www.bsc.es

EXCELENCIA
SEVERO
OCHOA

BSC

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

Runtime Aware Architectures

Prof. Mateo Valero, UPC

erc

BSC Director

HiPEAC

A Computer Architecture Workshop: Visions for the Future (Celebrating Yale@75)

Austin September 2014

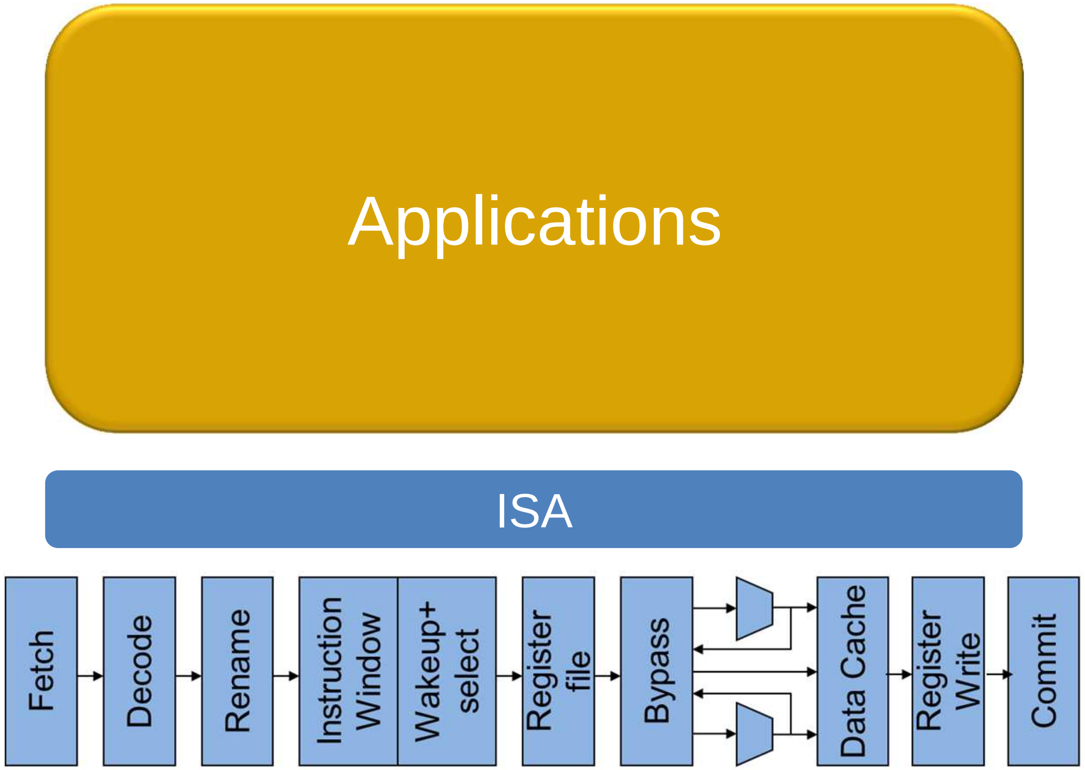
The
Yale Patt
ity of Texas
CEDI 2007

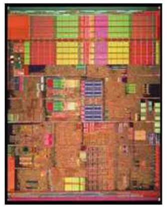
Approaches to increase single processor performance

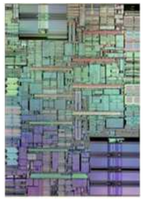
- Exploit Instruction Level Parallelism (ILP)
  - Superscalar processors
  - VLIW processors
- Exploit Data Level Parallelism (DLP)
  - Vector processors
  - SIMD extensions
- Exploit Thread Level Parallelism (TLP)
  - Simultaneous Multi-Threading (SMT) processors
  - Chip Multi-Processing (CMP) processors

Design of Superscalar Processors

- Decoupled from the software stack





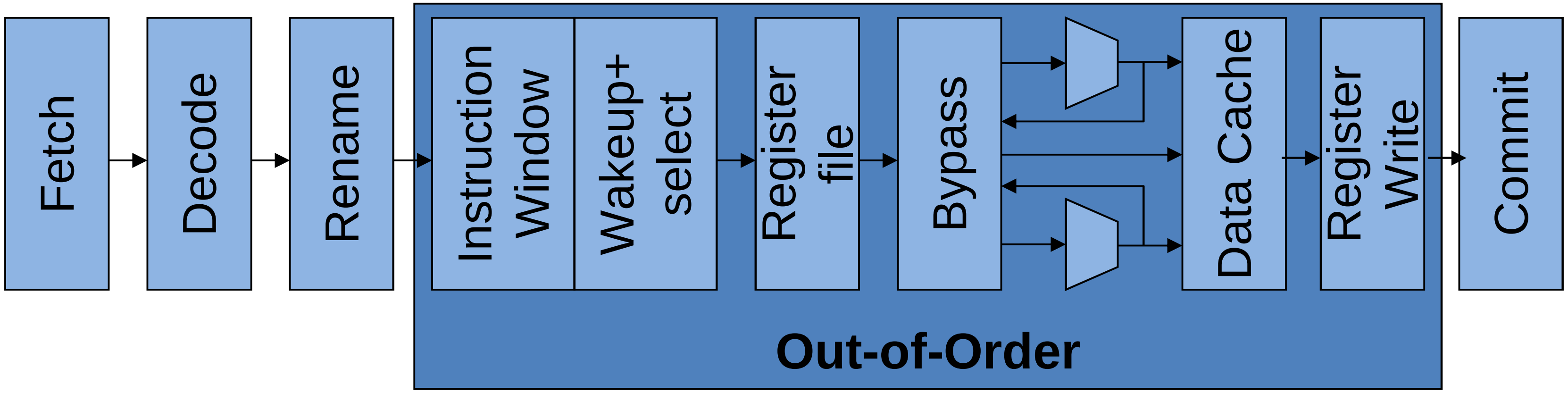


Programs
“decoupled”
from hardware

Simple interface
Sequential program

ILP

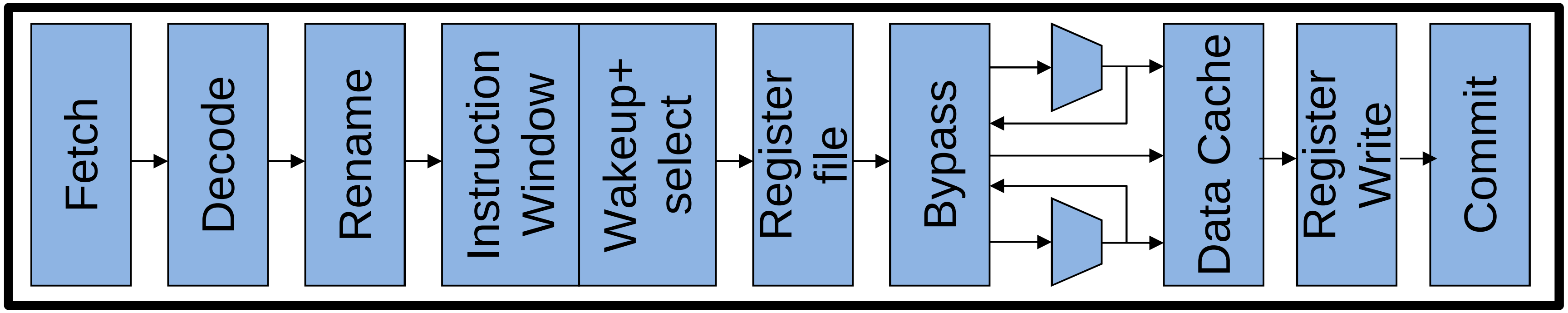
Superscalar Processors



- Fetch multiple instructions every cycle
- Register renaming to eliminate added dependencies
- Instructions wait for source operands and functional units
- Out-of-Order (OoO) execution, but in order graduation
- Predict branches and speculative execution

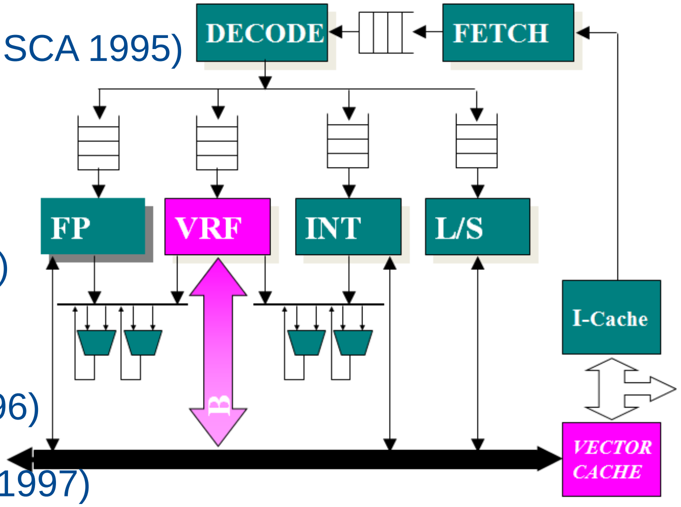
J.E. Smith and S.Vajapeyam. IEEE Computer. Sept. 1997.

Superscalar Processors (Some UPC Contributions)



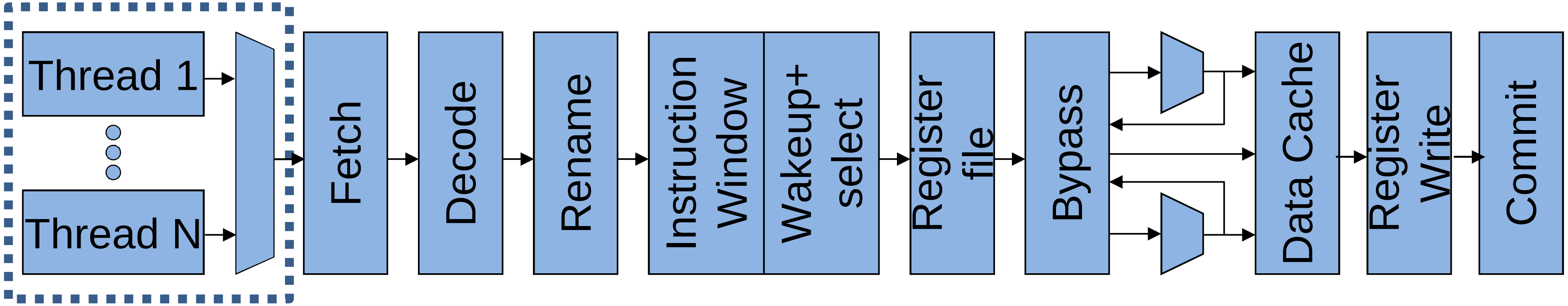
- Software Trace Cache (ICS'99)
- Prophet/Critic Hybrid Branch Prediction (ISCA'04)
- Virtual-Physical Registers (HPCA'98)
- Two-level Register File (ISCA'00, MICRO'00)
- Non Consistent Register File for VLIW (HPCA'95)
- Cache Memory with Hybrid Mapping (IASTED87). Later called Victim Cache
- Dual Data Cache (ICS95)
- Fuzzy Computation (ICS'01, IEEE CAL'02, IEEE Trans. Comput.'05). Currently known as *Approximate Computing* ☺
- Kilo-Instruction Processors (ISHPC'03, HPCA'06, ISCA'08)
- Checkpointing Mechanism for The Memory Consistency Model in Multiprocessors. Later, people called it Bulk Committ ☺

Advanced Vector Architectures (Some UPC Contributions)



- Out-of-Order Access to Vectors (ISCA 1992, ISCA 1995)
- Command Memory Vector (PACT 1998)
  - In-memory computation
- Decoupled Vector Architectures (HPCA 1996)
  - Cray SX1
- Out-of-order Vector Architectures (Micro 1996)
- Multithreaded Vector Architectures (HPCA 1997)
- SMT Vector Architectures (HICS 1997, IEEE MICRO J. 1997)
- Vector register-file organization (PACT 1997)
- Vector Microprocessors (ICS 1999, SPAA 2001)
- Architectures with Short Vectors (PACT 1997, ICS 1998)
  - Tarantula (ISCA 2002), Knights Corner
- Vector Architectures for Multimedia (HPCA 2001, Micro 2002)
- VLIW vector architectures for multimedia (paper salami)
- High-Speed Buffers Routers (Micro 2003, IEEE TC 2006)
- Vector Architectures for Data Bases (Micro 2012)

Exploiting TLP(Some UPC Contributions)

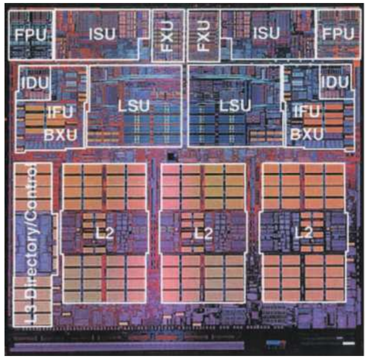


- Simultaneous Multithreading (SMT)
  - Benefits of SMT Processors:
    - Increase core resource utilization
  - Basic pipeline unchanged:
    - Few replicated resources, other shared
- Some of our contributions:
  - Dynamically Controlled Resource Allocation (MICRO 2004)
  - Quality of Service (QoS) in SMTs (IEEE TC 2006)
  - Runahead Threads for SMTs (HPCA 2008)

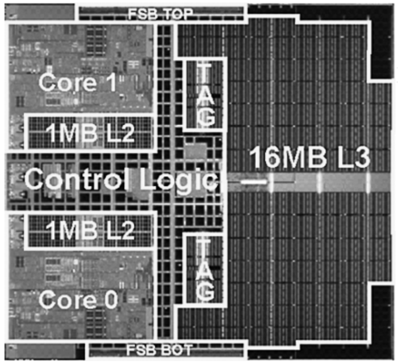
The MultiCore Era

- Moore's Law + Memory Wall + Power Wall

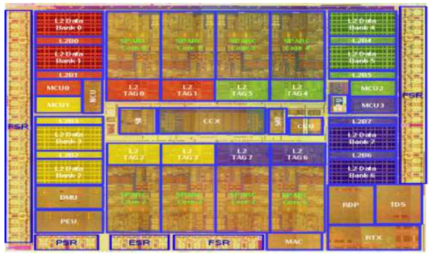
Chip MultiProcessors (CMPs)



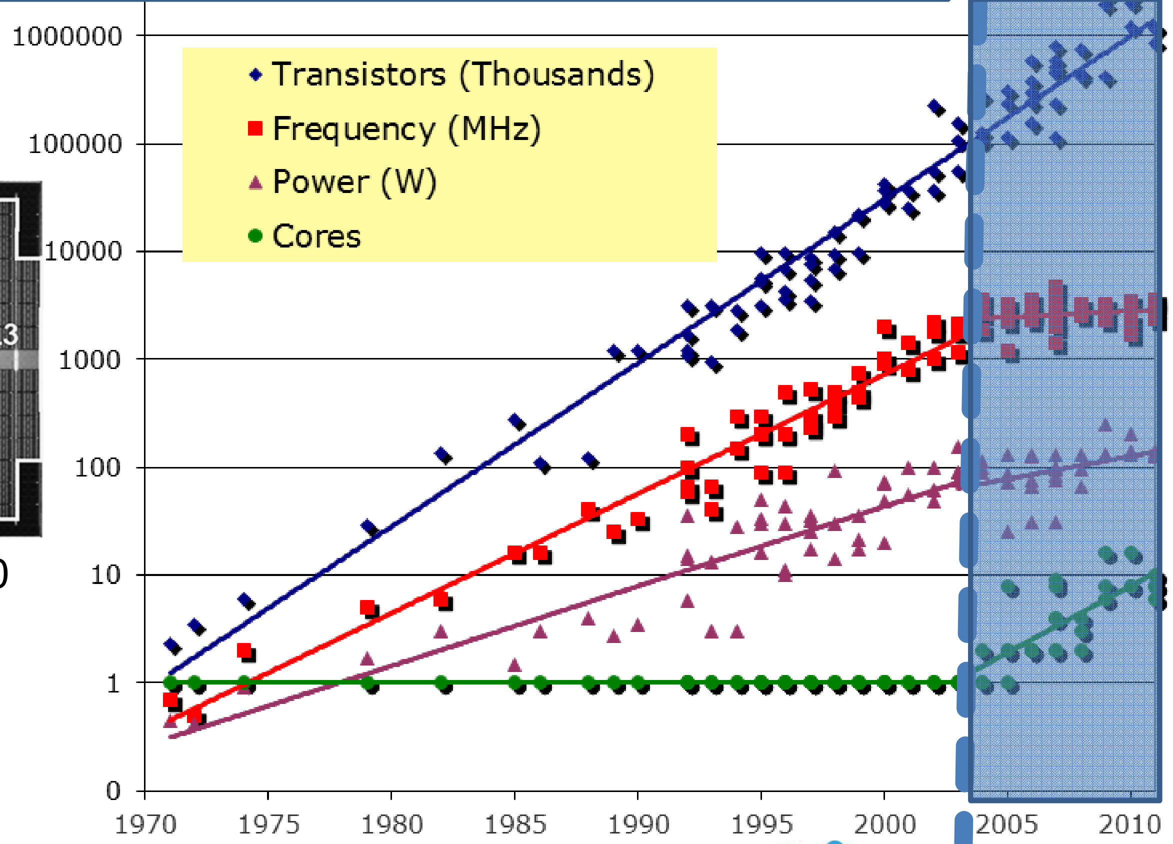
POWER4 (2001)



Intel Xeon 7100 (2006)



UltraSPARC T2 (2007)



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

RoMoL Project

How are the Multicore architectures designed?

IBM Power4 (2001)

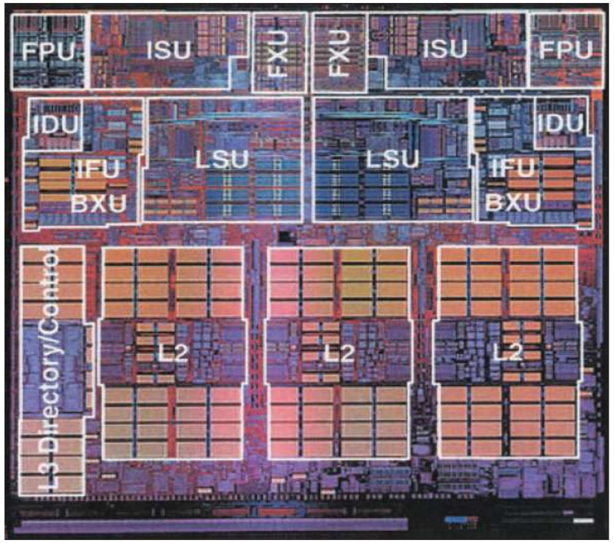
- 2 cores, ST
- 0.7 MB/core L2, 16MB/core L3 (off-chip)
- 115W TDP
- 10GB/s mem BW

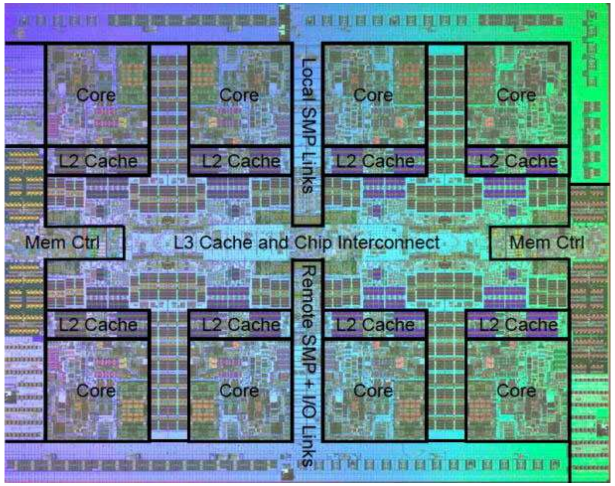
IBM Power7 (2010)

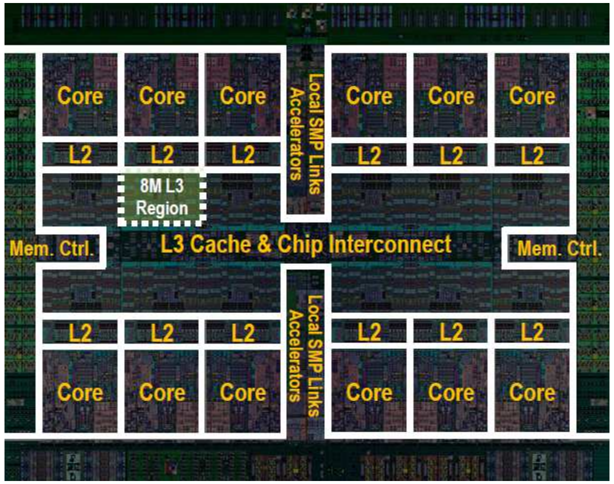
- 8 cores, SMT4
- 256 KB/core L2 16MB/core L3 (on-chip)
- 170W TDP
- 100GB/s mem BW

IBM Power8 (2014)

- 12 cores, SMT8
- 512 KB/core L2 8MB/core L3 (on-chip)
- 250W TDP
- 410GB/s mem BW







Barcelona Supercomputing Center
Centro Nacional de Supercomputación

RoMoL Project

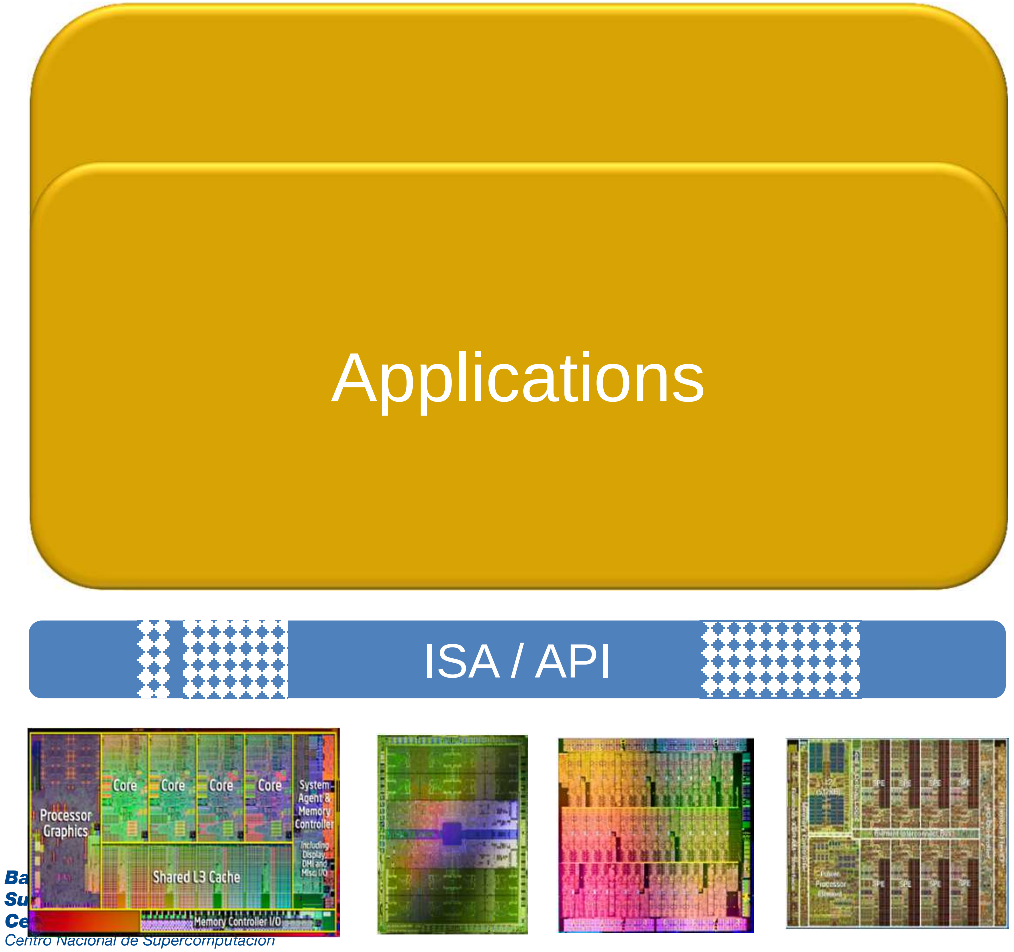
Challenges of Multi-core Processors Design

- Efficient design of future parallel systems challenges:
  - Memory Wall
  - Power Wall
  - Programmability Wall
  - Upcoming Reliability Wall
  - Wide range of application domains from mobiles up to supercomputers
- Many-cores exacerbate the problems derived from the Walls
- A tight collaboration between software and hardware is a must

Question: How should we program, use and design those parallel systems?

Living in the programming revolution

- Multicores made the interface to leak…

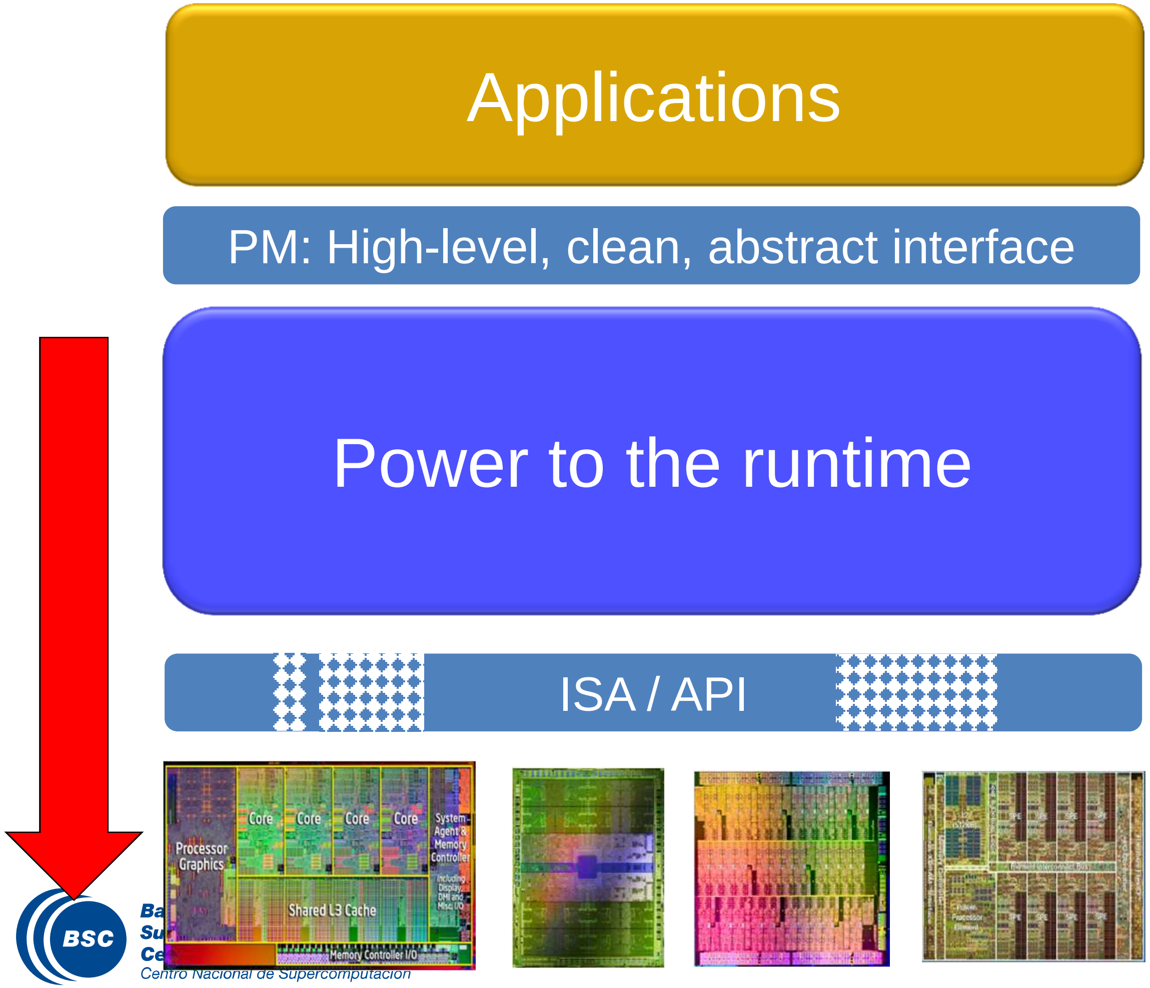


Parallel application logic
+
Platform specificites

Parallel hardware with multiple address spaces (hierarchy, transfer), control flows, …

Vision in the programming revolution

- Need to decouple again



Applications

PM: High-level, clean, abstract interface

Power to the runtime

ISA / API

Application logic

Arch. independent

General purpose

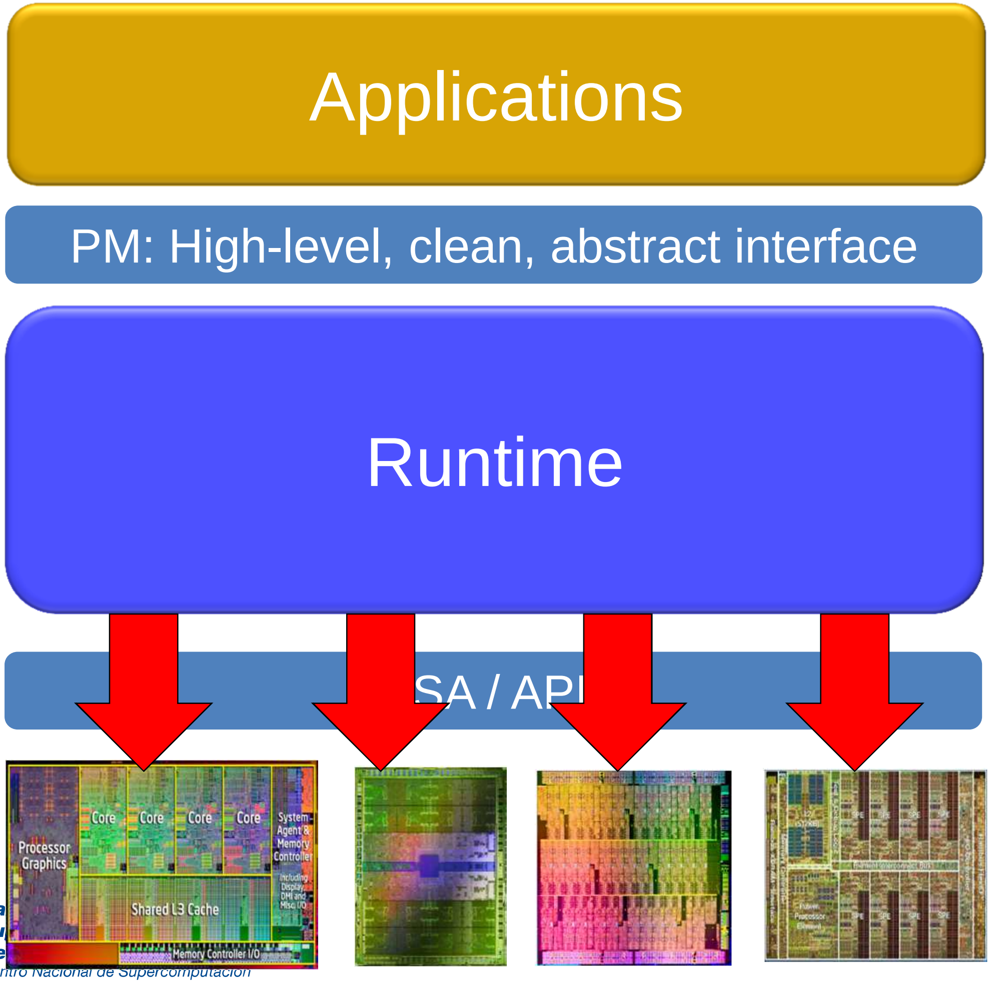
Single address space

The efforts are focused on **efficiently using** the underlying hardware

RoMoL Project

Runtime Aware Architectures (RoMoL Project)

- The runtime **drives** the hardware design



Applications

PM: High-level, clean, abstract interface

Runtime

SA / API

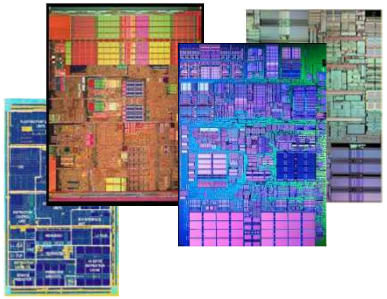
Task based PM annotated by the user

Data dependencies detected at runtime

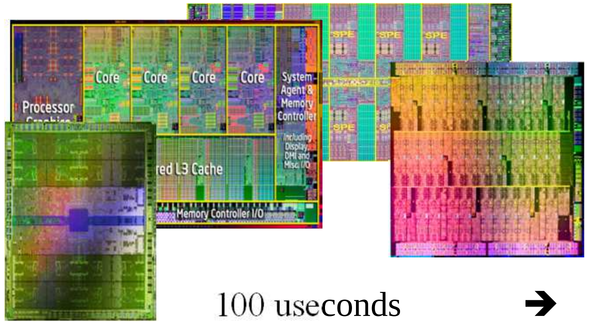
Dynamic scheduling

“Reuse” architectural ideas under new constraints

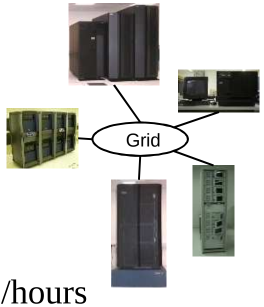
Computing: a matter of perspective



≅



≅



ns → 100 useconds → minutes/hours

Mapping of concepts:

Instructions	→ Block operations	→ Full binary
Functional units	→ Cores	→ machines
Fetch &decode unit	→ Core	→ home machine
Registers (name space)	→ Main memory	→ Files
Registers (storage)	→ Local memory	→ Files

<u>Granularity</u>

Stay sequential

Just look at things from a bit further away

Architects do know how to run parallel

OmpSs: a sequential program …

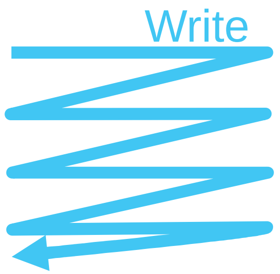
```
void vadd3 (float A[BS], float B[BS],
            float C[BS]);

void scale_add (float sum, float A[BS],
               float B[BS]);

void accum (float A[BS], float *sum);
```

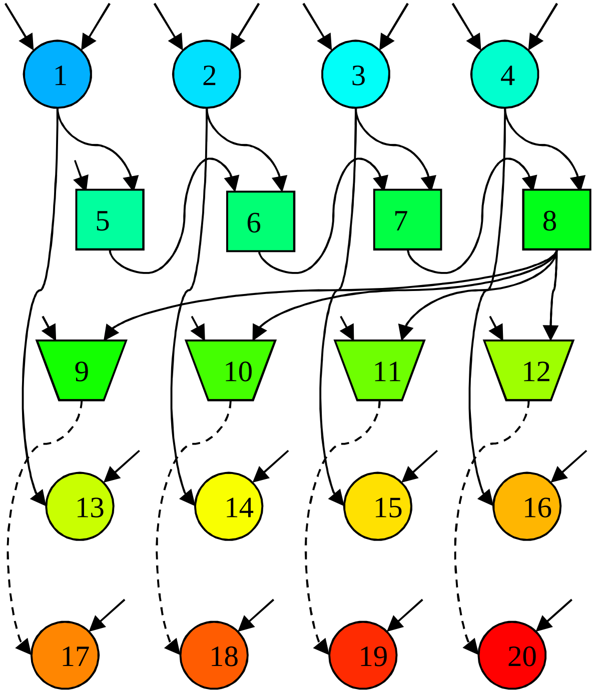
```
for (i=0; i<N; i+=BS)             // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)            //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)            // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)            // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)            // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

OmpSs: … taskified …

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],                ●
            float C[BS]);
#pragma css task input(sum, A) inout(B)
void scale_add (float sum, float A[BS],              ▼
               float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);                ■
```

```
for (i=0; i<N; i+=BS)             // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)            //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)            // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)            // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)            // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```





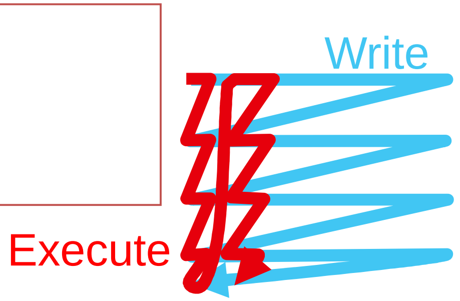
Color/number: order of task instantiation
Some antidependences covered by flow dependences not drawn

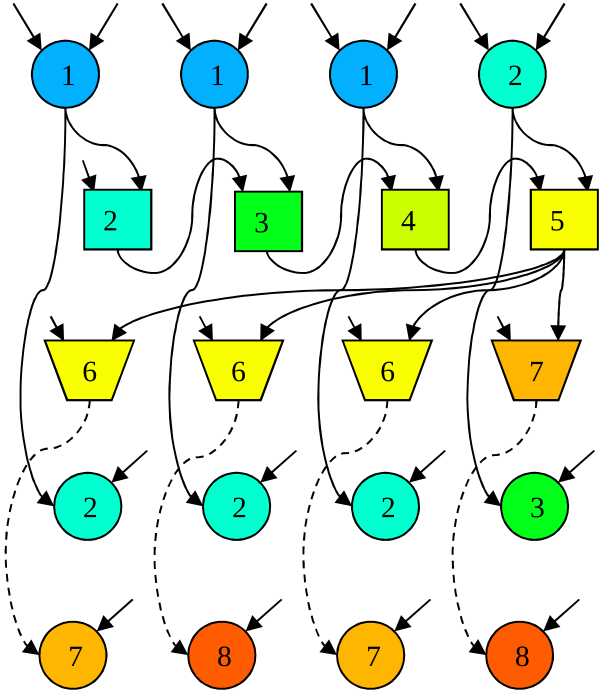
OmpSs: … and executed in a data-flow model

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
            float C[BS]);
#pragma css task input(sum, A) inout(B)
void scale_add (float sum, float A[BS],
                float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)                  // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)                 //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)                 // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)                 // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)                 // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

Decouple
how we write
form
how it is executed





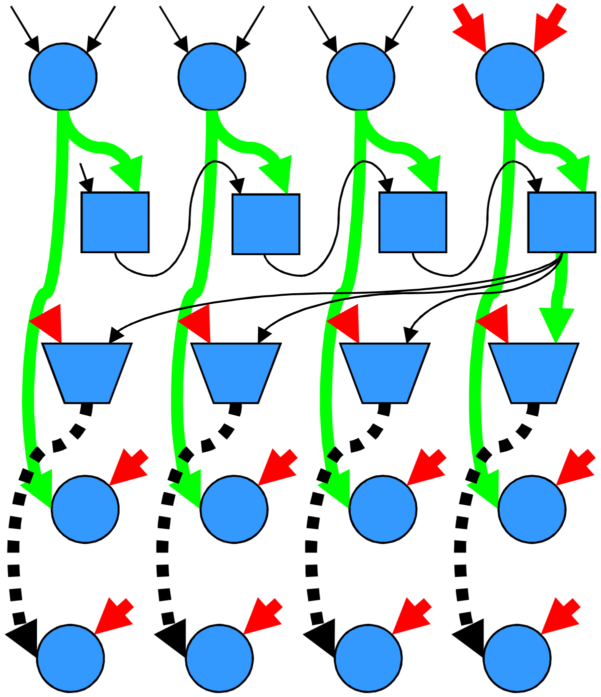
Color/number: a possible order of task execution

OmpSs: the potential of data access information

- **Flat global address space seen by programmer**

- Flexibility to dynamically traverse dataflow graph “optimizing”
  - Concurrency. Critical path
  - Memory access: data transfers performed by run time

- Opportunities for automatic
  - Prefetch →
  - Reuse →
  - Eliminate antidependences (rename) ⇢
  - Replication management
    - Coherency/consistency handled by the runtime
    - Layout changes



Runtime Aware Architectures (RAA)

- Re-design memory hierarchy
  - Hybrid (cache + local memory)
  - Non-volatile memory, 3D stacking
  - Simplified coherence protocols, non-coherent islands of cores
- Exploitation of data locality:
  - Reuse, prefetching, in-memory computation

Memory Wall

- Heterogeneity of tasks and Hardware
  - Critical path exploitation
- Accelerators
  - Numerical, data bases, proteomics, big data

Power Wall

- Efficient data movement
  - Overlap communication and computation
  - Latency aware interconnection network

Program. Wall

- Hardware acceleration of the runtime system
  - Task dependency graph management
- Load balancing and scheduling
  - Asynchrony and critical path exploitation

Resilience Wall

- Task-based check-pointing
- Algorithmic-based fault tolerance

Bringing the Superscalar vision to the Multicore level

Superscalar World	Multicore World
Out-of-Order, Kilo-Instruction Processor, Distant Parallelism	Task-based, Data-flow Graph, Dynamic Parallelism
Branch Predictor, Speculation Fuzzy Computation	Tasks Output Prediction, Speculation
Dual Data Cache, Sack for VLIW	Hybrid Memory Hierarchy, NVM
Register Renaming, Virtual Regs	Late Task Memory Allocation
Cache Reuse, Prefetching, Victim C.	Data Reuse, Prefetching
In-memory Computation	In-memory FU's
Accelerators, Different ISA's, SMT	Heterogeneity of Tasks and HW
Critical Path Exploitation	Task-criticality
Resilience	Resilience
	Load Balancing and Scheduling
	Interconnection Network
	Data Movement

Memory Wall	**Power Wall**
Programmability Wall	**Resilience Wall**

Conclusions

- Things are changing very quickly… but many lessons learned in the past decades can be useful nowadays
- The design of future multicore/parallel systems has to dramatically change
  - Hardware-Software co-design
- Runtime has to drive the design of future multicores
- Runtime Aware Architectures will allow
  - Efficient management of parallelism and energy
  - Improve memory management and reduce data movements
  - Increase reliability
- Ensure continued performance improvements, once more Riding on Moore's Law (RoMoL)

RoMoL Team

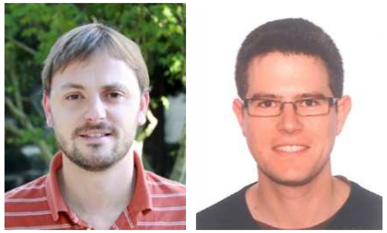
- Riding on Moore's Law (RoMoL, http://www.bsc.es/romol)
  - ERC Advanced Grant: 5-year project 2013 – 2018.

European Research Council
Established by the European Commission

- Our team:
  - **CS Department @ BSC**
  - PI:

Project Coordinators:

  - Staff Researchers:



Staff:

  - Students:

- **Open for collaborations!**

www.bsc.es

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

Thank you !