



**Barcelona  
Supercomputing  
Center**  
Centro Nacional de Supercomputación



# From Classical to Runtime Aware Architectures

**Prof. Mateo Valero**

BSC Director



Barcelona, July 4, 2018



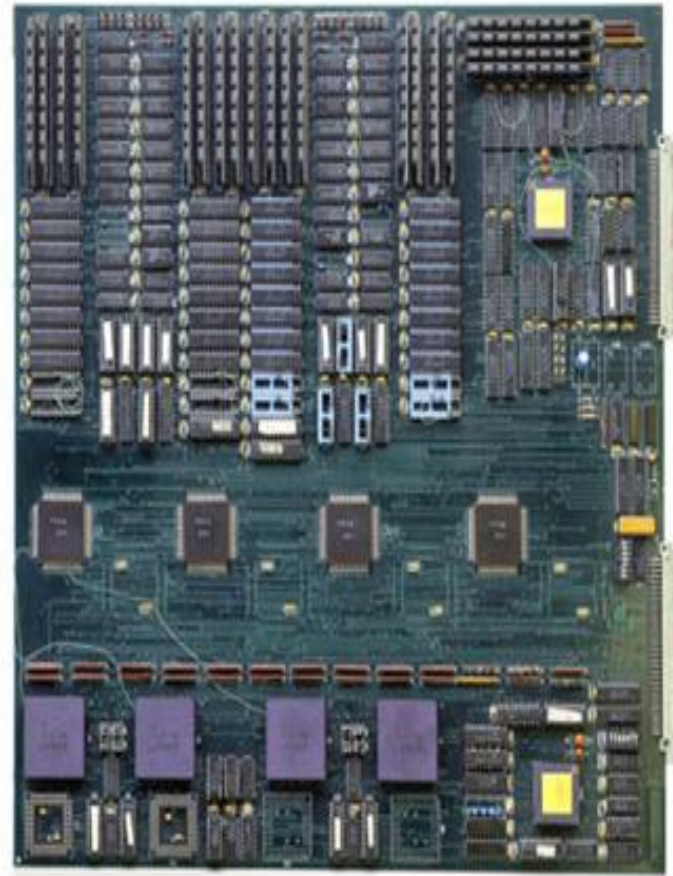
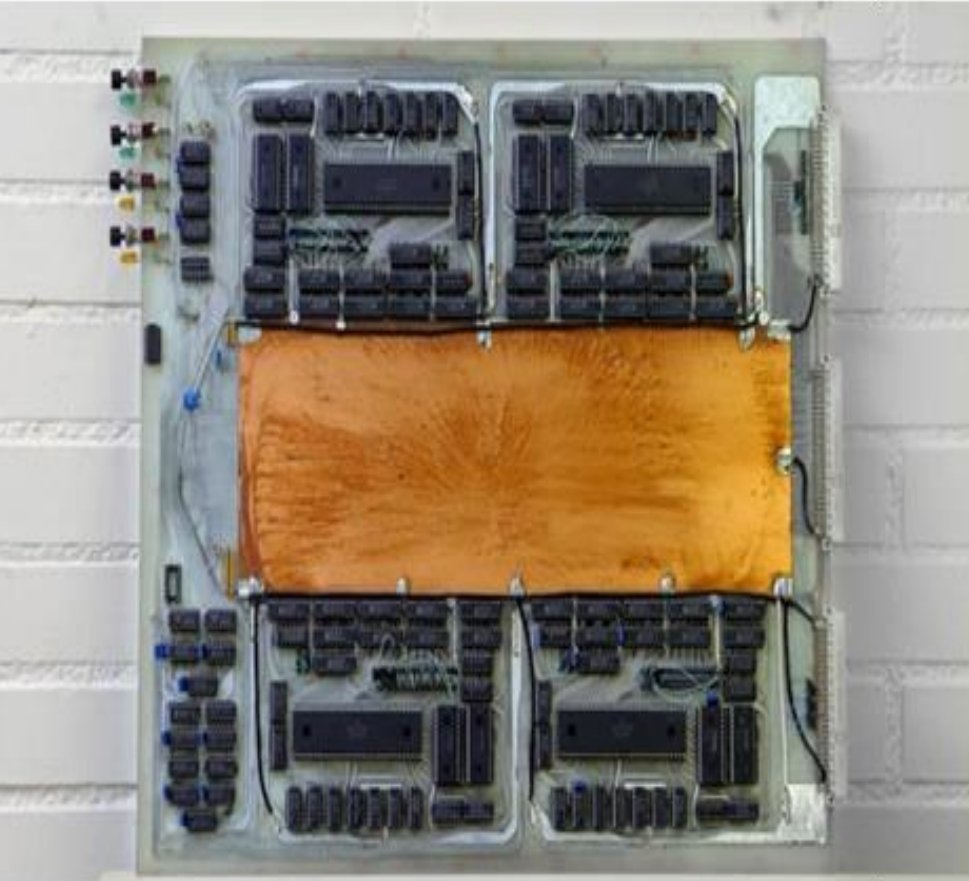
# Professor Tomas Lang



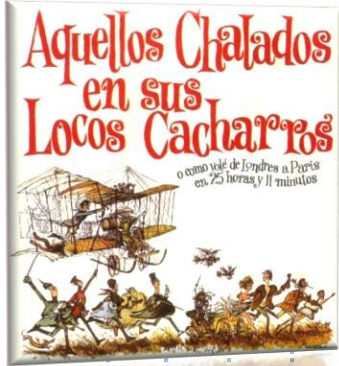


# Once upon a time ...

1986 and 1988 , UPC multiprocessor prototypes



# Our Origins...



**Barcelona Supercomputing Center**  
Centro Nacional de Supercomputación



Parsys Multiprocessor



Parsytec CCI-8D  
4.45 Gflop/s



Compaq GS-140 12.5 Gflop/s  
Compaq GS-160 23.4 Gflop/s

BULL NovaScale 5160  
48 Gflop/s



Maricel  
14.4 Tflops, 20 KW



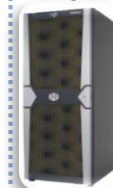
Transputer cluster



Convex C3800



SGI Origin 2000  
32 Gflop/s



SGI Altix 4700  
819.2 Gflops



SL8500  
6 Petabytes



Research prototypes



Connection Machine CM-200  
0,64 Gflop/s



IBM RS-6000 SP & IBM p630  
192+144 Gflop/s



IBM PP970 / Myrinet  
MareNostrum  
42.35, 94.21 Tflop/s

1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010



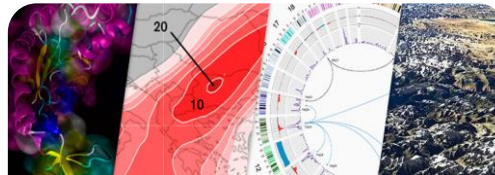
**Barcelona Supercomputing Center**  
Centro Nacional de Supercomputación

# Barcelona Supercomputing Center Centro Nacional de Supercomputación

## BSC-CNS objectives



Supercomputing services  
to Spanish and  
EU researchers



R&D in Computer,  
Life, Earth and  
Engineering Sciences



PhD programme,  
technology transfer,  
public engagement

BSC-CNS is  
a consortium  
that includes

Spanish Government

60%



Catalan Government

30%



Univ. Politècnica de Catalunya (UPC)

10%



# Mission of BSC Scientific Departments



## Computer Sciences

To influence the way machines are built, programmed and used: programming models, performance tools, Big Data, computer architecture, energy efficiency



## Earth Sciences

To develop and implement global and regional state-of-the-art models for short-term air quality forecast and long-term climate applications



## Life Sciences

To understand living organisms by means of theoretical and computational methods (molecular modeling, genomics, proteomics)



## CASE

To develop scientific and engineering software to efficiently exploit super-computing capabilities (biomedical, geophysics, atmospheric, energy, social and economic simulations)

# MareNostrum4

Total peak performance: **13,7 Pflops**

General Purpose Cluster: 11.15 Pflops (1.07.2017)

CTE1-P9+Volta: 1.57 Pflops (1.03.2018)

CTE2-Arm V8: 0.5 Pflops (????)

CTE3-KNH?: 0.5 Pflops (????)



## MareNostrum 1

2004 – 42,3 Tflops

1<sup>st</sup> Europe / 4<sup>th</sup> World

New technologies

## MareNostrum 2

2006 – 94,2 Tflops

1<sup>st</sup> Europe / 5<sup>th</sup> World

New technologies

## MareNostrum 3

2012 – 1,1 Pflops

12<sup>th</sup> Europe / 36<sup>th</sup> World

## MareNostrum 4

2017 – 11,1 Pflops

2<sup>nd</sup> Europe / 13<sup>th</sup> World

New technologies



# MareNostrum 4



# From MN3 to MN4

## MareNostrum 4, chosen as the most beautiful data centre in the world

11 December 2017

The award, organised by DataCenter Dynamics, has been granted by popular vote.



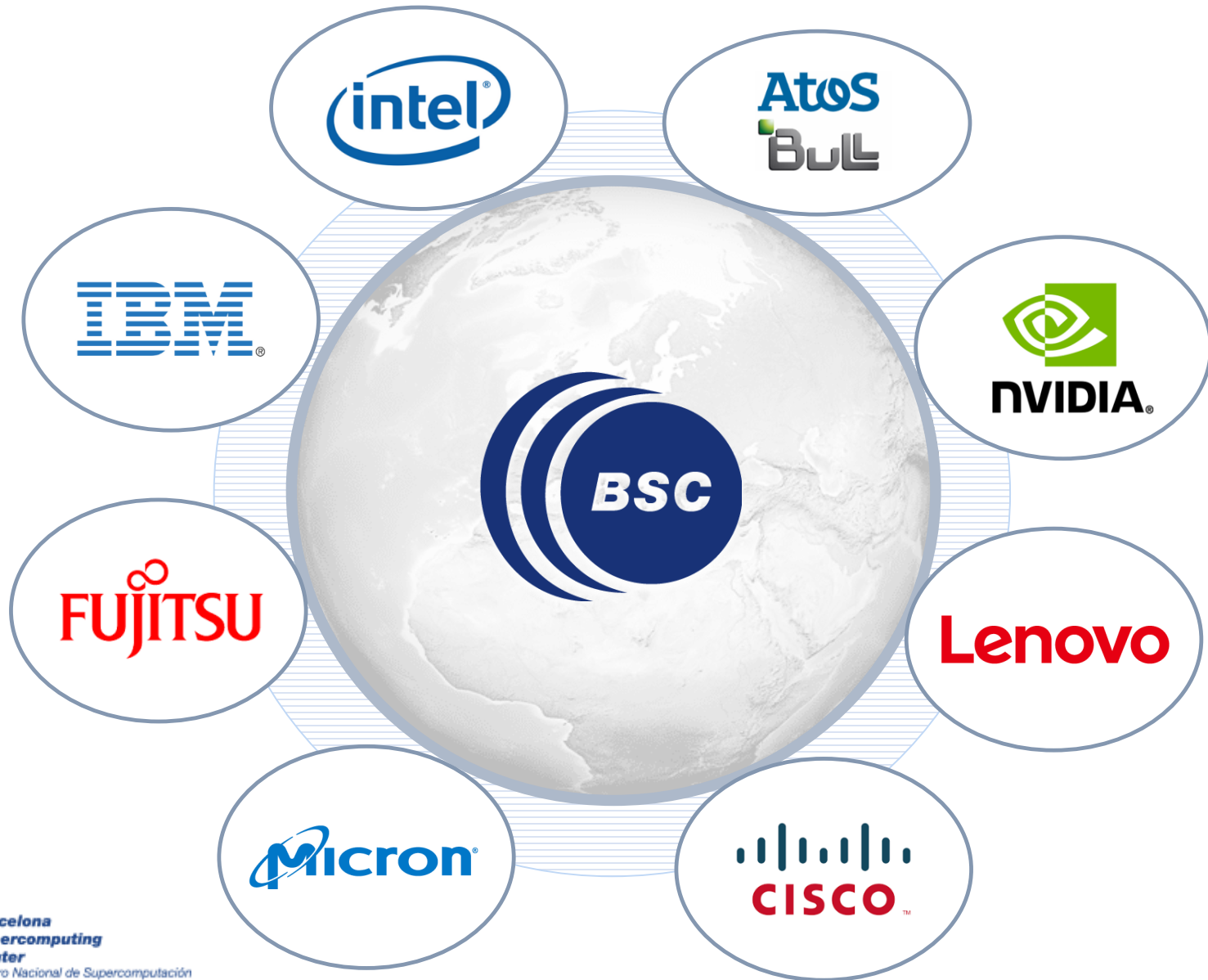
MareNostrum 4 supercomputer has been the winner of the Most Beautiful Data Center in the world Prize, hosted by the [Datacenter Dynamics \(DCD\) Company](#).

There are 15 prizes in different categories, besides the prize for the most beautiful data centre, which is elected by popular vote. MareNostrum 4 competed with such impressive facilities as the Switch Pyramid in Michigan, the Bahnhof Pionen in Stockholm or the Norwegian Green Mountain. BSC supercomputer has prevailed for its particular location, inside the chapel of Torre Girona, located in the North Campus of the Universitat Politècnica de Catalunya (UPC).

The awards ceremony took place on December 7<sup>th</sup> in London and both Mateo Valero, BSC Director, and Sergi Girona, Operations department Director, received the prize.

### About MareNostrum 4

# BSC & The Global IT Industry 2018



# Collaborations with Industry



Research into advanced technologies for the exploration of hydrocarbons, subterranean and subsea reserve modelling and fluid flows



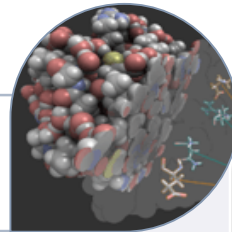
Research on wind farms optimization and wing energy production forecasts



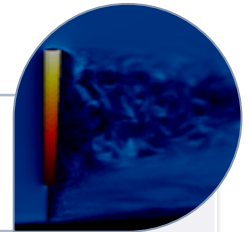
Collaboration agreement for the development of advanced systems of deep learning with applications to banking services



BSC's dust storm forecast system licensed to be used to improve the safety of business flights.



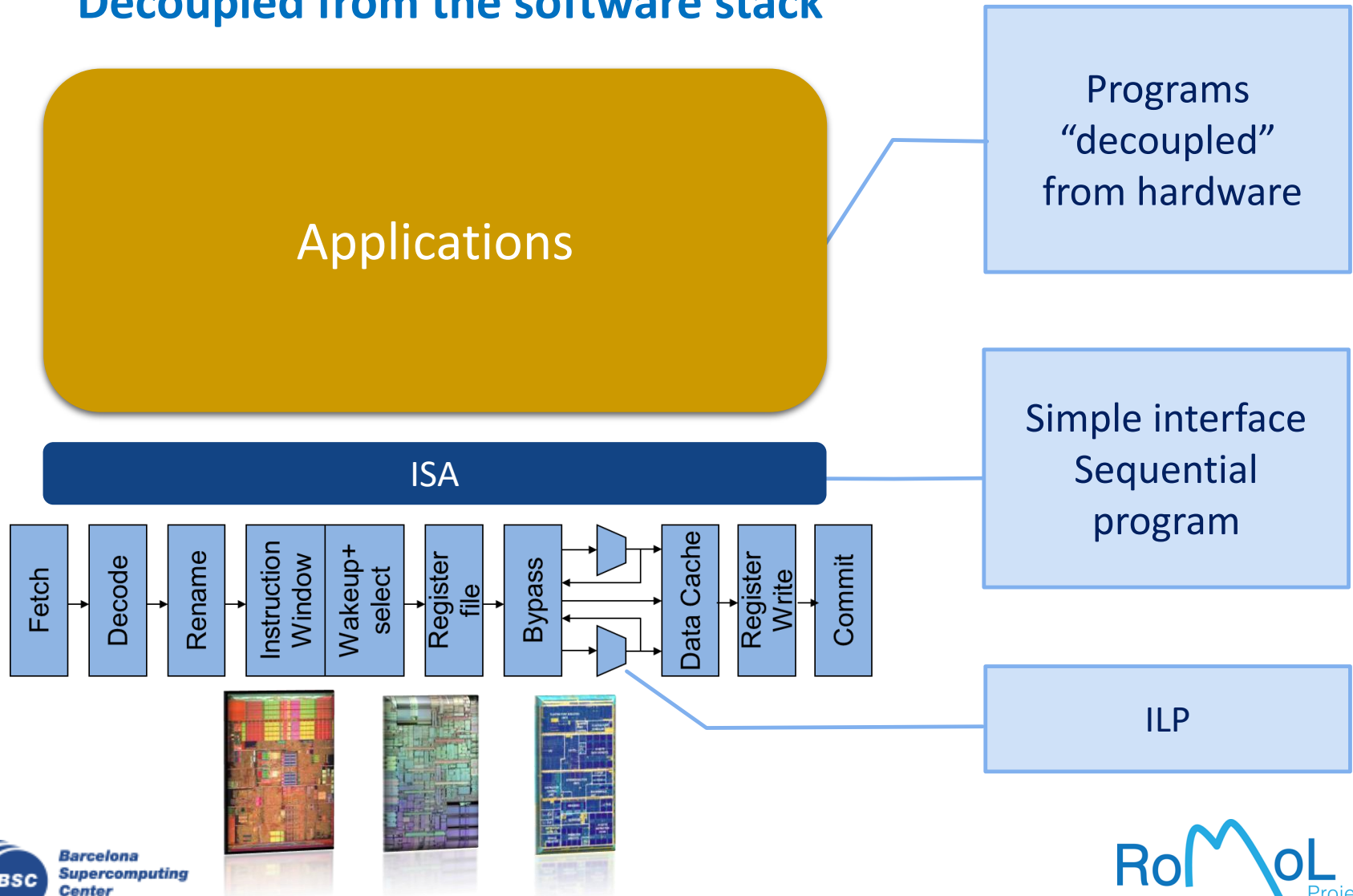
Research on the protein-drug mechanism of action in Nuclear Hormone receptors and developments on PELE method to perform protein energy landscape explorations



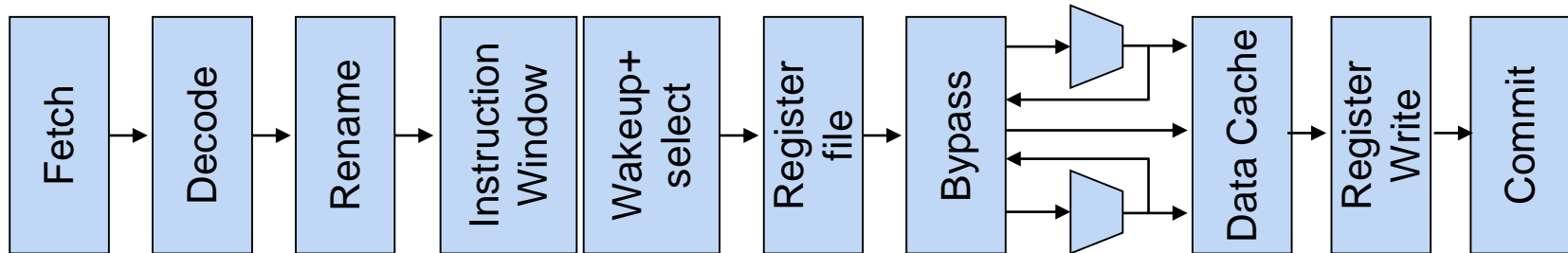
Simulation of fluid-structure interaction problem with the multi-physics software Alya

# Design of Superscalar Processors

Decoupled from the software stack



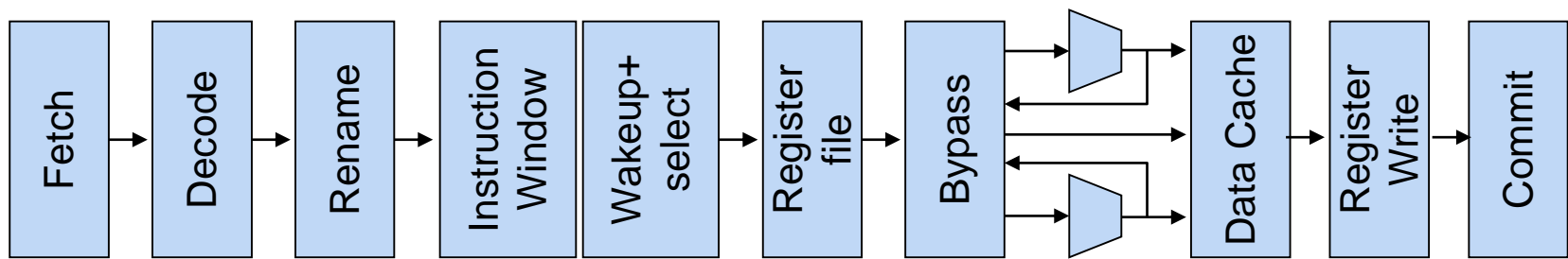
# Latency Has Been a Problem from the Beginning... ☹️



- Feeding the pipeline with the right instructions:
  - Software trace cache (ICS'99)
  - Prophet/Critic Hybrid Branch Predictor (ISCA'04)
- Locality/reuse
  - Cache Memory with Hybrid Mapping (IASTED87). Victim Cache ☺️
  - Dual Data Cache (ISCA'05)
- A novel renaming mechanism that boosts software prefetching (ICS'01)
- Virtual-Physical Registers (HPCA'98)
- Kilo Instruction Processors (ISHPC03,HPCA'06, ISCA'08)

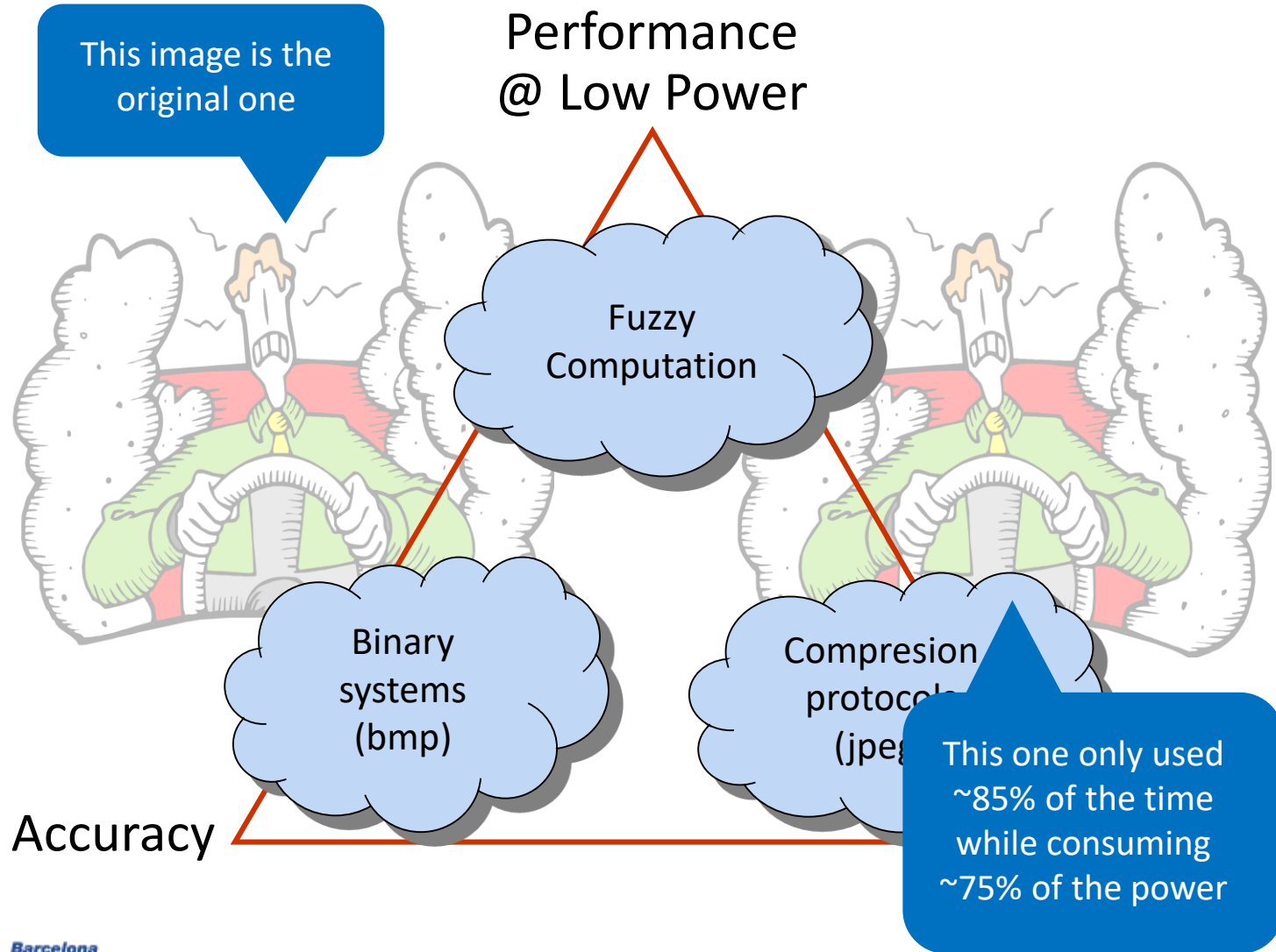
**Memory Wall**

# ... and the Power Wall Appeared Later ☹️☹️☹️



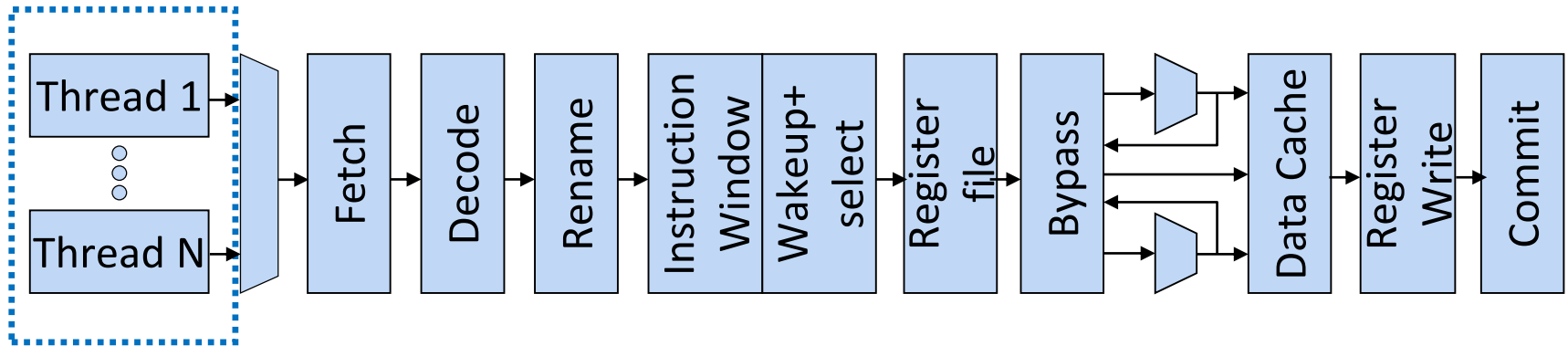
- Better Technologies
- Two-level organization (Locality Exploitation)
  - Register file for Superscalar (ISCA'00)
  - Instruction queues (ICCD'05)
  - Load/Store Queues (ISCA'08)
- Direct Wakeup, Pointer-based Instruction Queue Design (ICCD'04, ICCD'05)
- Content-aware register file (ISCA'09)
- Fuzzy computation (ICS'01, IEEE CAL'02, IEEE-TC'05). Currently known as Approximate Computing 😊

# Fuzzy computation





# SMT and Memory Latency .. 😊



- 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)

# Time Predictability (in multicore and SMT processors)

## Definition:

- Ability to provide a minimum performance to a task
- Requires biasing processor resource allocation

## Where is it required:

- Increasingly required in handheld/desktop devices
- Also in embedded hard real-time systems (car, plane, trains, ...)

## How to achieve it:

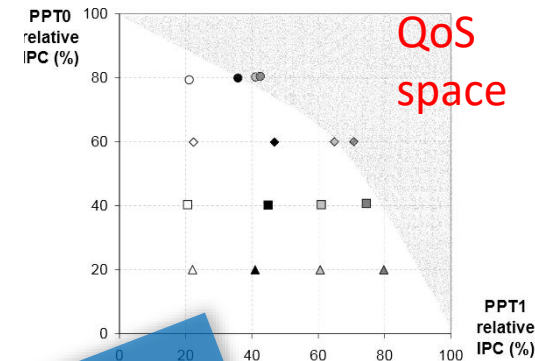
- Controlling how resources are assigned to co-running tasks

## Soft real-time systems

- SMT: DCRA resource allocation policy (MICRO 2004, IEEE Micro 2004)
- Multicore cache partitioning (ACM OSR 2009, IEEE Micro 2009)

## Hard real-time systems

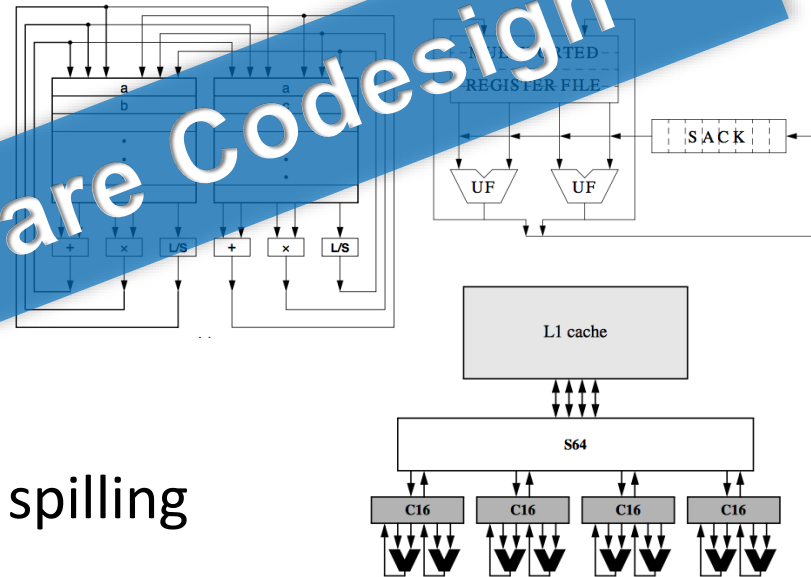
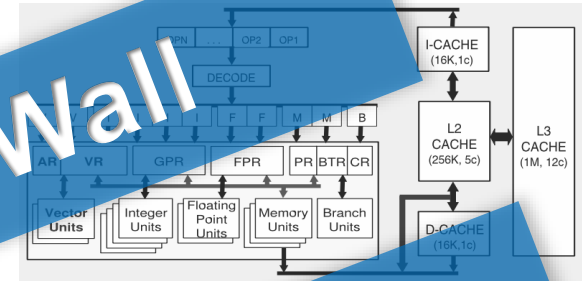
- Deterministic resource 'securing' (ISCA 2009)
- Time-Randomised designs (DAC 2014 best paper award)



**Predictability Wall**

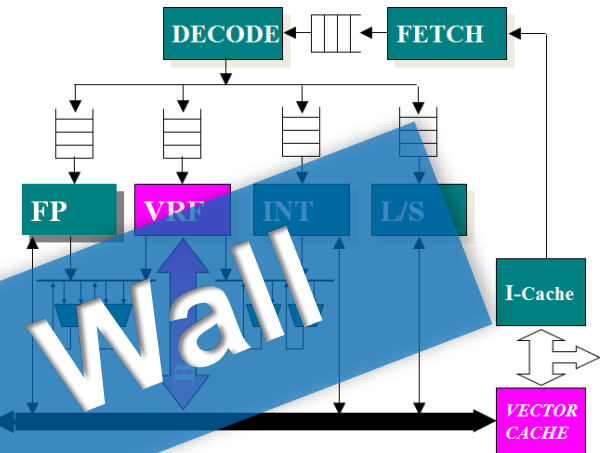
# Statically scheduled VLIW architectures

- Power-efficient FU
  - Clustering
  - Widening (MICRO-98)
  - $\mu$ SIMD and multimedia vector units (ICPP-05)
- Locality-aware R
  - Sack (CONPAR-94)
  - Non-consistent (HPCA95)
  - Two-level hierarchical (MICRO-00)
- Integrated scheduling techniques, register allocation and spilling (MICRO-95, PACT-96, MICRO-96, MICRO-01)

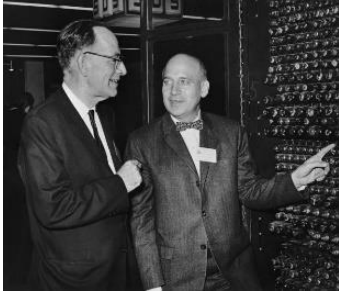


# Vector Architectures... Memory Latency and Power ☺☺☺

- Out-of-Order Access to Vectors (ISCA 1992, ISCA 1995)
- Command Memory Vector (PACT 1998)
  - In-memory computation
- Decoupling 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 1999)
- Vector register-file organization (PACT 1997)
- Vector Microprocessors (ISCA 1999, SPAA 2001)
- Architectures with Short Vectors (PACT 1997, ICS 1998)
  - Cray T3E (ISCA 2002), Knights Corner
- Vector Architectures for Multimedia (HPCA 2001, Micro 2002)
- High-Speed Buffers Routers (Micro 2003, IEEE TC 2006)
- Vector Architectures for Data-Base (Micro 2012, HPCA2015, ISCA2016)



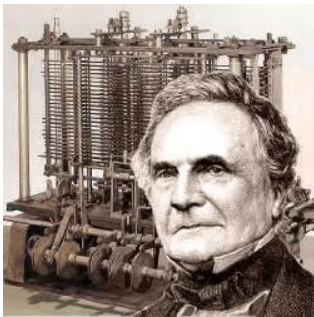
# Awards in Computer Architecture



**Eckert-Mauchly: IEEE Computer Society and ACM:.....** “For extraordinary leadership in building a world class computer architecture research center, for seminal contributions in the areas of vector computing and multithreading, and for pioneering basic new approaches to instruction-level parallelism.” June 2007



**Seymour Cray: IEEE Computer Society:.....** “In recognition of seminal contributions to vector, out-of-order, multithreaded, and VLIW architectures.” November 2015

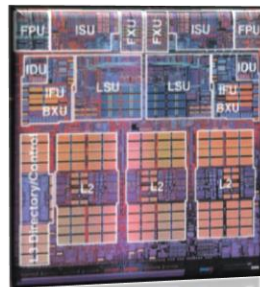


**Charles Babbage: IEEE Computer Society: .....**“For contributions to parallel computation through brilliant technical work, mentoring PhD students, and building an incredibly productive European research environment.”. April, 2017

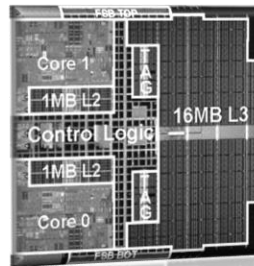
# The MultiCore Era

## Moore's Law + Memory Wall + Power Wall

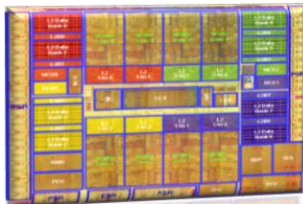
### Chip MultiProcessors (CMPs)



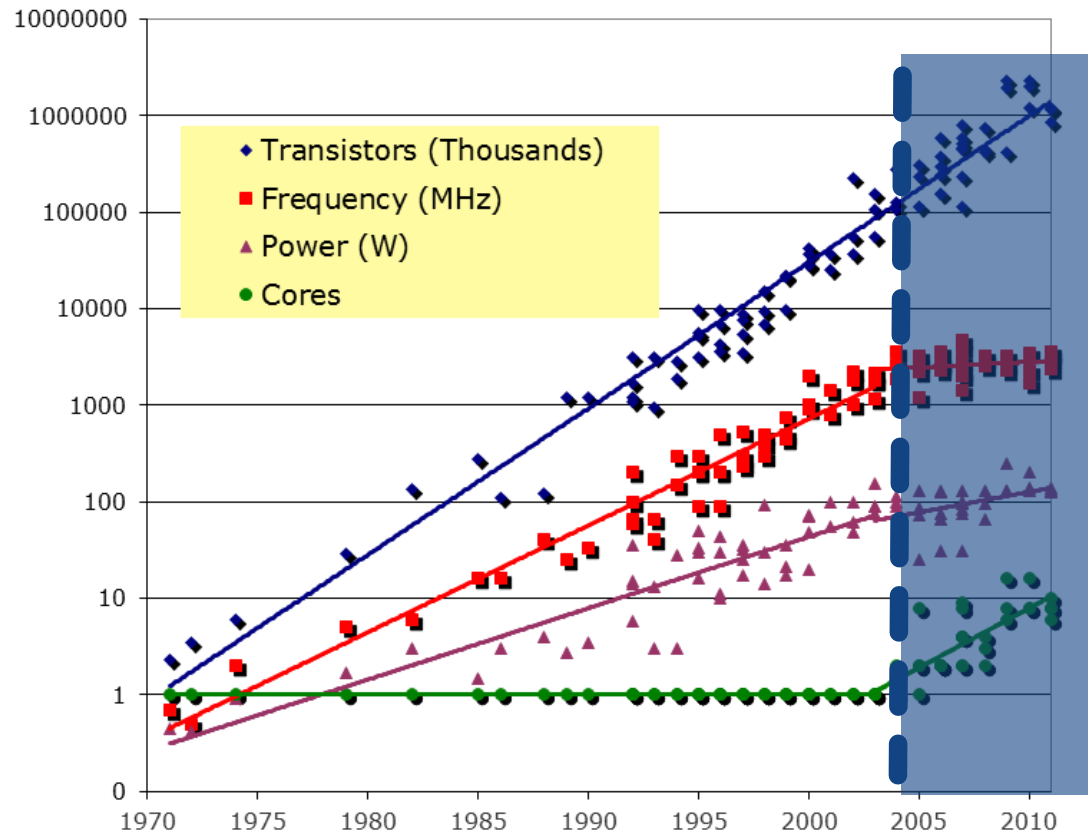
POWER4  
(2001)



Intel Xeon  
7100 (2006)



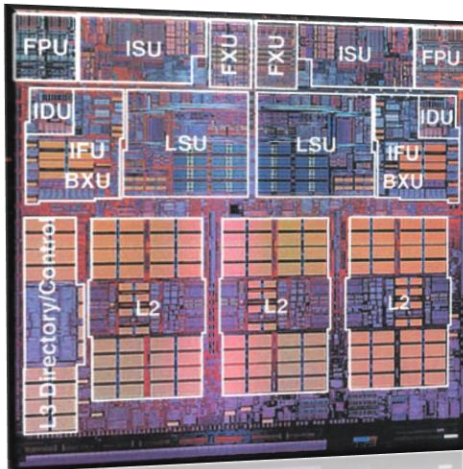
UltraSPARC T2  
(2007)



# How Multicores Were Designed at the Beginning?

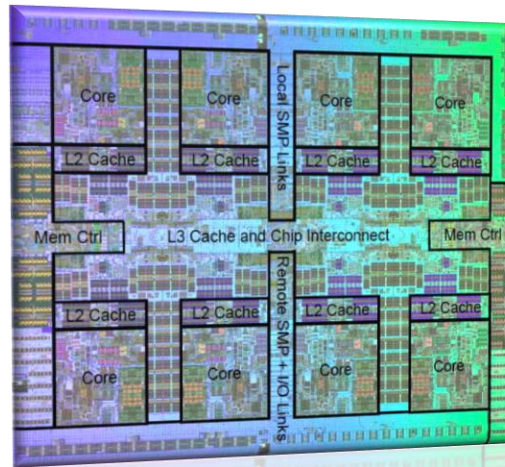
## IBM Power4 (2001)

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



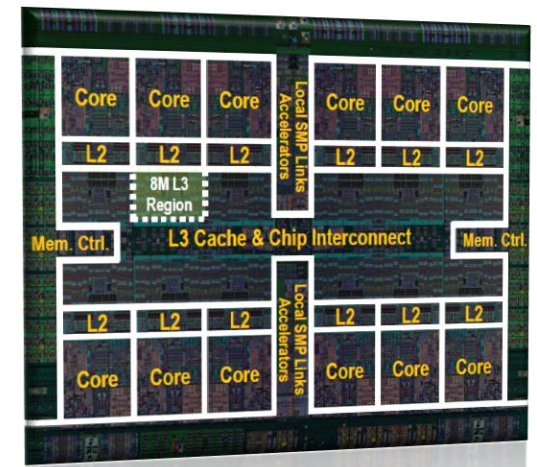
## IBM Power7 (2010)

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



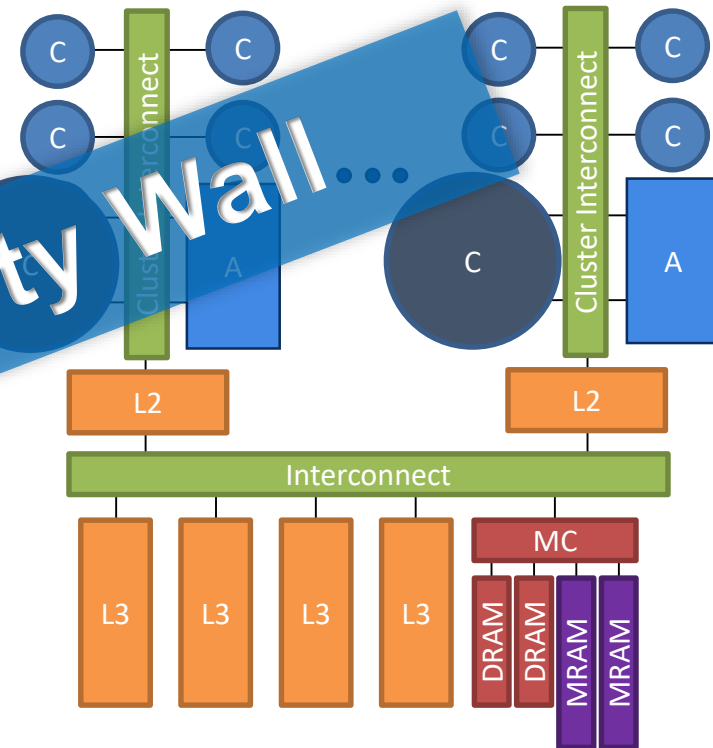
## IBM Power8 (2014)

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



# How To Parallelize Future Applications?

- From sequential to parallel codes
- Efficient runs on manycore processors implies handling:
  - Massive amount of cores and available parallelism
  - Heterogeneous systems
    - Same or multiple ISAs
    - Accelerators, specialization
  - Deep and heterogeneous memory hierarchy
    - Non-Uniform Memory Access (NUMA)
    - Multiple address spaces
  - Stringent energy budget
  - Load Balancing

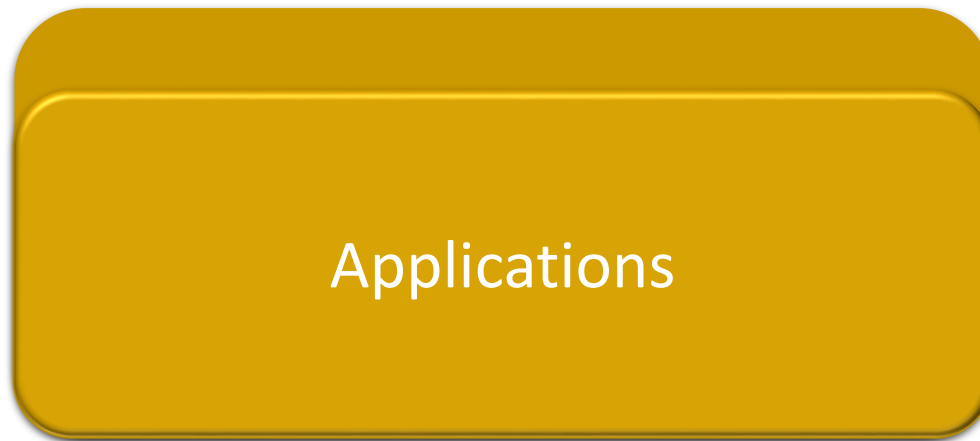


**A Really Fuzzy Space**



# Living in the Programming Revolution

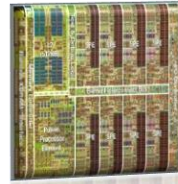
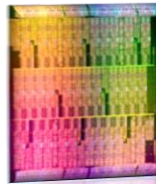
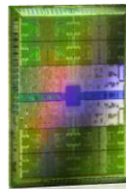
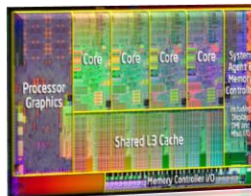
Multicores made the interface to leak...



Parallel application logic + Platform specificities



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



# Vision in the Programming Revolution

Need to decouple again

Applications

Application logic  
Arch. independent

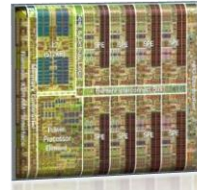
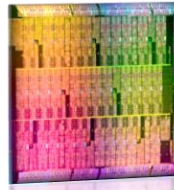
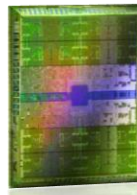
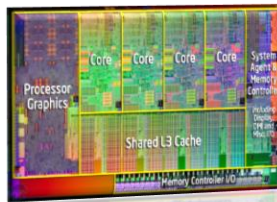
PM: High-level, clean, abstract interface

General purpose  
Single address space

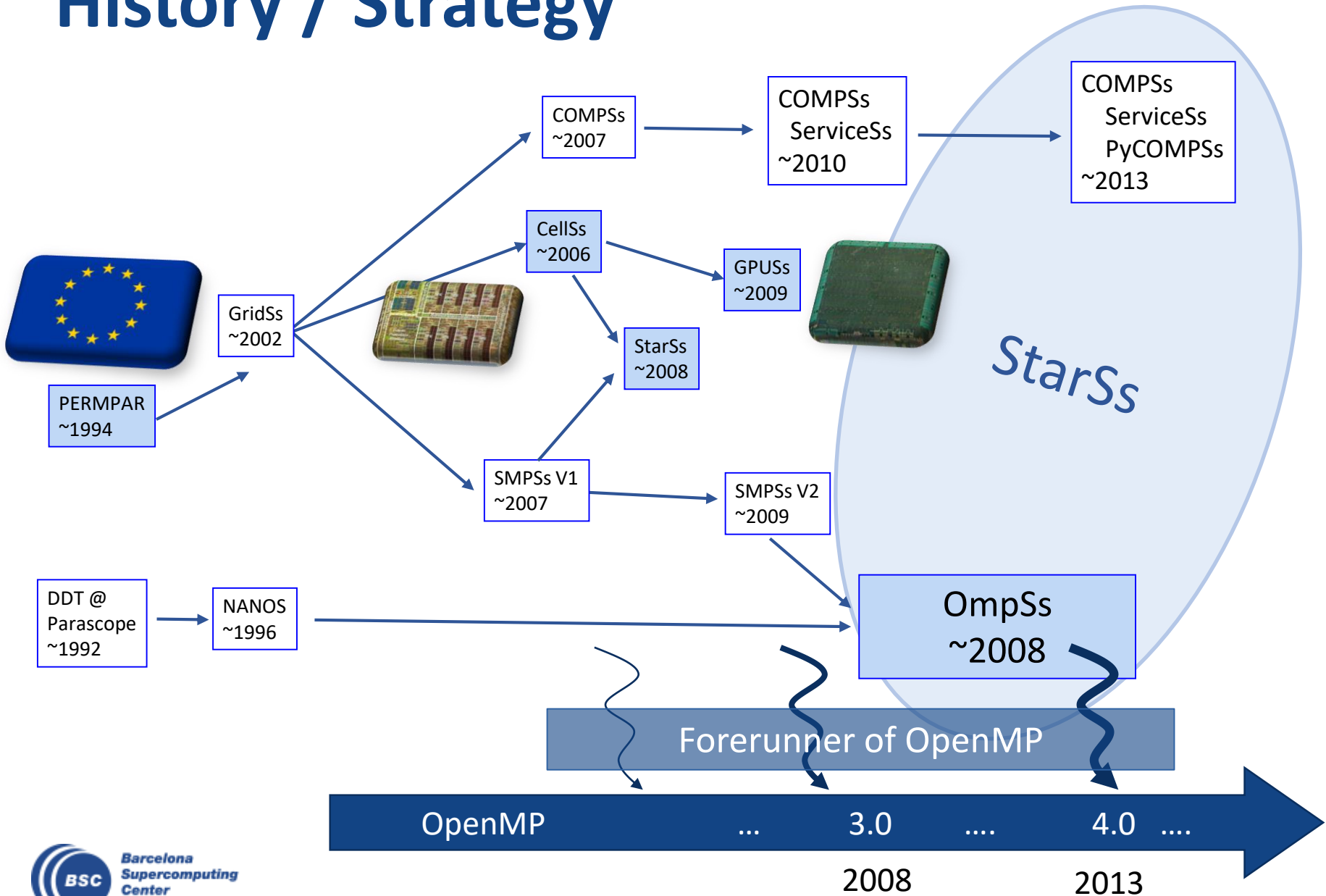
Power to the runtime

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

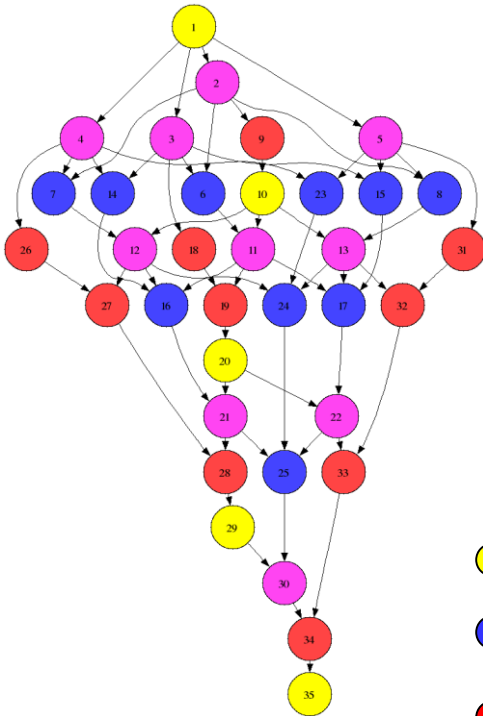
ISA / API



# History / Strategy



# OmpSs: data-flow execution of sequential programs



```
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]);
        }
    }
}
```

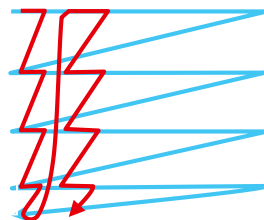
```
#pragma omp task inout ([TS][TS]A)
void spotrf (float *A);
#pragma omp task input ([TS][TS]A) inout ([TS][TS]C)
void ssyrk (float *A, float *C);
#pragma omp task input ([TS][TS]A,[TS][TS]B) inout ([TS][TS]C)
void sgemm (float *A, float *B, float *C);
#pragma omp task input ([TS][TS]T) inout ([TS][TS]B)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

Decouple how we write applications from how they are executed

Write

Clean offloading to hide architectural complexities

Execute



# 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]);
```

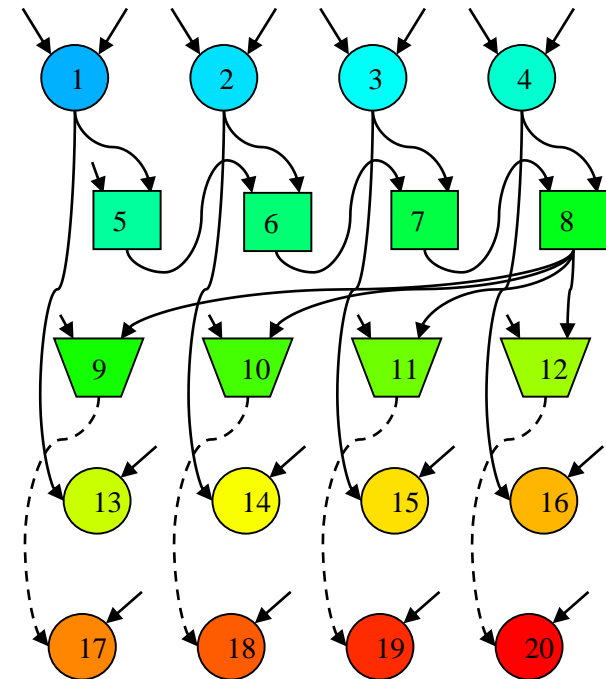


```
#pragma css task input(A) inout(sum)
```

```
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

# ... and Executed in a Data-Flow Model

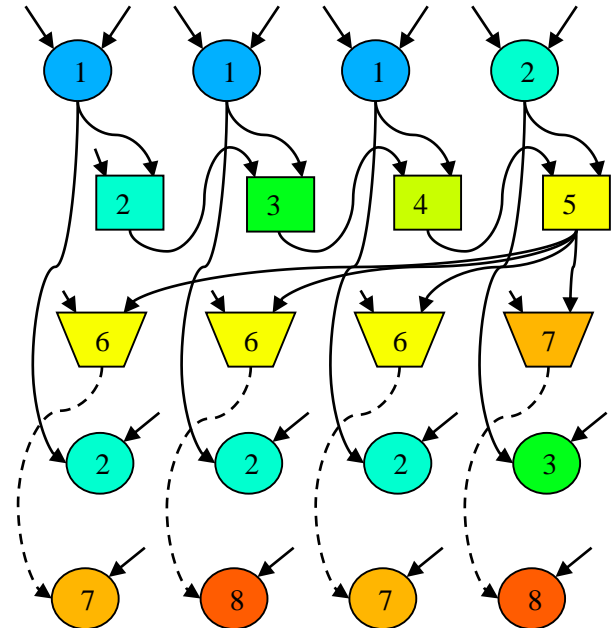
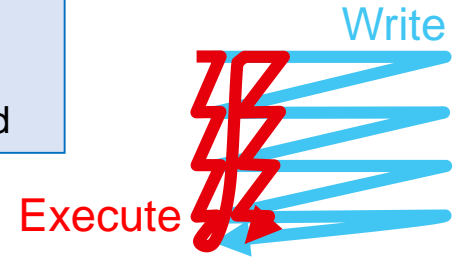
```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
           float C[BS]);

#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);
```

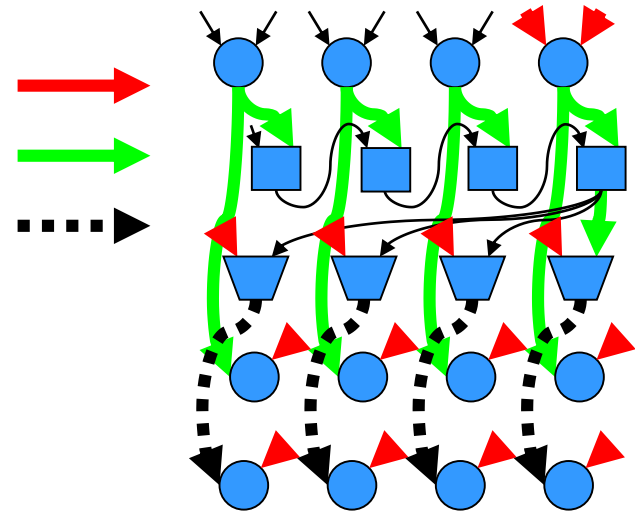
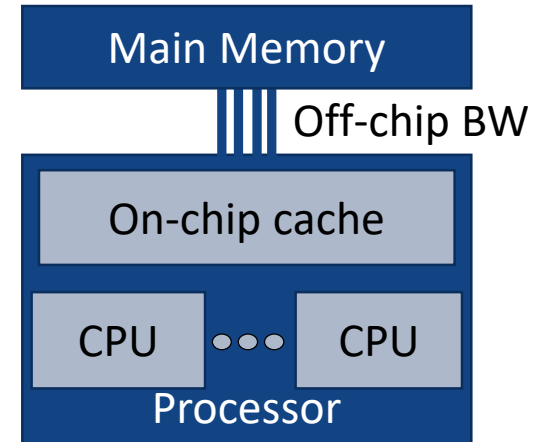
```
for (i=0; i<N; i+=BS)           // C=A+B
    vadd3 (&A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)           //sum(C[i])
    accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)           // B=sum*A
    scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)           // A=C+D
    vadd3 (&C[i], &D[i],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

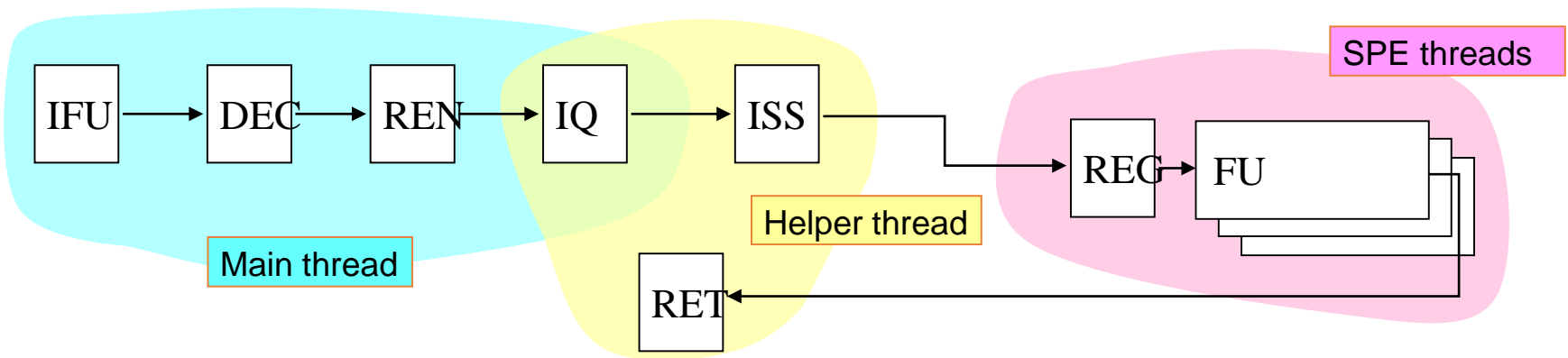
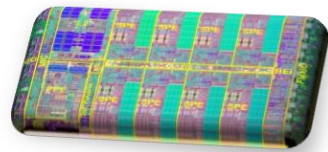
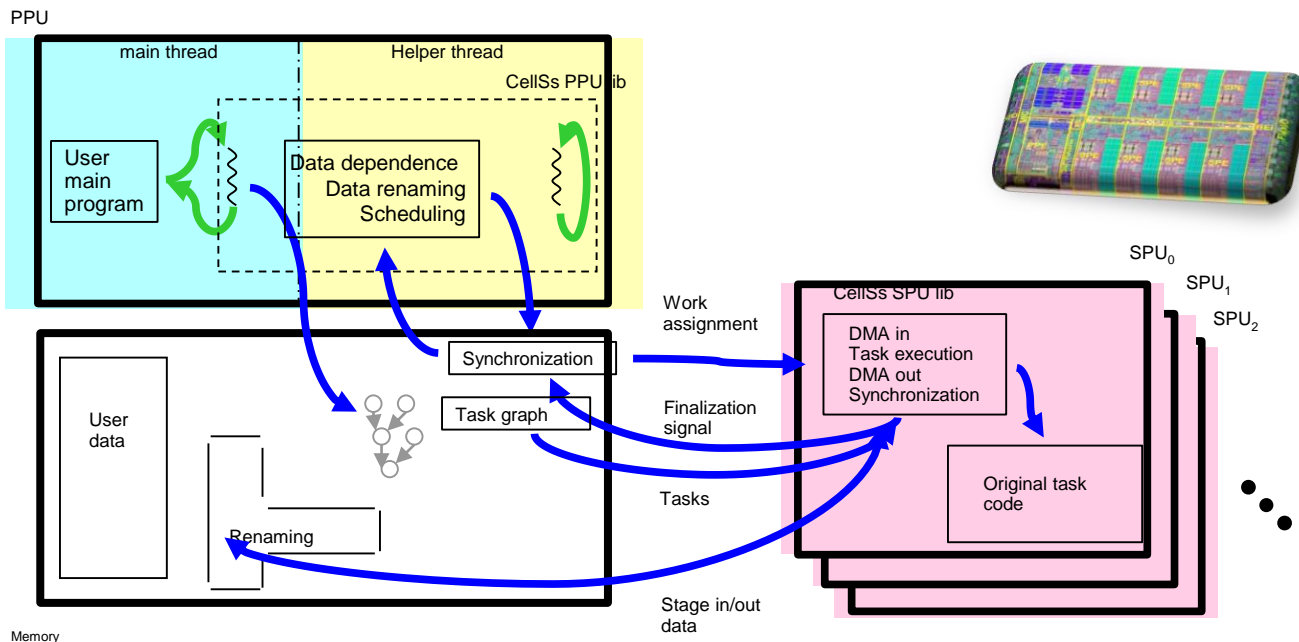


# OmpSs: Potential of Data Access Info

- 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



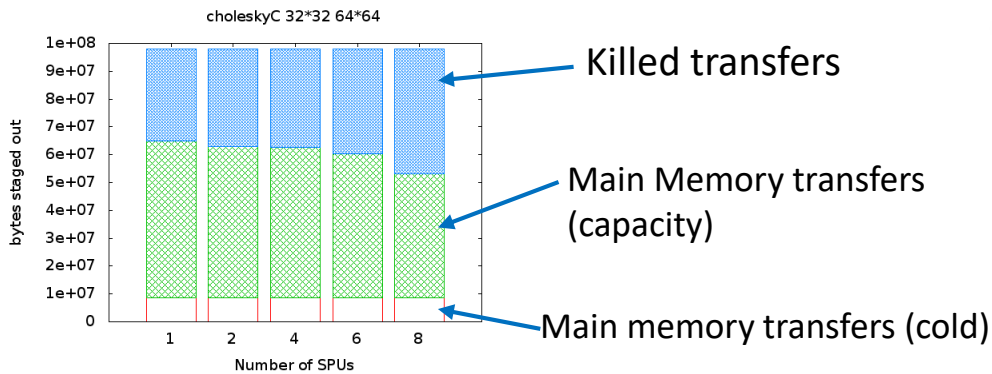
# CellSs implementation



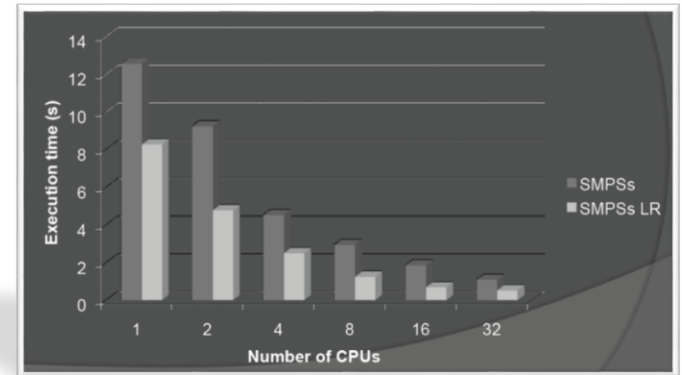


# Renaming @ Cell

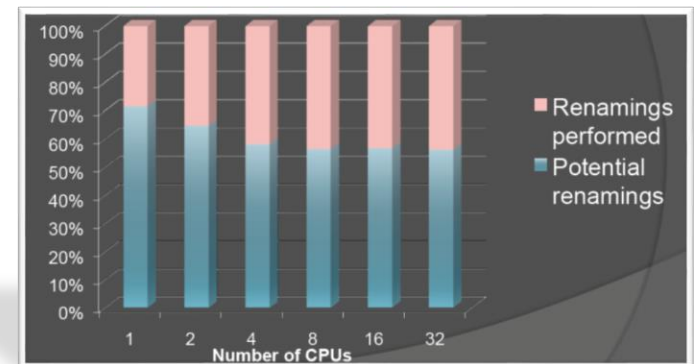
- Experiments on the CellSs (predecessor of OmpSs)
  - **Renaming** to avoid anti-dependences
    - Eager (similarly done **at SS designs**)
      - At task instantiation time
    - Lazy (similar to **virtual registers**)
      - Just before task execution



SMPSs: Stream benchmark reduction in execution time



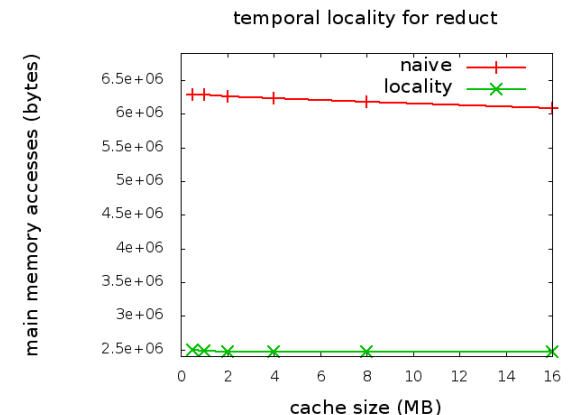
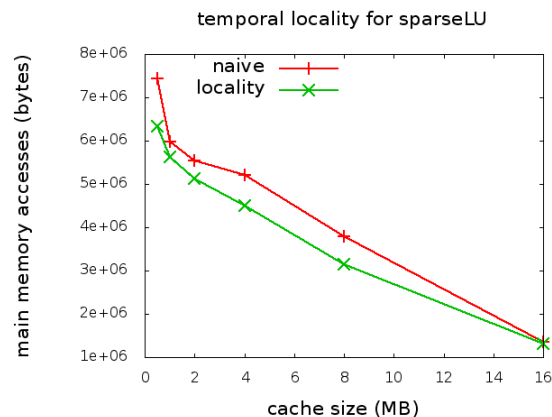
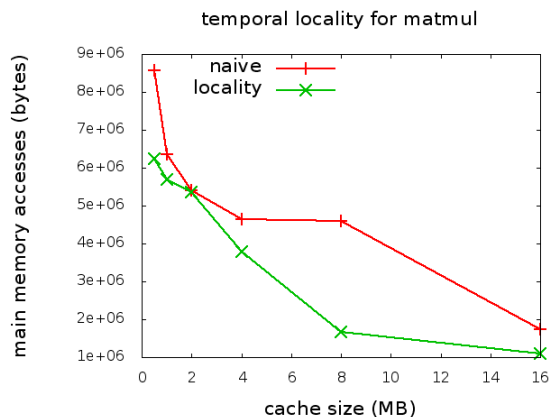
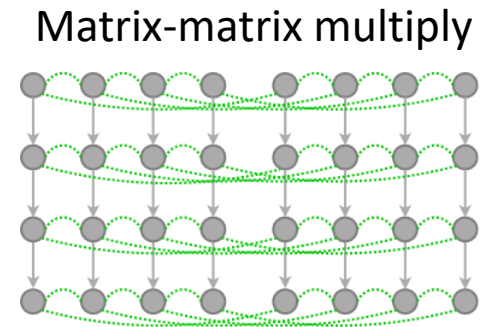
SMPSs: Jacobi reduction in # renamings



P. Bellens, et al, "CellSs: Scheduling Techniques to Better Exploit Memory Hierarchy" Sci. Prog. 2009

# Data Reuse @ Cell

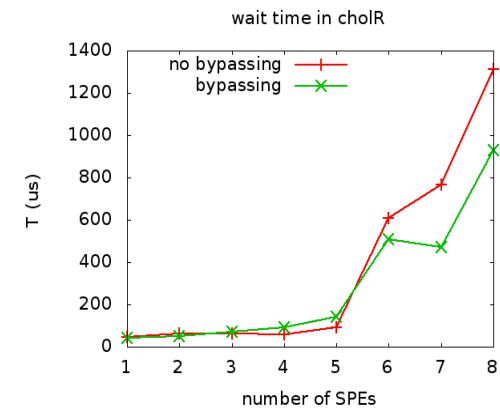
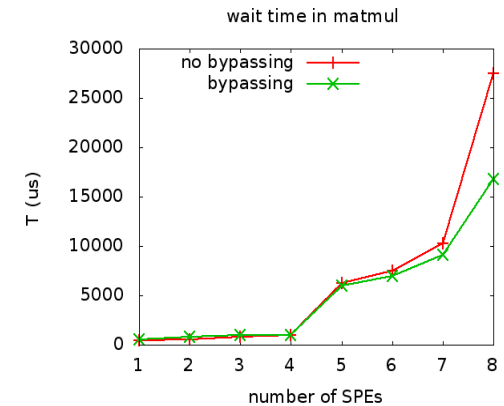
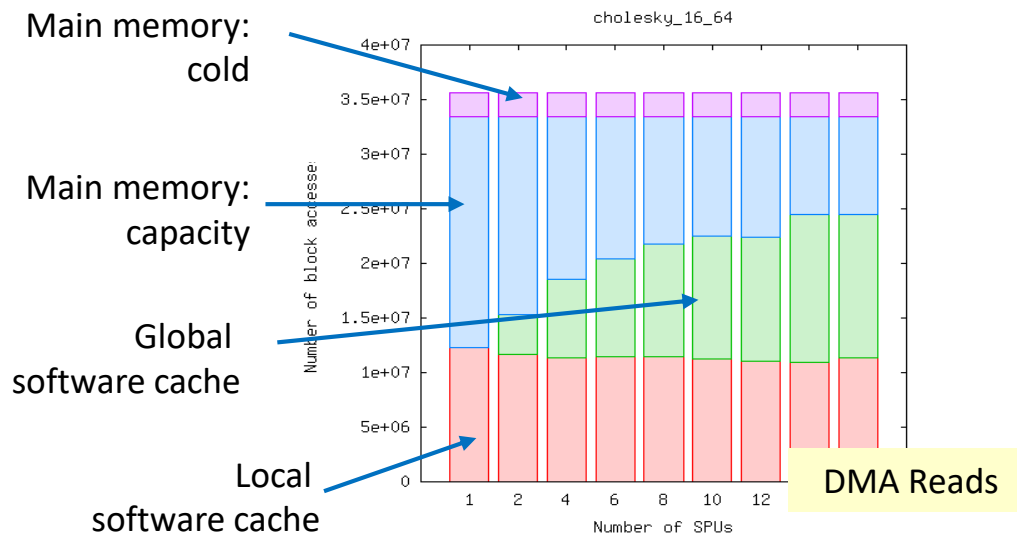
- Experiments on the CellS
  - Data Reuse
  - Locality arcs in dependence graph
- Good locality but high overhead → no time improvement



P. Bellens, et al, "CellS: Scheduling Techniques to Better Exploit Memory Hierarchy" Sci. Prog. 2009

# Reducing Data Movement @ Cell

- Experiments on the CellSs (predecessor of OmpSs)
  - Bypassing / [global software cache](#)
  - Distributed implementation
    - @each SPE
    - Using object descriptors managed atomically with specific hardware support (line level LL-SC)



# GPUSs implementation

- Architecture implications

- Large local store O(GB) → large task granularity ← Good
- Data transfers: Slow, non overlapped ← Bad

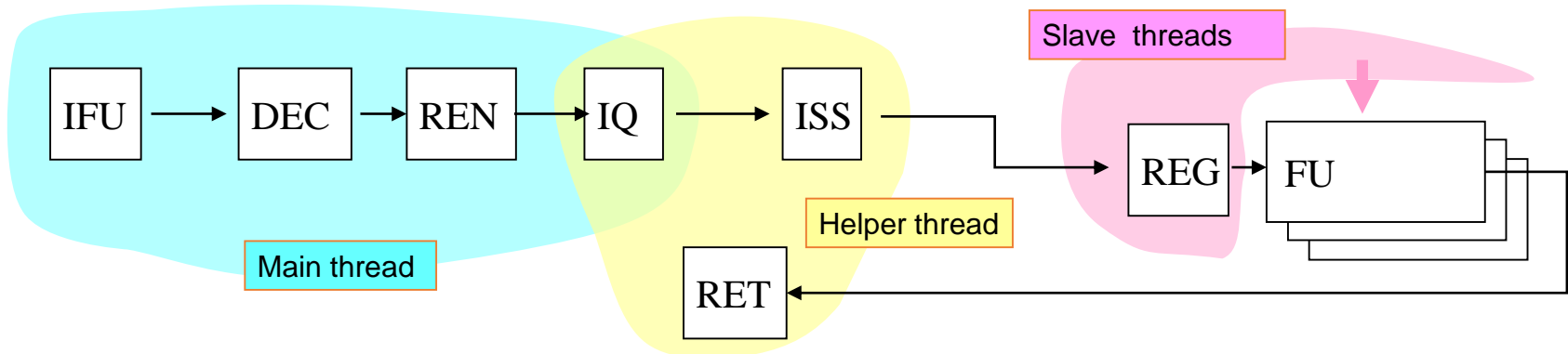
- Cache management

- Write-through
- Write-back



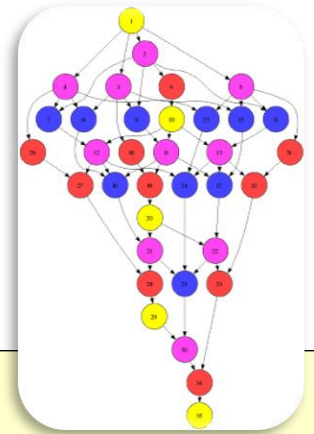
- Run time implementation

- Powerful main processor and multiple cores
- Dumb accelerator (not able to perform data transfers, implement software cache,...)



# Prefetching @ multiple GPUs

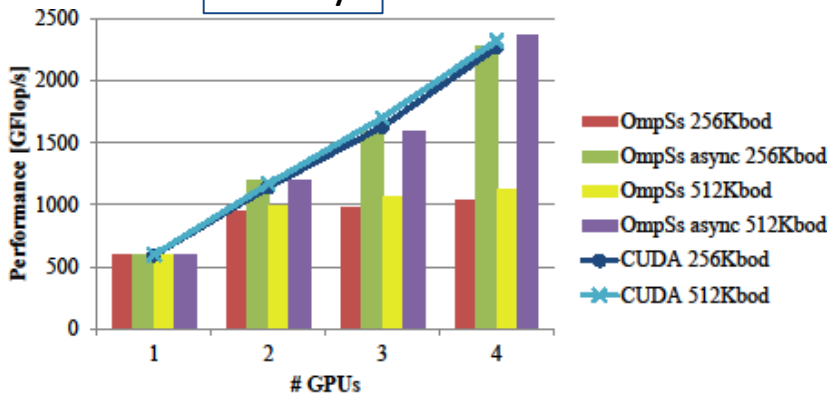
- Improvements in runtime mechanisms (OmpSs + CUDA)
  - Use of multiple streams
  - High asynchrony and overlap (transfers and kernels)
  - Overlap kernels
  - Take overheads out of the critical path
- Improvement in schedulers
  - Late binding of locality aware decisions
  - Propagate priorities



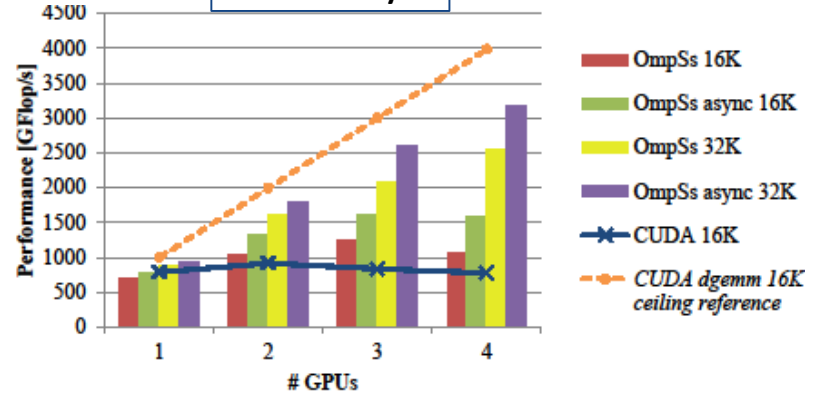
```

void Cholesky( float *A[NT][NT] ) {
  int i, j, k;
  for (k=0; k<NT; k++) {
    #pragma omp task inout (A[k][k])
    ● spotrf (A[k][k]) ;
    for (i=k+1; i<NT; i++) {
      #pragma omp task in (A[k][k]) inout (A[k][i])
      ● strsm (A[k][k], A[k][i]) ;
    }
    for (i=k+1; i<NT; i++) {
      for (j=k+1; j<i; j++) {
        #pragma omp task in (A[k][i], A[k][j]) inout (A[j][i])
        ● sgemm (A[k][i], A[k][j], A[j][i]) ;
      }
      #pragma omp task in (A[k][i]) inout (A[i][i])
      ● ssyrk (A[k][i], A[i][i]) ;
    }
  }
}
    
```

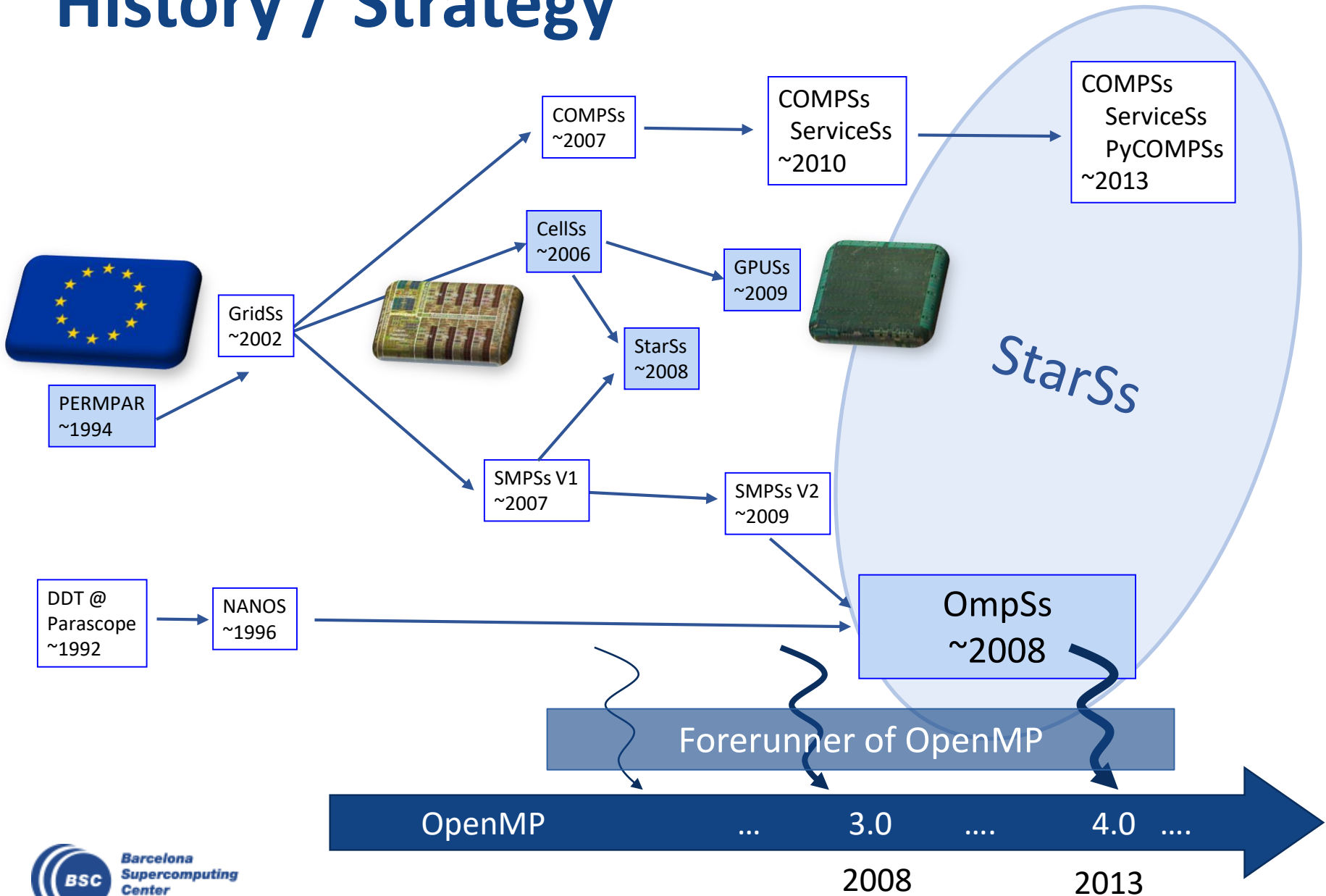
Nbody



Cholesky

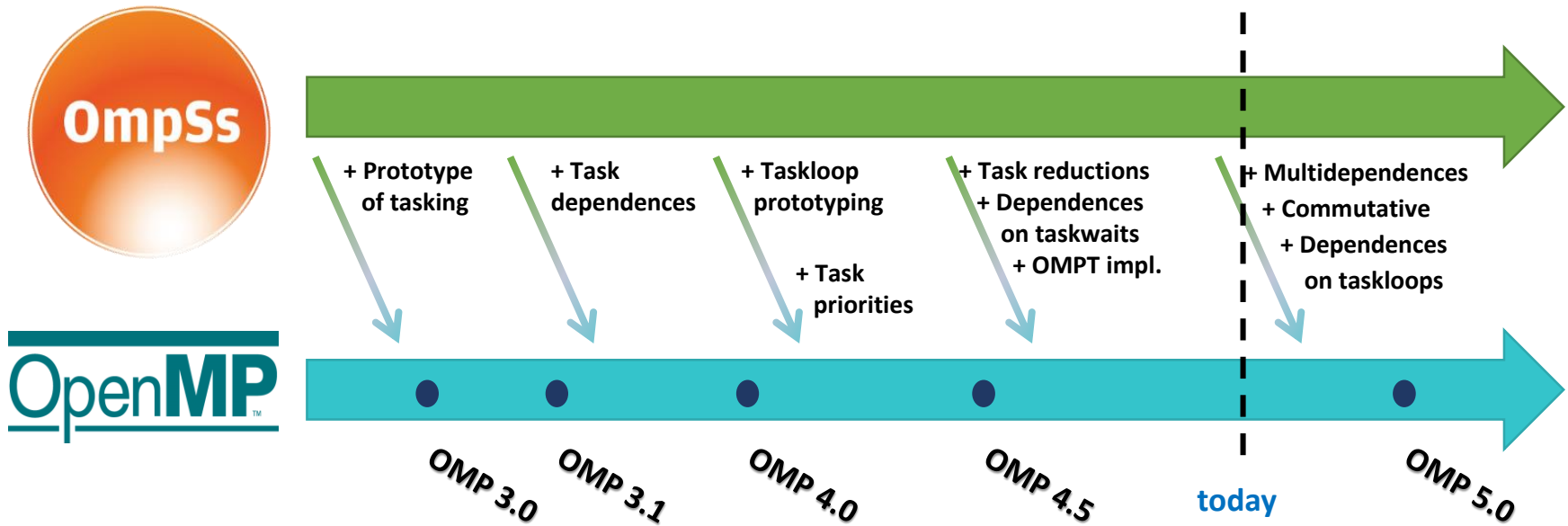


# History / Strategy



# OmpSs

## A forerunner for OpenMP



# Runtime Aware Architectures

« The runtime **drives** the hardware design

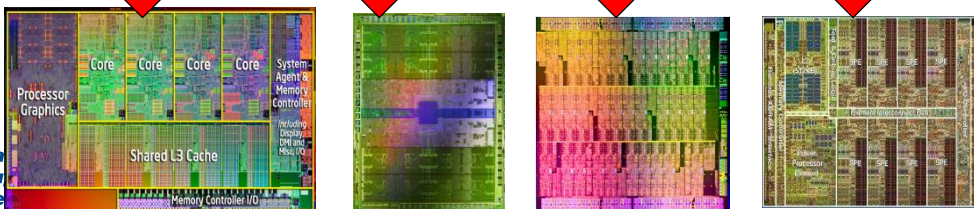
– Tight collaboration between software and hardware layers

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



# Superscalar vision at Multicore level

## Superscalar World

Out-of-Order, Kilo-Instruction Processor,  
Distant Parallelism

Branch Predictor, Speculation

Fuzzy Computation

Dual Data Cache, Sack for VLIW

Register Renaming, Virtual Regs

Cache Reuse, Prefetching, Victim Cache

In-memory Computation

Accelerators, Different ISA's, SMT

Critical Path Exploitation

Resilience

**Memory Wall**

**Power Wall**

**Programmability  
Wall**

**Resilience Wall**

## Multicore World

Task-based, Data-flow Graph, Dynamic  
Parallelism

Tasks Output Prediction,  
Speculation

Hybrid Memory Hierarchy, NVM

Late Task Memory Allocation

Data Reuse, Prefetching

In-memory FU's

Heterogeneity of Tasks and HW

Task-criticality

Resilience

**Load Balancing and Scheduling**

Interconnection Network

Data Movement

# RoMoL Research Lines

- Management of hybrid memory hierarchies with scratchpad memories (ISCA'15, PACT'15) and stacked DRAMs (ICS'18)
- Runtime Exploitation of Data Locality (PACT'16, TPDS'18)
- Exploiting the Task Dependency Graph (TDG) to reduce data movements (ICS'18)
- Architectural Support for Task-dependence Management (IPDPS'17, HPCA'18)
- Vector Extensions to Optimize DBMS (Micro'12, HPCA'15, ISCA'16)
- Criticality-aware task scheduling (ICS'15) and acceleration (IPDPS'16)
- Approximate Task Memoization (IPDPS'17)
- Dealing with Variation due to Hardware Manufacturing (ICS'16)

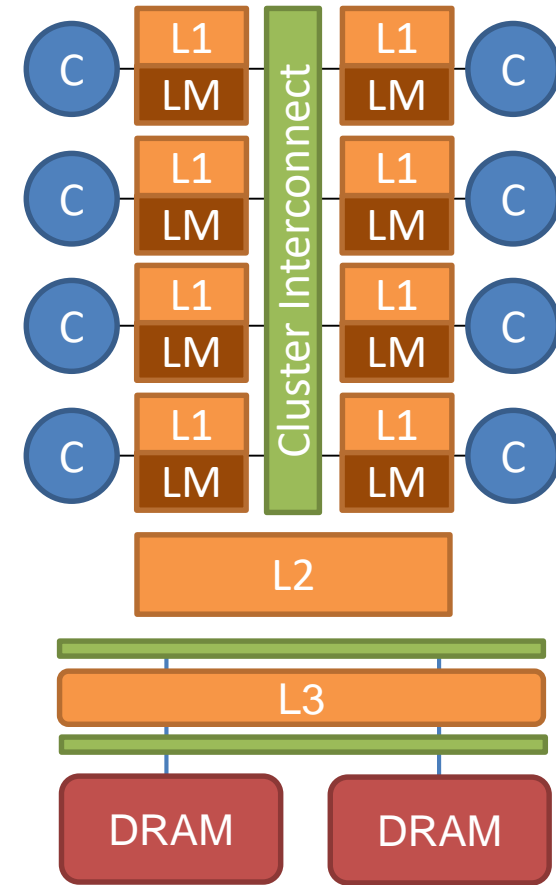
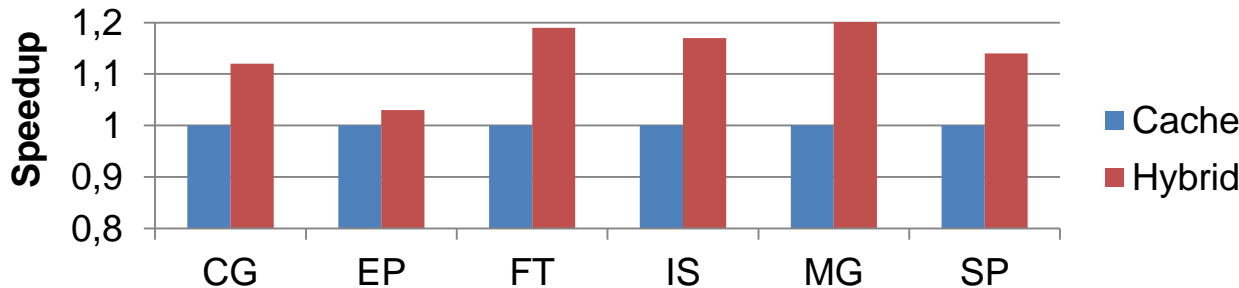
# Runtime Aware Architectures (RAA)

- ⌘ 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

# Transparent Management of Local Memories

- Hybrid memory hierarchy
  - L1 cache + Local memories (LM)
- More difficult to manage, but
  - More energy efficient
  - Less coherence traffic
- LM Management in OpenMP (SC'12, ISCA'15)
  - Strided accesses served by the LM
  - Irregular accesses served by the L1 cache
  - HW support for coherence and consistency



LI. Alvarez et al. Hardware-Software Coherence Protocol for the Coexistence of Caches and Local Memories. SC 2012.

LI. Alvarez et al. Coherence Protocol for Transparent Management of Scratchpad Memories in shared Memory Manycore Architectures. ISCA 2015.



# Transparent Management of Local Memories

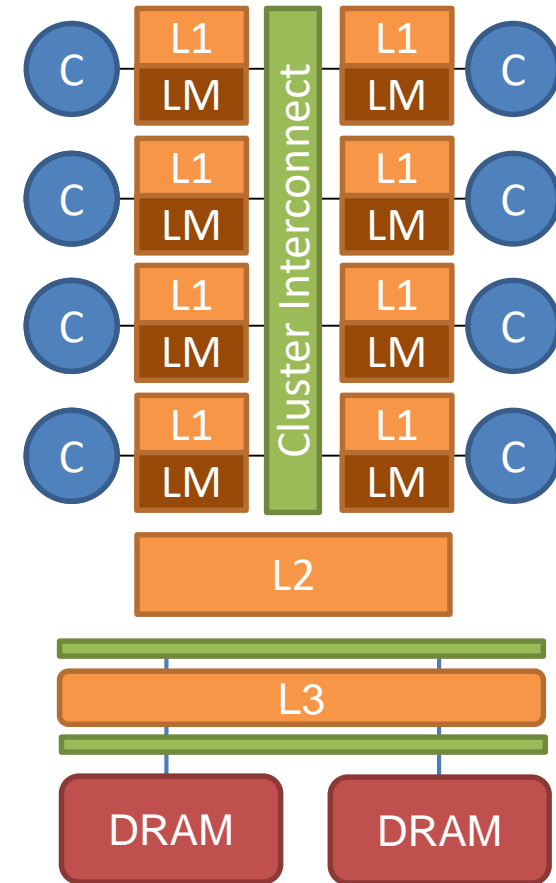
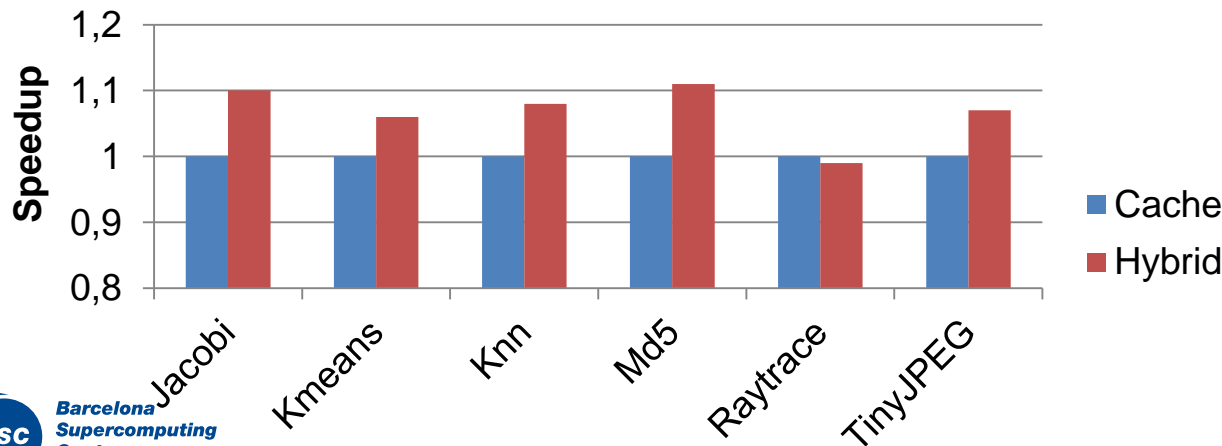
## LM Management in Task-based Programming Models (PACT'15)

- Inputs and outputs mapped to the LMs
- Runtime manages DMA transfers

8.7% speedup in execution time

14% reduction in power

20% reduction in network-on-chip traffic

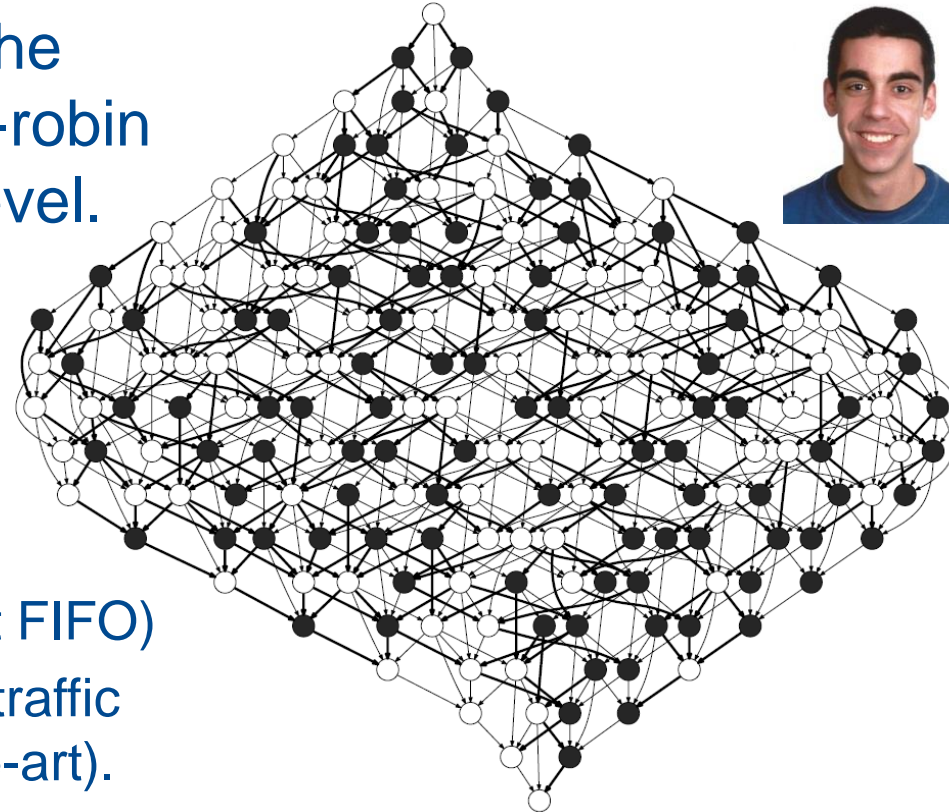


LI. Alvarez et al.  
Runtime-Guided  
Management of Hybrid  
Memory Hierarchies in  
Multicore Architectures.  
PACT 2015.

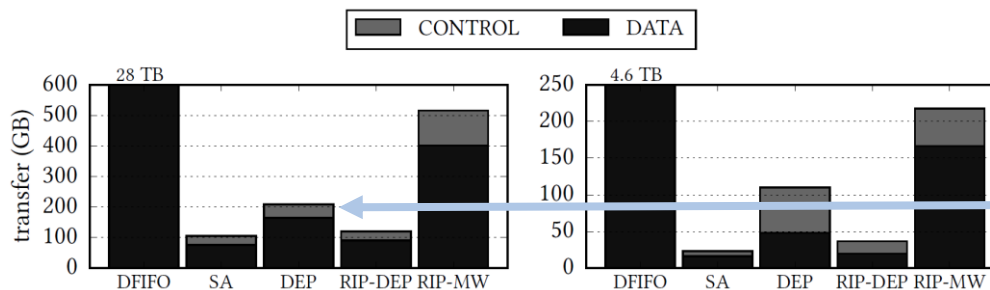


# Exploiting the Task Dependency Graph (TDG) to Reduce Coherence Traffic

- ❧ To reduce coherence traffic, the state-of-the-art applies round-robin mechanisms at the runtime level.
- ❧ Exploiting the information contained at the TDG level is effective to
  - improve performance (3.16x wrt FIFO)
  - dramatically reduce coherence traffic (2.26x reduction wrt state-of-the-art).



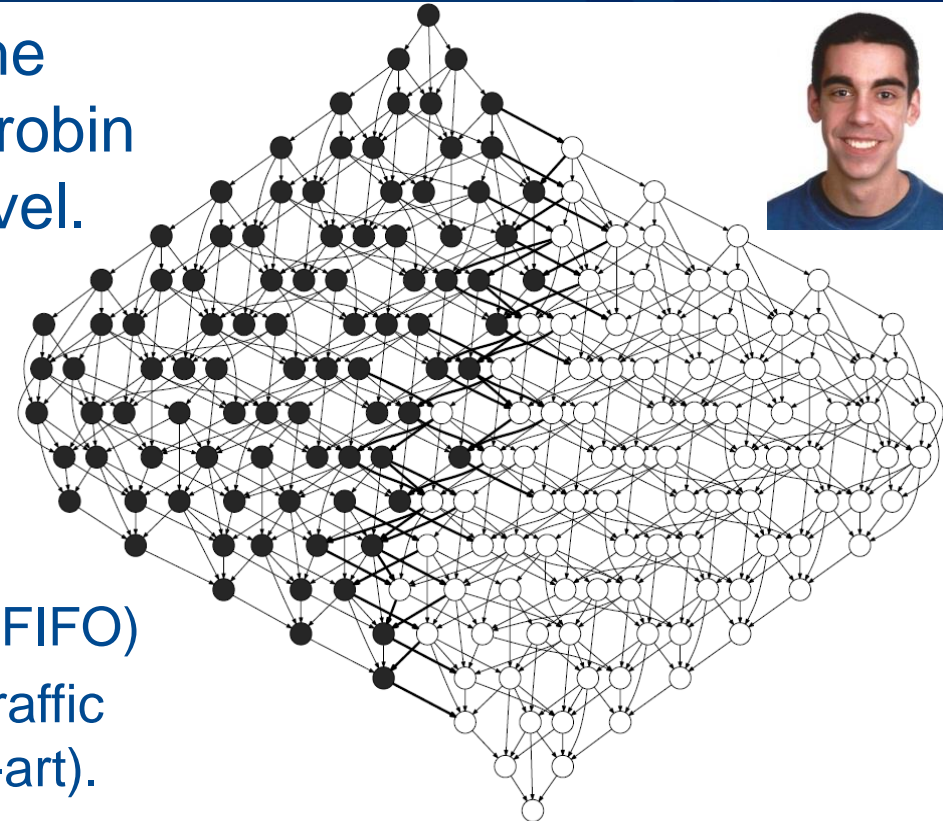
State-of-the-art Partition (DEP)  
Gauss-Seidel TDG



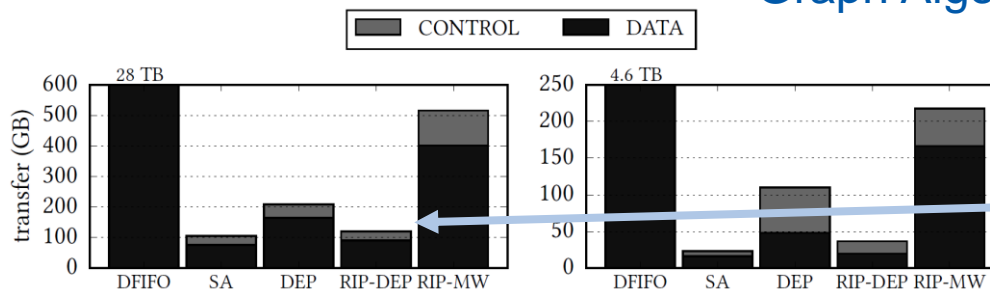
DEP requires ~200GB  
of data transfer across a  
288 cores system

# Exploiting the Task Dependency Graph (TDG) to Reduce Coherence Traffic

- ❧ To reduce coherence traffic, the state-of-the-art applies round-robin mechanisms at the runtime level.
- ❧ Exploiting the information contained at the TDG level is effective to
  - improve performance (3.16x wrt FIFO)
  - dramatically reduce coherence traffic (2.26x reduction wrt state-of-the-art).



Graph Algorithms-Driven Partition (RIP-DEP)  
Gauss-Seidel TDG



(a) Gauss-Seidel

(b) Integral histogram

RIP-DEP requires ~90GB of data transfer across a 288 cores system

# Transparent Management of Stacked DRAMs

## ☞ Heterogeneous memory system

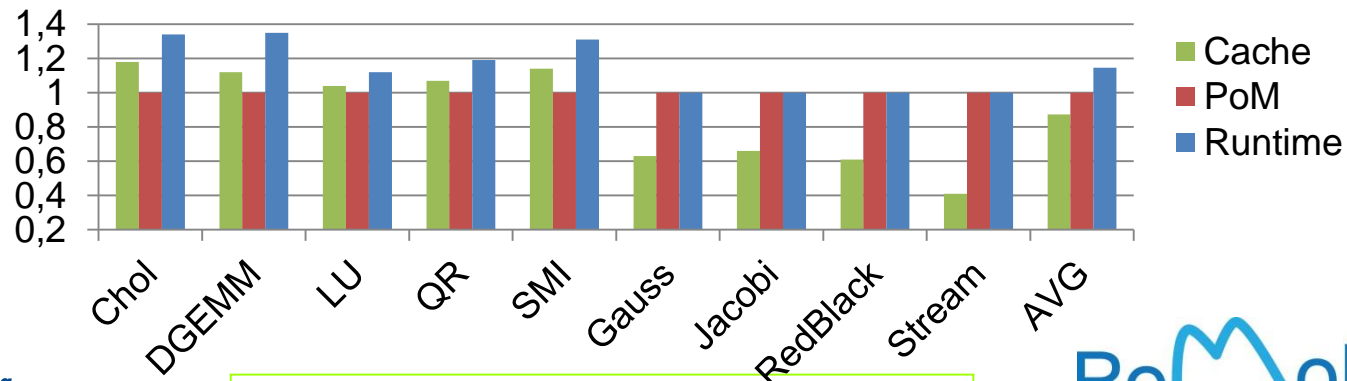
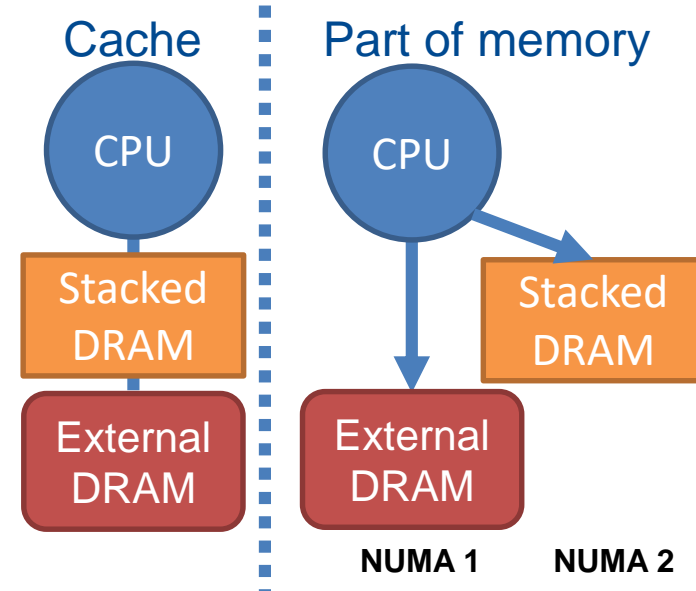
- 3D stacked HBM + off-chip DDR4

## ☞ Very high bandwidth, but

- Difficult to manage
- Part of memory (PoM) or cache?

## ☞ Runtime-managed stacked DRAM

- Map task data to the stacked DRAM
- Parallelize data copies to reduce copy overheads
- Reuse-aware bypass to avoid unworthy copies
- 14% average performance benefits on an Intel Knight's Landing





# 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

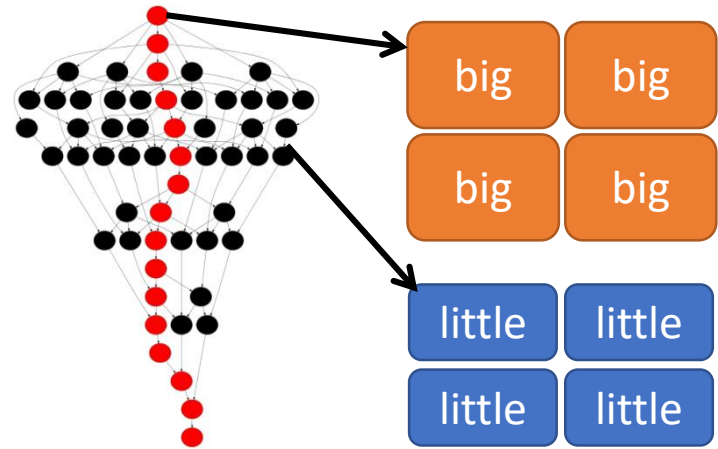
- ⌘ Heterogeneity of tasks and Hardware
  - Critical path exploitation
  - Manufacturing Variability
- ⌘ Management of shared resources

Power Wall

# OmpSs in Heterogeneous Systems

## Heterogeneous systems

- Big-little processors
- Accelerators
- Hard to program

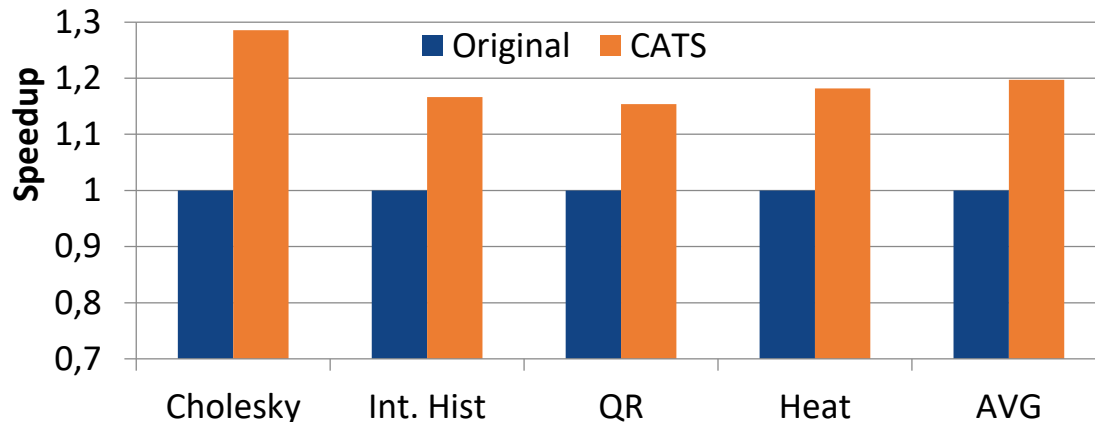


## Task-based programming models can adapt to these scenarios

- Detect tasks in the critical path and run them in fast cores
- Non-critical tasks can run in slower cores
- Assign tasks to the most energy-efficient HW component
- Runtime takes care of balancing the load
- Same performance with less power consumption

# Criticality-Aware Task Scheduler

- CATS on a big.LITTLE processor (ICS'15)
  - 4 Cortex A15 @ 2GHz
  - 4 Cortex A7 @ 1.4GHz



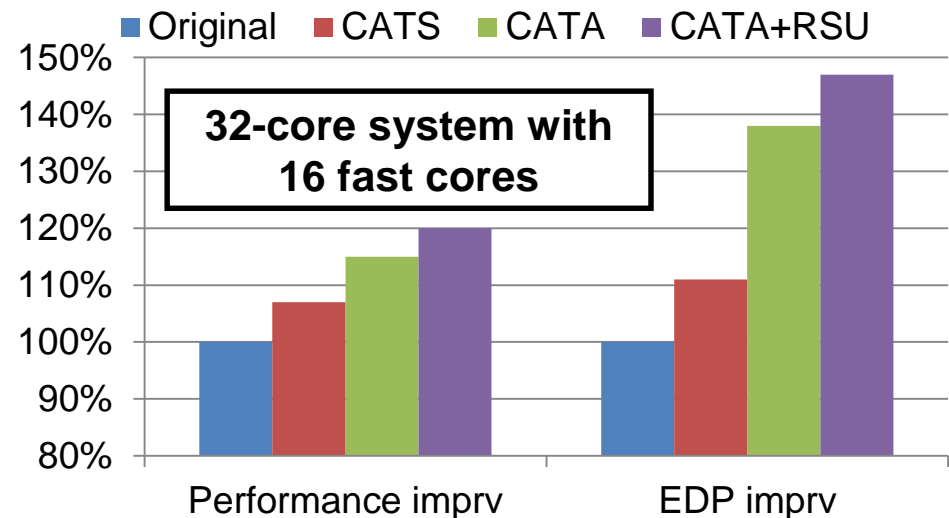
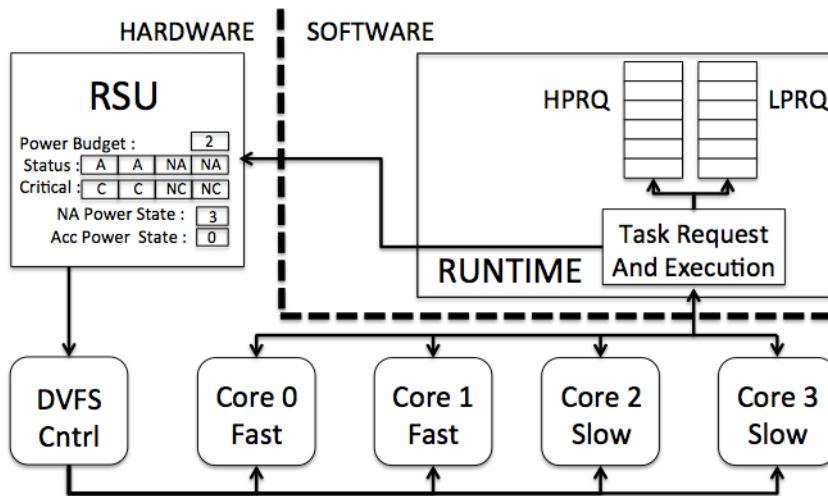
- Effectively solves the problem of *blind assignment* of tasks
  - Higher speedups for double precision-intensive benchmarks
- But still suffers from *priority inversion* and *static assignment*

# Criticality-Aware Task Acceleration



## ⌘ CATA: accelerating critical tasks (IPDPS'16)

- Runtime drives per-core DVFS reconfigurations meeting a global power budget
- Solves *priority inversion* and *static assignment* issues
- Reconfiguration overhead grows with the number of cores
  - Hardware Runtime Support Unit (RSU) reconfigures DVFS



# 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
  - Manufacturing variability
- ⌘ Management of shared resources

## Power Wall

## Programmability Wall

- ⌘ Hardware acceleration of the runtime system
  - Task dependency graph management
- ⌘ Task Memoization and Approximation

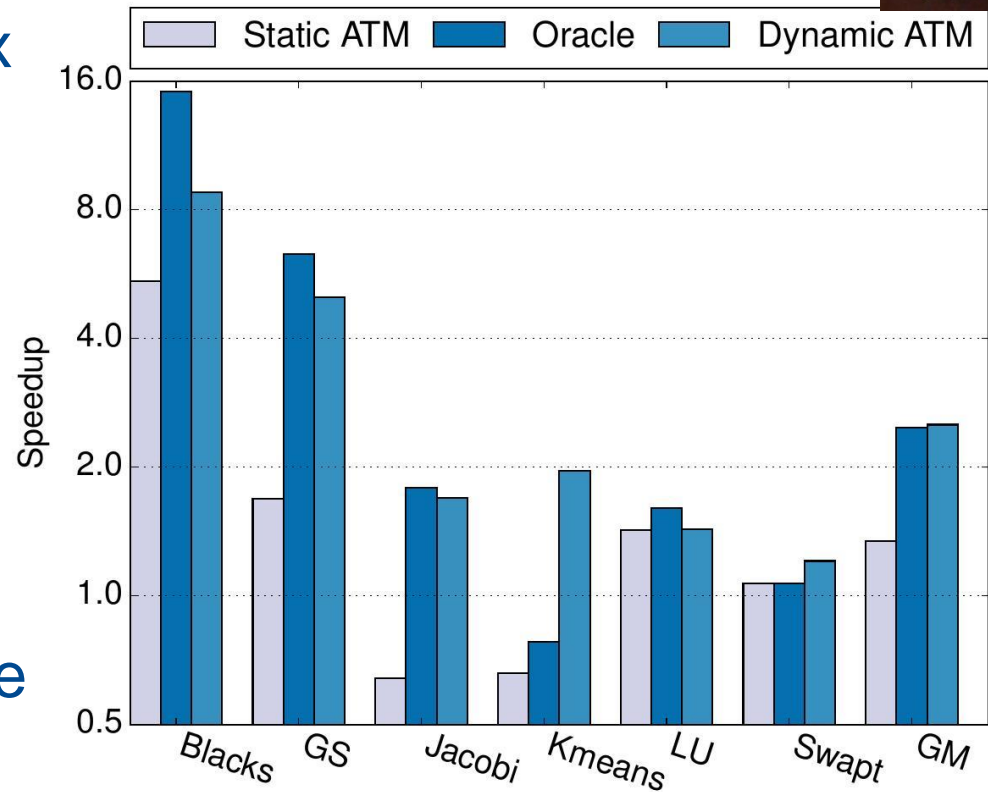
## Resilience Wall

- ⌘ Task-based checkpointing
- ⌘ Algorithmic-based fault tolerance

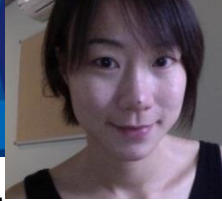
# Approximate Task Memoization (ATM)



- ATM aims to eliminate redundant tasks (IPDPS'17)
- ATM detects correlations between task inputs and outputs to memoize similar tasks
  - Static ATM** achieves 1.4x average speedup when only applying memoization techniques
  - With task approximation, **Dynamic ATM** achieves 2.5x average speedup with an average 0.7% accuracy loss, competitive with an off-line **Oracle** approach



# TaskSuperscalar (TaskSs) Pipeline



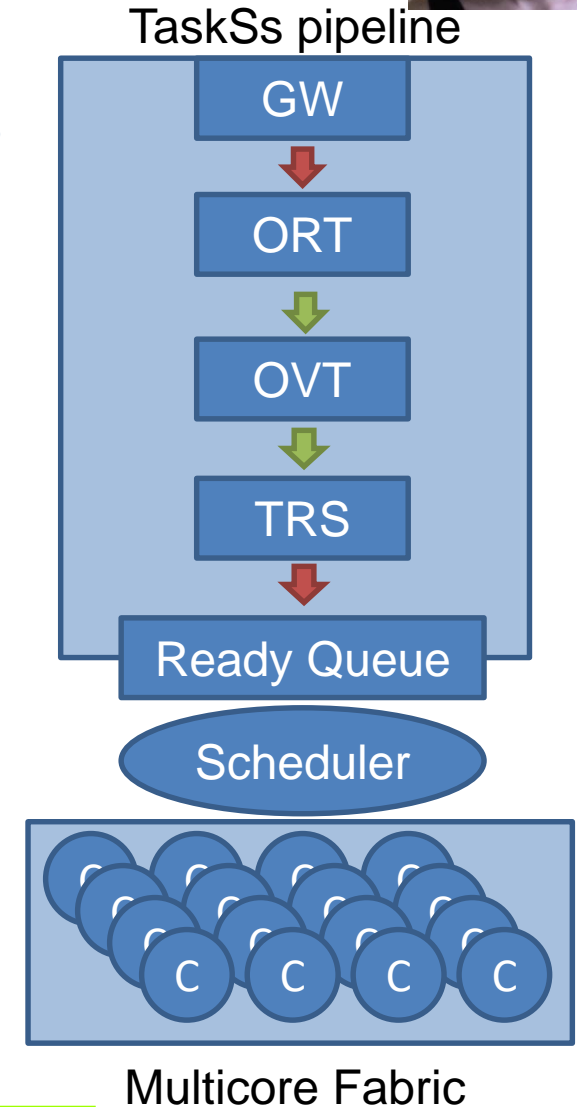
## Hardware design for a distributed task superscalar pipeline frontend (MICRO'10)

- Can be embedded into any manycore fabric
- Drive hundreds of threads
- Work windows of thousands of tasks
- Fine grain task parallelism

## TaskSs components:

- Gateway (GW): Allocate resources for task meta-data
- Object Renaming Table (ORT)
  - Map memory objects to producer tasks
- Object Versioning Table (OVT)
  - Maintain multiple object versions
- Task Reservation Stations (TRS)
  - Store and track task in-flight meta-data

## Implementing TaskSs @ Xilinx Zynq (ISPASS'16, IPDPS'17)

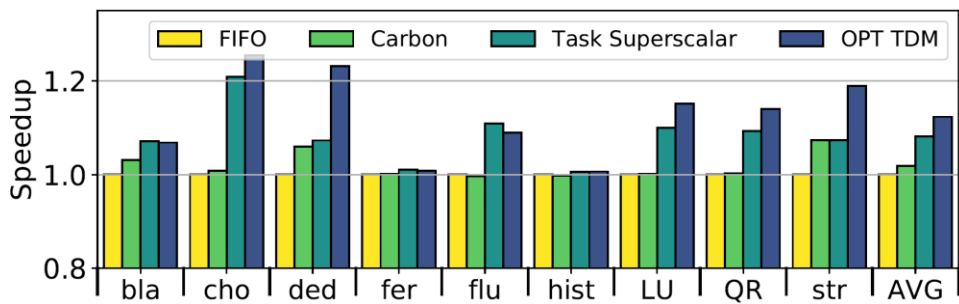
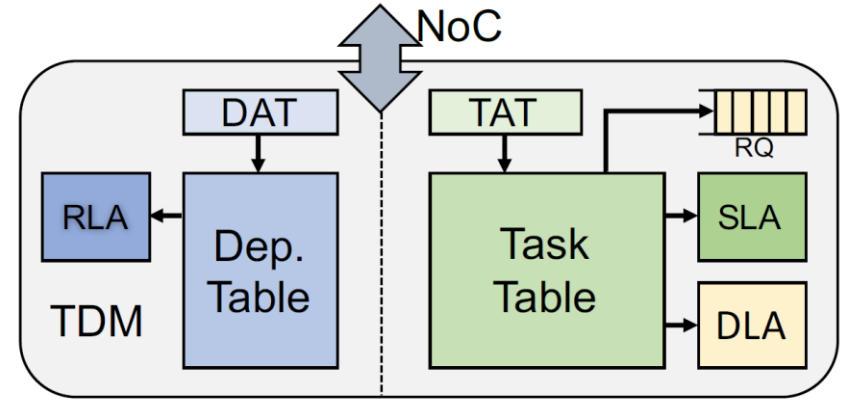


Y. Etsion et al, "Task Superscalar: An Out-of-Order Task Pipeline", MICRO 2010

X. Tan et al, "General Purpose Task-Dependence Management Hardware for Task-based Dataflow Programming Models", IPDPS 2017

# Architectural Support for Task Dependence Management (TDM) with Flexible Software Scheduling

- Task creation is a bottleneck since it involves dependence tracking
- Our hardware proposal (TDM)
  - takes care of dependence tracking
  - exposes scheduling to the SW
- Our results demonstrate that this flexibility allows TDM to beat the state-of-the-art



Task descriptor Address	Task ID
0x8AB0...4600	0
0x8AB0...5240	2
⋮	⋮

Task descriptor Address	Predec. count	Successor		Dep. list ptr.
		count	list ptr.	
0x8AB0...4600	3	1	0	11
0x8AB0...5240	2	1	8	0
⋮	⋮	⋮	⋮	⋮

Dependence Address	Dep. ID
0x0BCE...0860	2
0x0964...4628	1
⋮	⋮

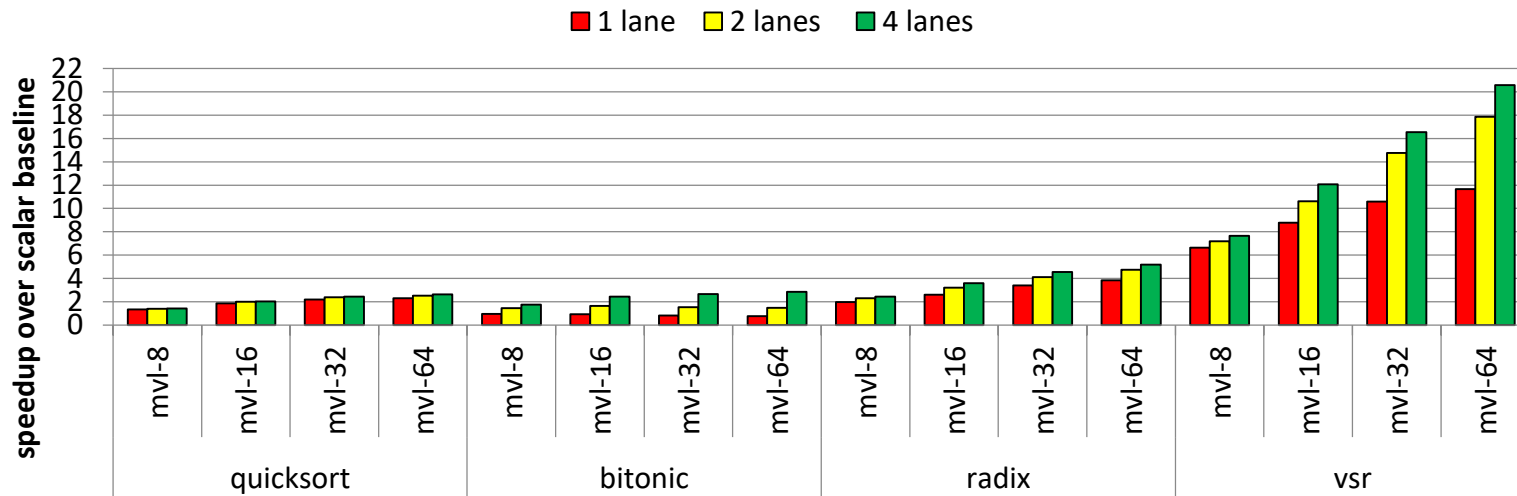
Last writer task ID	Reader List ptr.
0	2
⋮	⋮





# Hash Join, Sorting, Aggregation, DBMS

- Goal: Vector acceleration of data bases
- “Real vector” extensions to x86
  - Pipeline operands to the functional unit (like Cray machines, not like SSE/AVX)
  - Scatter/gather, masking, vector length register
  - Implemented in PTLSim + DRAMSim2
- Hash join work published in [MICRO 2012](#)
  - 1.94x (large data sets) and 4.56x (cache resident data sets) of speedup for TPC-H
    - Memory bandwidth is the bottleneck
- Sorting paper published in [HPCA 2015](#)
  - Compare existing vectorized quicksort, bitonic mergesort, radix sort on a consistent platform
- Propose novel approach (VSR) for vectorizing radix sort with 2 new instructions
  - Similarity with AVX512-CD instructions (but cannot use Intel’s instructions because the algorithm requires strict ordering)
  - Small CAM
- 3.4x speedup over next-best vectorised algorithm with the same hardware configuration due to:
  - Transforming strided accesses to unit-stride
  - Eliminating replicated data structures
- Ongoing work on aggregations
- Reduction to a group of values, not a single scalar value [ISCA 2016](#)
  - Building from VSR work



# RoMoL Infrastructure

## Dimemas:

- Off-Socket Communicatons (MPI)

## TaskSim:

- On-socket Parallelism (OmpSs)
- Coarse Memory Hierarchy

## Gem5:

- Detailed Memory Hierarchy
- Fine Grain Parallelism (OmpSs)
- Processors Pipeline

## Simulation infrastructure:

We have built a **MU**lti-scale **S**imulation **A**pproach (MUSA)

**Applications:** Building a representative set to show OmpSs' strengths<sup>o</sup>

## Applications:

- HPC codes (OmpSs, MPI+OmpSs)
- PARSECSs (OmpSs)

## Analysis Tools

- Extrae (tracing OmpSs and MPI+OmpSs codes)
- Paraver (analysis of OmpSs and MPI+OmpSs codes)

## Analysis Tools:

We are using a solid tool kit to analyze our codes on real HW

# Related Work

- **Rigel Architecture (ISCA 2009)**
  - No L1D, non-coherent L2, read-only, private and cluster-shared data
  - Global accesses bypass the L2 and go directly to L3
- **SARC Architecture (IEEE MICRO 2010)**
  - Throughput-aware architecture
  - TLBs used to access remote LMs and migrate data accross LMs
- **Runnemedede Architecture (HPCA 2013)**
  - Coherence islands (SW managed) + Hierarchy of LMs
  - Dataflow execution (codelets)
- **Carbon (ISCA 2007)**
  - Hardware scheduling for task-based programs
- **Holistic run-time parallelism management (ICS 2013)**
- **Runtime-guided coherence protocols (IPDPS 2014)**

# RoMoL ... papers

- V. Marjanovic et al., “Effective communication and computation overlap with hybrid MPI/SMPs.” **PPoPP 2010**
- Y. Etsion et al., “Task Superscalar: An Out-of-Order Task Pipeline.” **MICRO 2010**
- N. Vujic et al., “Automatic Prefetch and Modulo Scheduling Transformations for the Cell BE Architecture.” **IEEE TPDS 2010**
- V. Marjanovic et al., “Overlapping communication and computation by using a hybrid MPI/SMPs approach.” **ICS 2010**
- T. Hayes et al., “Vector Extensions for Decision Support DBMS Acceleration”. **MICRO 2012**
- L. Alvarez, et al., “Hardware-software coherence protocol for the coexistence of caches and local memories.” **SC 2012**
- M. Valero et al., “Runtime-Aware Architectures: A First Approach”. **SuperFRI 2014**
- L. Alvarez, et al., “Hardware-Software Coherence Protocol for the Coexistence of Caches and Local Memories.” **IEEE TC 2015**

# RoMoL ... papers

- M. Casas et al., “Runtime-Aware Architectures”. **Euro-Par 2015**.
- T. Hayes et al., “VSR sort: A novel vectorised sorting algorithm & architecture extensions for future microprocessors”. **HPCA 2015**
- K. Chronaki et al., “Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures”. **ICS 2015**
- L. Alvarez et al., “Coherence Protocol for Transparent Management of Scratchpad Memories in Shared Memory Manycore Architectures”. **ISCA 2015**
- L. Alvarez et al., “Run-Time Guided Management of Scratchpad Memories in Multicore Architectures”. **PACT 2015**
- L. Jaulmes et al., “Exploiting Asynchrony from Exact Forward Recoveries for DUE in Iterative Solvers”. **SC 2015**
- D. Chasapis et al., “PARSECSs: Evaluating the Impact of Task Parallelism in the PARSEC Benchmark Suite.” **ACM TACO 2016**.
- E. Castillo et al., “CATA: Criticality Aware Task Acceleration for Multicore Processors.” **IPDPS 2016**

# RoMoL ... papers

- T. Hayes et al “Future Vector Microprocessor Extensions for Data Aggregations.” **ISCA 2016.**
- D. Chasapis et al., “Runtime-Guided Mitigation of Manufacturing Variability in Power-Constrained Multi-Socket NUMA Nodes.” **ICS 2016**
- P. Caheny et al., “Reducing cache coherence traffic with hierarchical directory cache and NUMA-aware runtime scheduling.” **PACT 2016**
- T. Grass et al., “MUSA: A multi-level simulation approach for next-generation HPC machines.” **SC 2016**
- I. Brumar et al., “ATM: Approximate Task Memoization in the Runtime System.” **IPDPS 2017**
- K. Chronaki et al., “Task Scheduling Techniques for Asymmetric Multi-Core Systems.” **IEEE TPDS 2017**
- C. Ortega et al., “libPRISM: An Intelligent Adaptation of Prefetch and SMT Levels.” **ICS 2017**
- V. Dimic et al., “Runtime-Assisted Shared Cache Insertion Policies Based on Reference Intervals.” **EuroPAR 2017**

# 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:



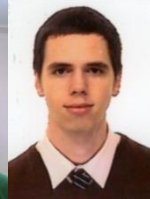
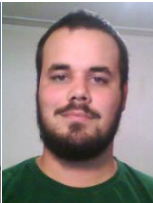
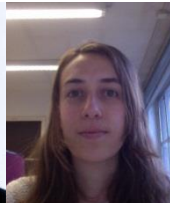
- Researchers:



Postdocs:



- Students:



- **Open for collaborations!**

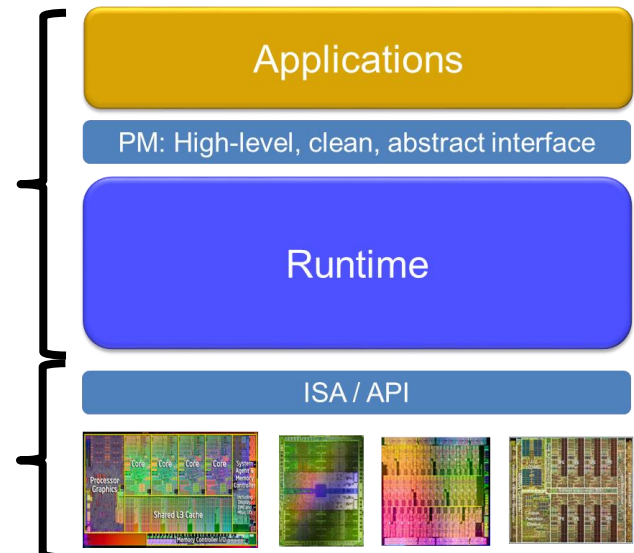
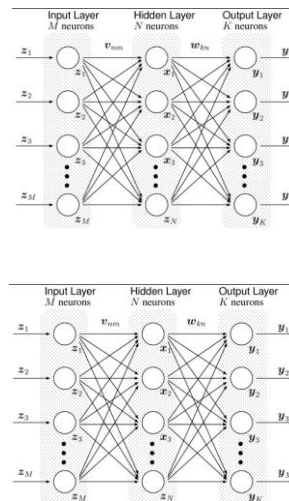


# High-level Overview of the Proposal and Goals

- ⌘ We propose a management agent able to dynamically adapt hardware and system software:
  - At the HW level, it will monitor the architecture status, evaluate, and adapt it.
  - At the system SW level, it will monitor the program state and adapt the OS and runtime system policies (E.g. scheduling).
- ⌘ The goal is to predict hardware and software states as many cycles as possible in advance to enable optimizations (E.g. pre-fetcher).

Management Agent

SW models  
HW models






# Mare Nostrum RISC-V inauguration 202X

Por el autor de *El código Da Vinci*

# DAN BROWN ORIGEN

MN-RISC-V



 Planeta





**Barcelona  
Supercomputing  
Center**  
Centro Nacional de Supercomputación



EXCELENCIA  
SEVERO  
OCHOA

THANK YOU!

[www.bsc.es](http://www.bsc.es)