concurrent tasks can be identified a-priori (matrix operations).
- Dynamic task generation (generated during computation)

Task Sizes

- Task sizes may be uniform (i.e., all tasks are the same size) or non-uniform.
- Non-uniform task sizes may be such that they can be determined (or estimated) a-priori or not.
- Examples in this class include discrete optimization problems, in which it is difficult to estimate the effective size of a state space.

Size of Data Associated with Tasks

- The size of data associated with a task may be small or large when viewed in the context of the size of the task.
- A small context of a task implies that an algorithm can easily communicate this task to other processes dynamically.
- A large context ties the task to a process, or alternately, an algorithm may attempt to reconstruct the context at another processes as opposed to communicating the context of the task (e.g., 0/1 integer programming).

Characteristics of Task Interactions

- Tasks may communicate with each other in various ways. The associated dichotomy is:
- Static interactions: The tasks and their interactions are known a-priori. These are relatively simpler to code into programs.
- Dynamic interactions: The timing or interacting tasks cannot be determined a-priori. These interactions are harder to code, especially, as we shall see, using message passing APIs.

Characteristics of Task Interactions

- Regular interactions: There is a definite pattern (in the graph sense) to the interactions. These patterns can be exploited for efficient implementation.
- Irregular interactions: Interactions lack well-defined topologies.

Characteristics of Task Interactions

- Interactions may be read-only or read-write.
- In read-only interactions, tasks just read data items associated with other tasks.
- In read-write interactions tasks read, as well as modify data items associated with other tasks.
- In general, read-write interactions are harder to code, since they require additional synchronization primitives.

Characteristics of Task Interactions

- Interactions may be one-way or two-way.
- A one-way interaction can be initiated and accomplished by one of the two interacting tasks.
- A two-way interaction requires participation from both tasks involved in an interaction.
- One way interactions are somewhat harder to code in message passing APIs.

Mapping Techniques

- Once a problem has been decomposed into concurrent tasks, these must be mapped to processes (that can be executed on a parallel platform).
- Mappings must minimize overheads.
- Primary overheads are communication and idling.
- Minimizing these overheads often represents contradicting objectives.
- Assigning all work to one processor trivially minimizes communication at the expense of significant idling.

Mapping Techniques for Minimum Idling

Mapping techniques can be static or dynamic.

- Static Mapping: Tasks are mapped to processes a-priori. For this to work, we must have a good estimate of the size of each task. Even in these cases, the problem may be NP complete.
- Dynamic Mapping: Tasks are mapped to processes at runtime. This may be because the tasks are generated at runtime, or that their sizes are not known.

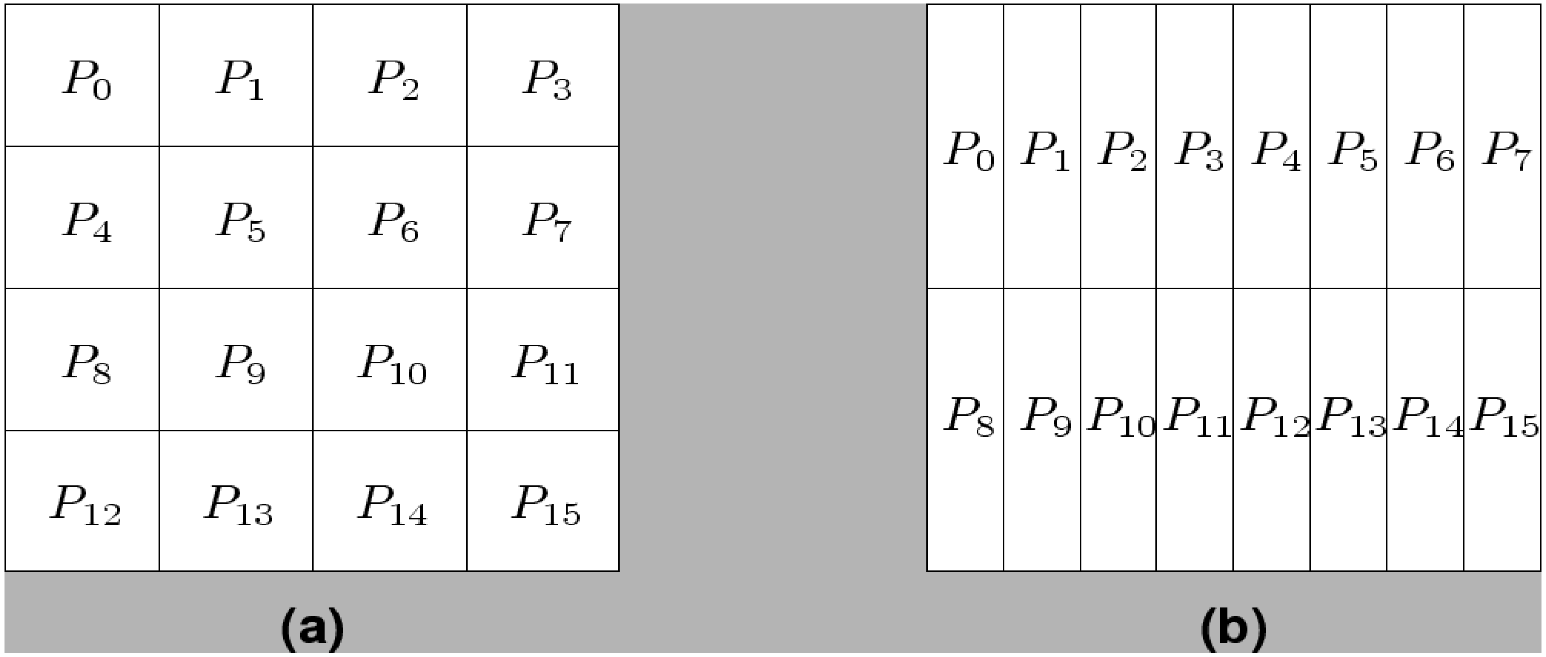
Other factors that determine the choice of techniques include the size of data associated with a task and the nature of underlying domain.

Schemes for Static Mapping

- Mappings based on data partitioning.
- Mappings based on task graph partitioning – functional decomposition
- Hybrid mappings.

Block Array Distribution Schemes

Block distribution schemes can be generalized to higher dimensions as well.

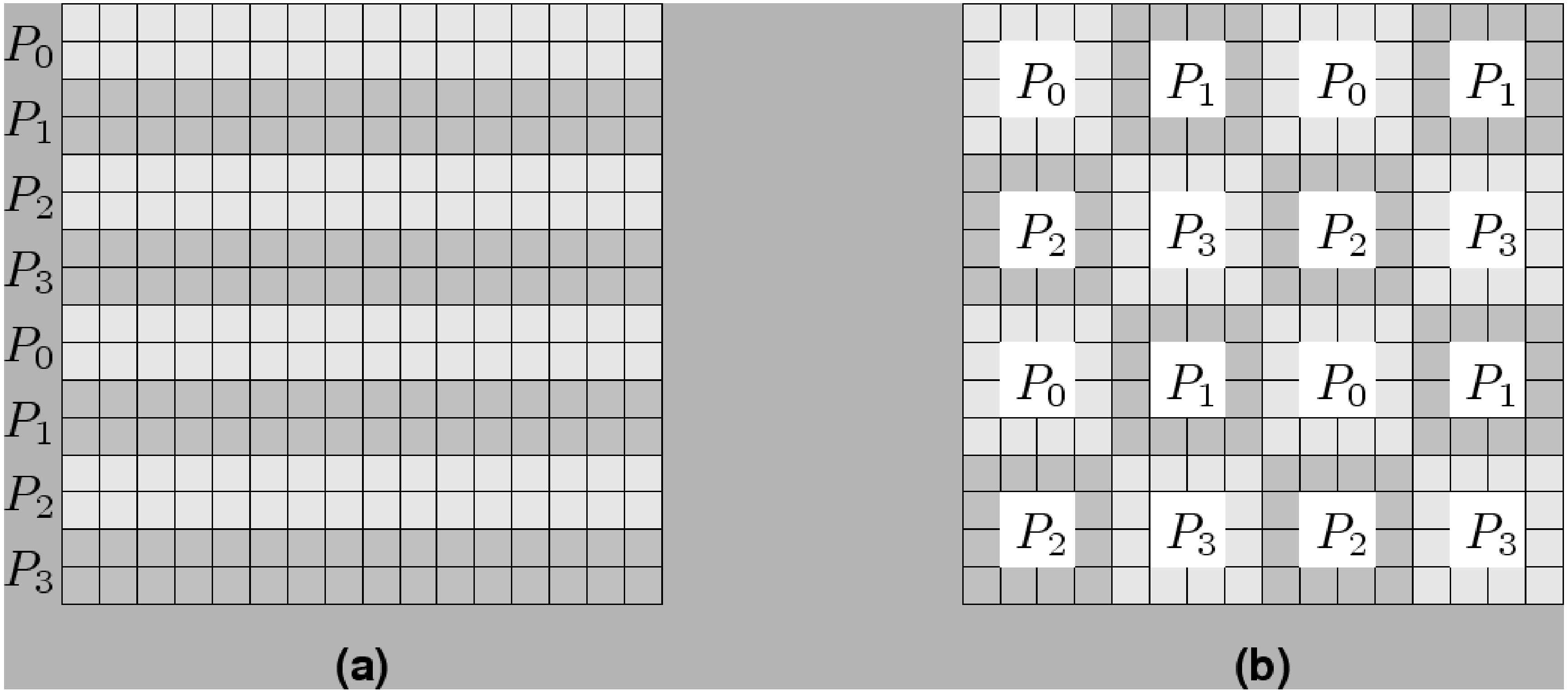

P_0 P_1 P_2 P_3
P_4 P_5 P_6 P_7
P_8 P_9 P_{10} P_{11}
P_{12} P_{13} P_{14} P_{15}
(a)
P_0 P_1 P_2 P_3 P_4 P_5 P_6 P_7
P_8 P_9 P_{10} P_{11} P_{12} P_{13} P_{14} P_{15}
(b)

Cyclic and Block Cyclic Distributions

- If the amount of computation associated with data items varies, a block decomposition may lead to significant load imbalances.
- A simple example of this is in LU decomposition (or Gaussian Elimination) of dense matrices.

Block-Cyclic Distribution

- A cyclic distribution is a special case in which block size is one.
- A block distribution is a special case in which block size is n/p , where n is the dimension of the matrix and p is the number of processes.



Mappings Based on Task Partitioning

- Partitioning a given task-dependency graph across processes.
- Determining an optimal mapping for a general task-dependency graph is an NP-complete problem.
- Excellent heuristics exist for structured graphs.

Hierarchical Mappings

- Sometimes a single mapping technique is inadequate.
- For example, the task mapping of the binary tree (quicksort) cannot use a large number of processors.
- For this reason, task mapping can be used at the top level and data partitioning within each level.

Minimizing Interaction Overheads

- Maximize data locality: Where possible, reuse intermediate data. Restructure computation so that data can be reused in smaller time windows.
- Minimize volume of data exchange: There is a cost associated with each word that is communicated. For this reason, we must minimize the volume of data communicated.
- Minimize frequency of interactions: There is a startup cost associated with each interaction. Therefore, try to merge multiple interactions to one, where possible.
- Minimize contention and hot-spots: Use decentralized techniques, replicate data where necessary.

Minimizing Interaction Overheads (continued)

- Overlapping computations with interactions: Use non-blocking communications, multithreading, and prefetching to hide latencies.
- Replicating data or computations.
- Using group communications instead of point-to-point primitives.
- Overlap interactions with other interactions.

Parallel Algorithm Models

- An algorithm model is a way of structuring a parallel algorithm by selecting a decomposition and mapping technique and applying the appropriate strategy to minimize interactions.
- Data Parallel Model (Data Decomposition): Tasks are statically (or semi-statically) mapped to processes and each task performs similar operations on different data.
- Task Graph Model (Functional Decomposition): Starting from a task dependency graph, the interrelationships among the tasks are utilized to promote locality or to reduce interaction costs.

Parallel Algorithm Models (cont.)

- SPMD – Single Program Multiple Data model
- MPMD – Multiple Programs Multiple Data model
- Master-Slave Model: One or more processes generate work and allocate it to worker processes. This allocation may be static or dynamic.
- Pipeline / Producer-Consumer Model: A stream of data is passed through a succession of processes, each of which perform some task on it.
- Hybrid Models: A hybrid model may be composed either of multiple models applied hierarchically or multiple models applied sequentially to different phases of a parallel algorithm.

Parallel Algorithms Design

- Start from an existing sequential algorithm and design new parallel one.
- Start from existing parallel algorithms and improve it.
- Design completely new parallel algorithm.

Parallel Algorithms Design - Summary

- Choose the initial decomposition technique depending on the given problem.
- Replicate the data to minimize the communication if necessary
- Define initial task sizes.
- Map initial parallel algorithm onto parallel architecture.
- Calibrate the algorithm by optimizing the task size and minimizing the communication.
- Arrive iteratively into the refined parallel algorithm.

Remember, the parallel algorithm has to be much faster than the sequential one!