www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

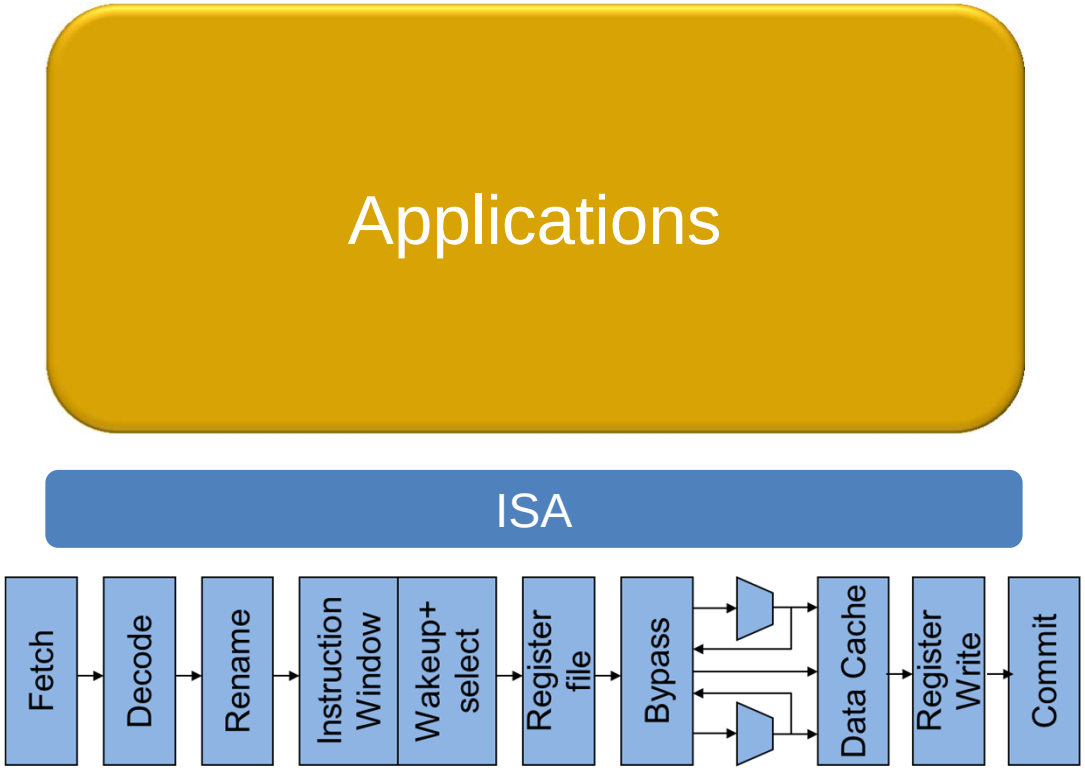
Why Exascale will not Appear without Runtime Aware Architectures

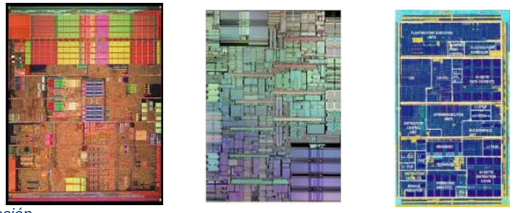
Prof. Mateo Valero

Russian Supercomputing Days
Moscow September 2015

Design of Superscalar Processors

- Decoupled from the software stack





Programs "decoupled" from hardware

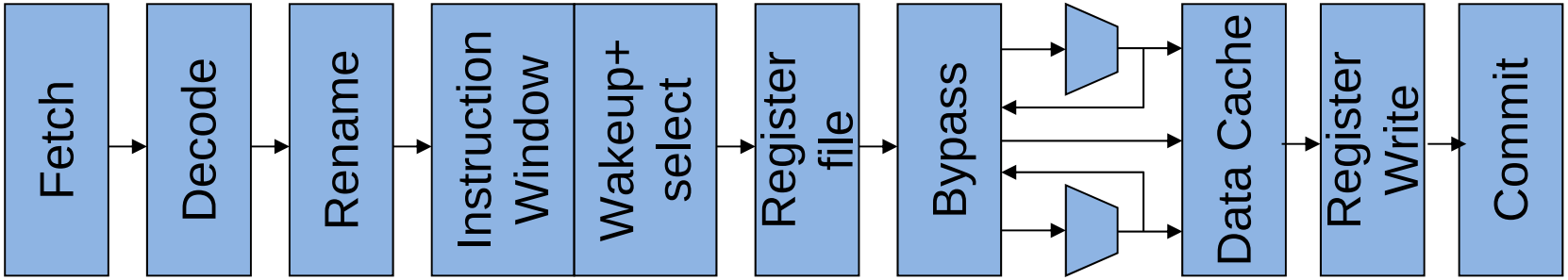
Simple interface Sequential program

ILP

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

RoMoL Project

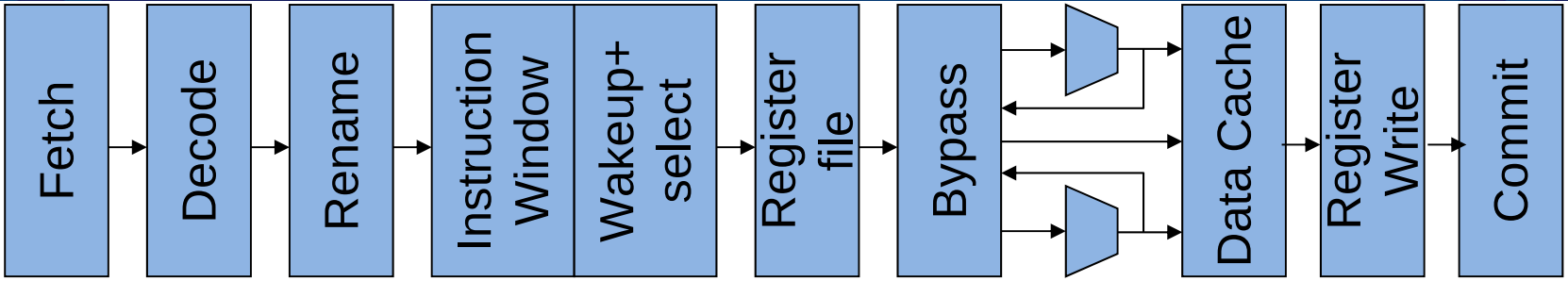
Latency Has Been a Problem from the Beginning… ☹



- Feeding the pipeline with the right instructions:
  - HW/SW trace cache (ICS'99)
  - Prophet/Critic Hybrid Branch Predictor (ISCA'04)
- Locality/reuse
  - Cache Memory with Hybrid Mapping (IASTED'87). Victim Cache ☺
  - Dual Data Cache (ICS¨95)
- 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 ☹ ☹ ☹

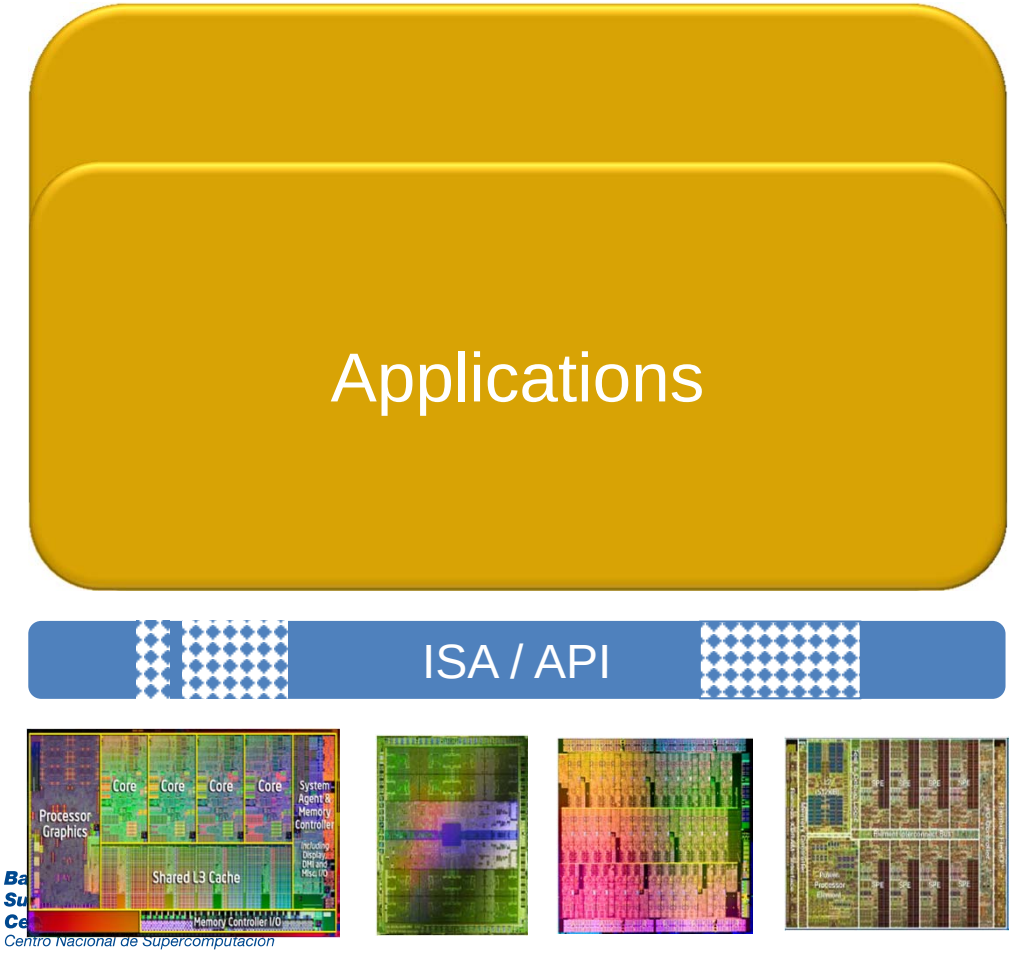


- Better Technologies
- Two-level organization (Locality Exploitation)
  - Register file for Superscalar (ISCA'00)
  - Instruction queues (ICCD'05)
  - Load/Store Queues (ISCA'08)
- Content-aware register file (ISCA'09)
- Direct Wakeup, Pointer-based Instruction Queue Design (ICCD'04, ICCD'05)
- Fuzzy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

Power Wall

Living in the Programming Revolution

- Multicores made the interface to leak…

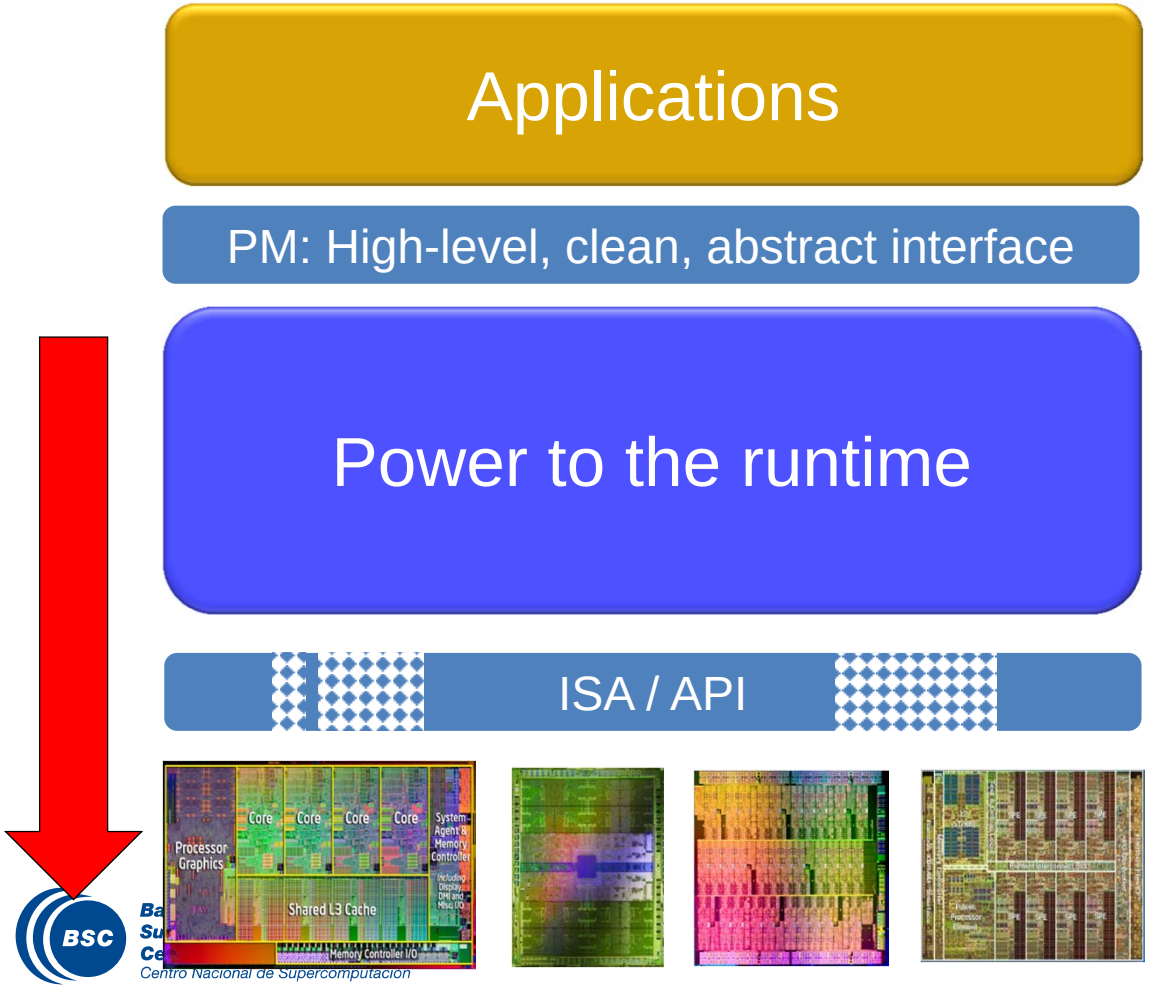


Parallel application logic
+
Platform specificites

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

Vision in the Programming Revolution

- Need to decouple again



Application logic

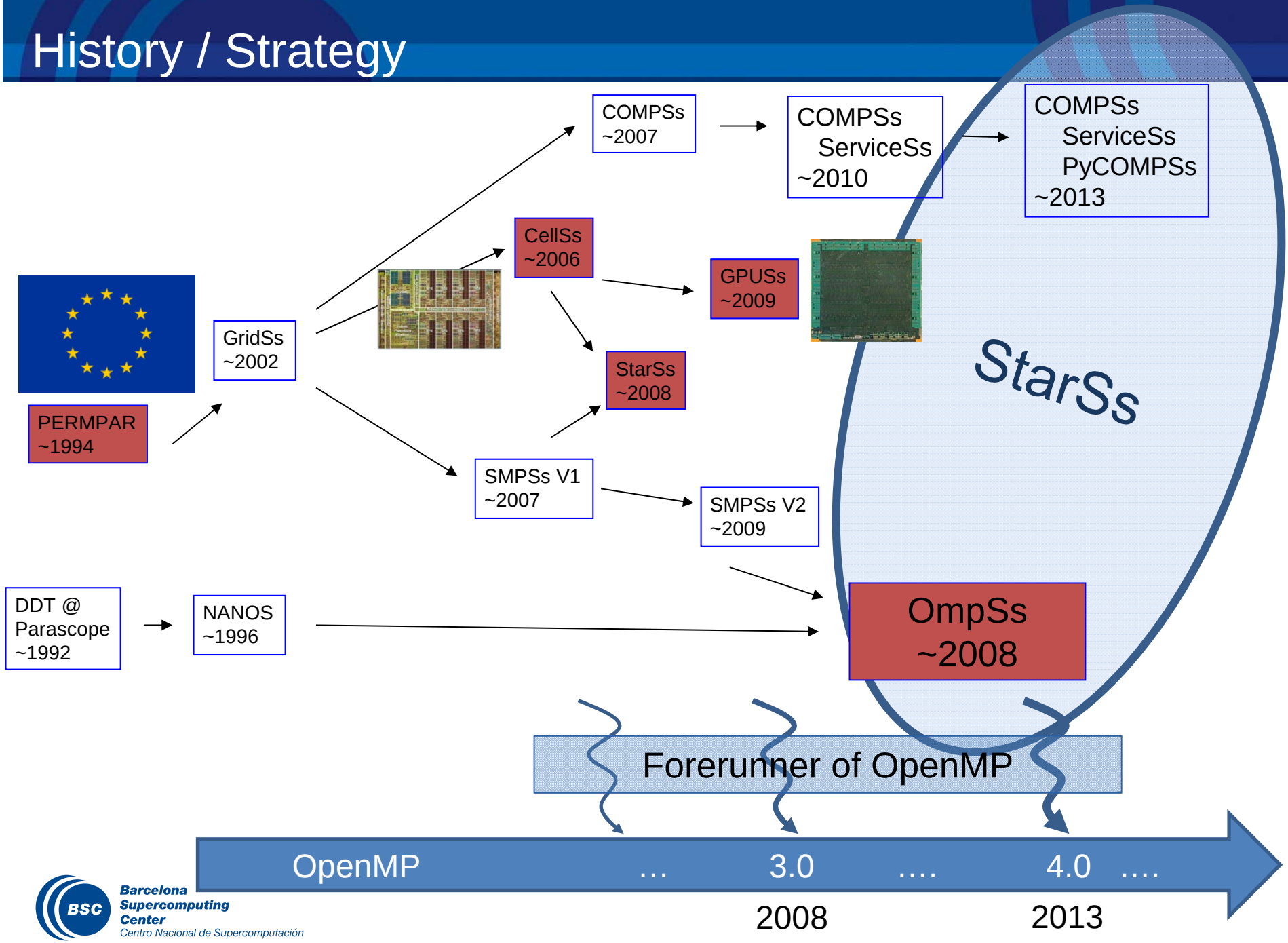
Arch. independent

General purpose

Single address space

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

History / Strategy



OmpSs: A Sequential Program …

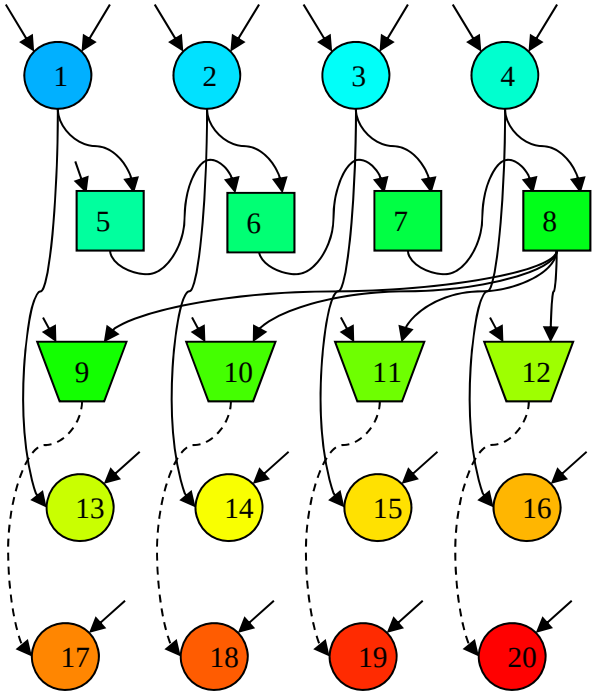
```
void vadd3 (float A[BS], float B[BS],
            float C[BS]);

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]);
```



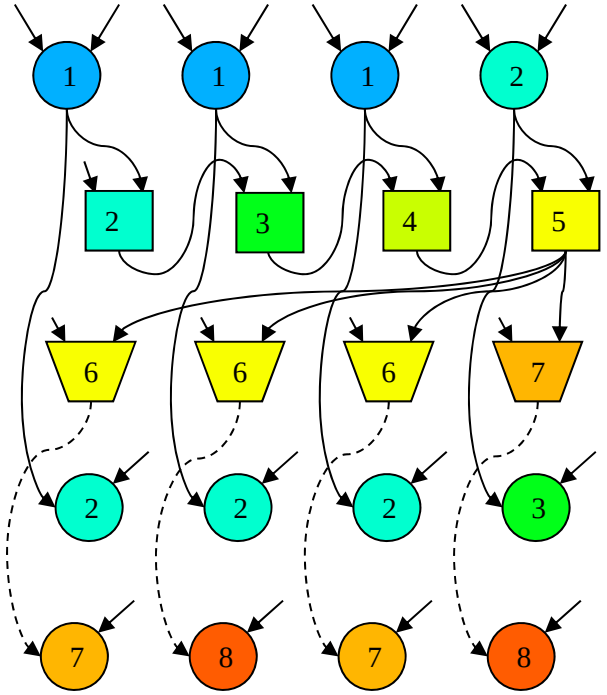
Color/number: order of task instantiation
Some antidependences covered by flow dependences not drawn

… 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 Ubiquity

- OmpSs @ Cell
  - CellSs [SC 2006, IBM JRD 2007]
  - Speculative Distributed Scheduling [IPDPS 2011]
- OmpSs @ Multicores [PPL 2011]
- OmpSs @ Clusters
  - Multicores [EuroPAR 2011, IPDPS 2013-1, ICS 2013]
  - Multicores+GPU [ICS 2011, IPDPS 2012]
- OmpSs @ Multicore+GPU [IPDPS 2013-2]
- OmpSs @ Zynq
  - Offload computation and Nanos++ runtime acceleration [FPGA 2014]
- OmpSs @ multiple GPUs
  - High asynchrony and overlap (transfers and kernels)
  - Improved schedulers

CellSs, StarSs, OmpSs,…. papers

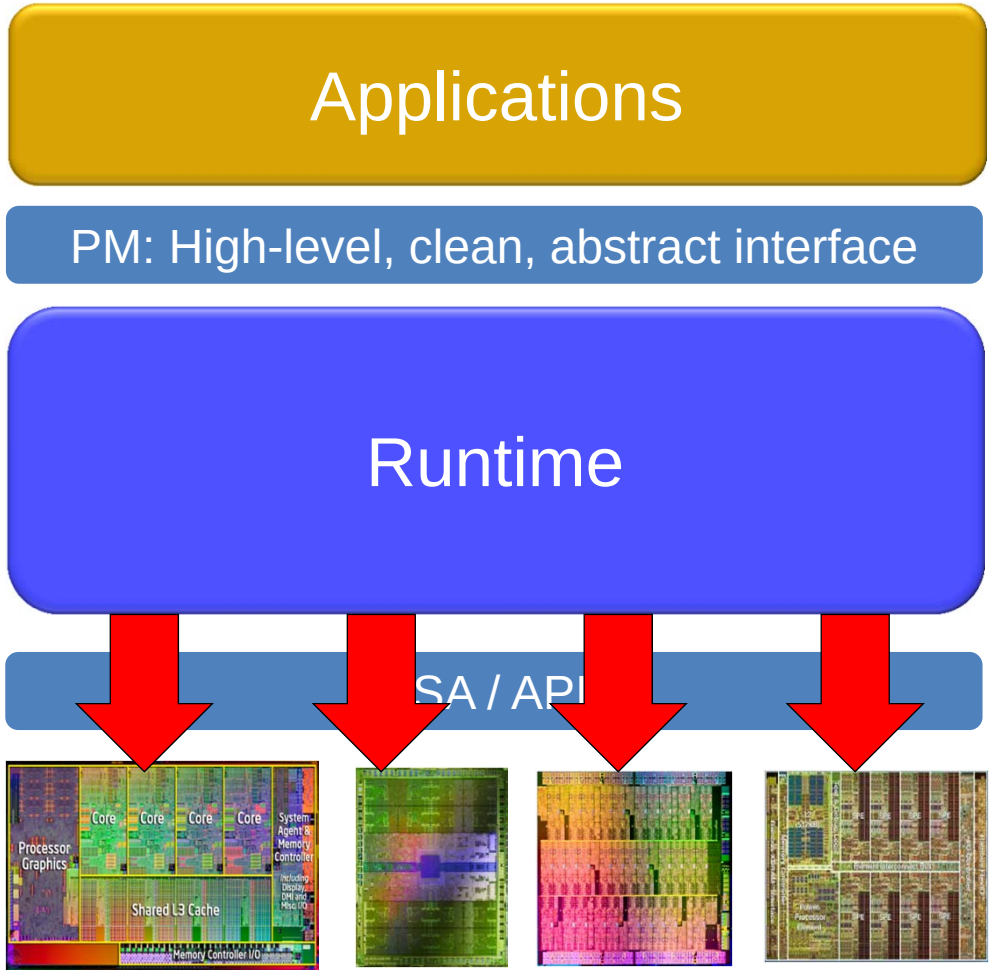
- P. Bellens,...“Memory – CellSs: a programming model for the Cell BE architecture.” **SC 2006**
- J. M. Pérez, et al. “CellSs: Making it easier to program the Cell Broadband Engine processor.” **IBM Journal of Research and Development 2007**
- J. M. Pérez, et al: “A dependency-aware task-based programming environment for multi-core architectures.” **CLUSTER 2008**
- P. Bellens,...“Exploiting Locality on the Cell/B.E. through Bypassing.” **SAMOS 2009**
- E. Ayguadé et al.:A Proposal to Extend the OpenMP Tasking Model for Heterogeneous Architectures. **IWOMP 2009**
- P. Bellens, et al. “Just-in-Time Renaming and Lazy Write-Back on the Cell/B.E**.” ICPP Workshops 2009**
- E. Ayguadé,: “An Extension of the StarSs Programming Model for Platforms with Multiple GPUs.” **Euro-Par 2009**
- P. Bellens, et al.”CellSs: Scheduling techniques to better exploit memory hierarchy.” **Scientific Programming 2009**
- A. Duran, et al. “A Proposal to Extend the OpenMP Tasking Model with Dependent Tasks.” **International Journal of Parallel Programming** 2009
- J.Labarta et al “BSC Vision Towards Exascale.” **IJHPCA 2009**

CellSs, StarSs, OmpSs,…. papers

- E. Ayguadé ET AL “Extending OpenMP to Survive the Heterogeneous Multi-Core Era.” **International Journal of Parallel Programming** 2010
- P. Bellens, …”A Study of Speculative Distributed Scheduling on the Cell/B.E**.” IPDPS 2011**
- J. Labarta, et al. “Hybrid Parallel Programming with MPI/StarSs.” **PARCO 2011**
- J. Bueno, et al. “Programming clusters of GPUs with OMPSs**. ICS 2011**
- A. Duran, et al “Ompss: a Proposal for Programming Heterogeneous Multi-Core Architectures.” **Parallel Processing Letters** 2011
- J. Dongarra et al, “The International Exascale Software Project roadmap” **IJHPCA 2011**
- V. Krishnan “OmpSs-OpenCL Programming Model for Heterogeneous Systems” **LCPC 2012**
- N. Vujic, “DMA-circular: an enhanced high level programmable DMA controller for optimized management of on-chip local memories.” **Conf. Computing Frontiers 2012**
- A. Fernández,”Task-Based Programming with OmpSs and Its Application**.” Euro-Par 2014**

Runtime Aware Architectures (RAA)

- The runtime **drives** the hardware design



Task based PM annotated by the user

Data dependencies detected at runtime

Dynamic scheduling

"Reuse" architectural ideas under new constraints

Current trends in HPC architectures make RAA a need

Memory Wall
Power Wall

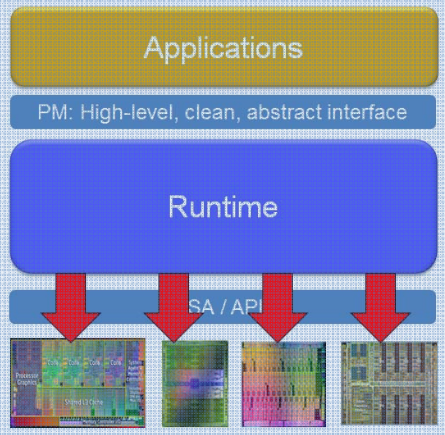
→ Trend towards massive multicore chips
Trend towards heterogeneity

↓

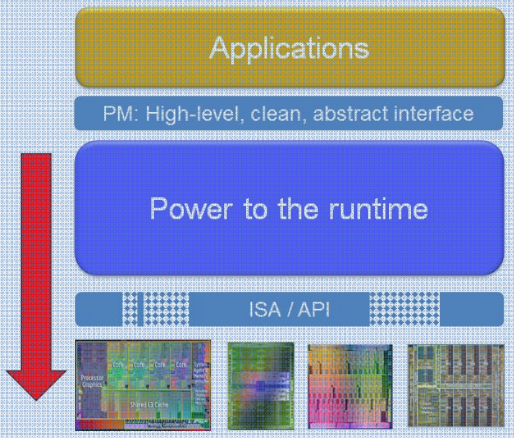
Runtime system support is required

↓

Runtime-Aware Architectures



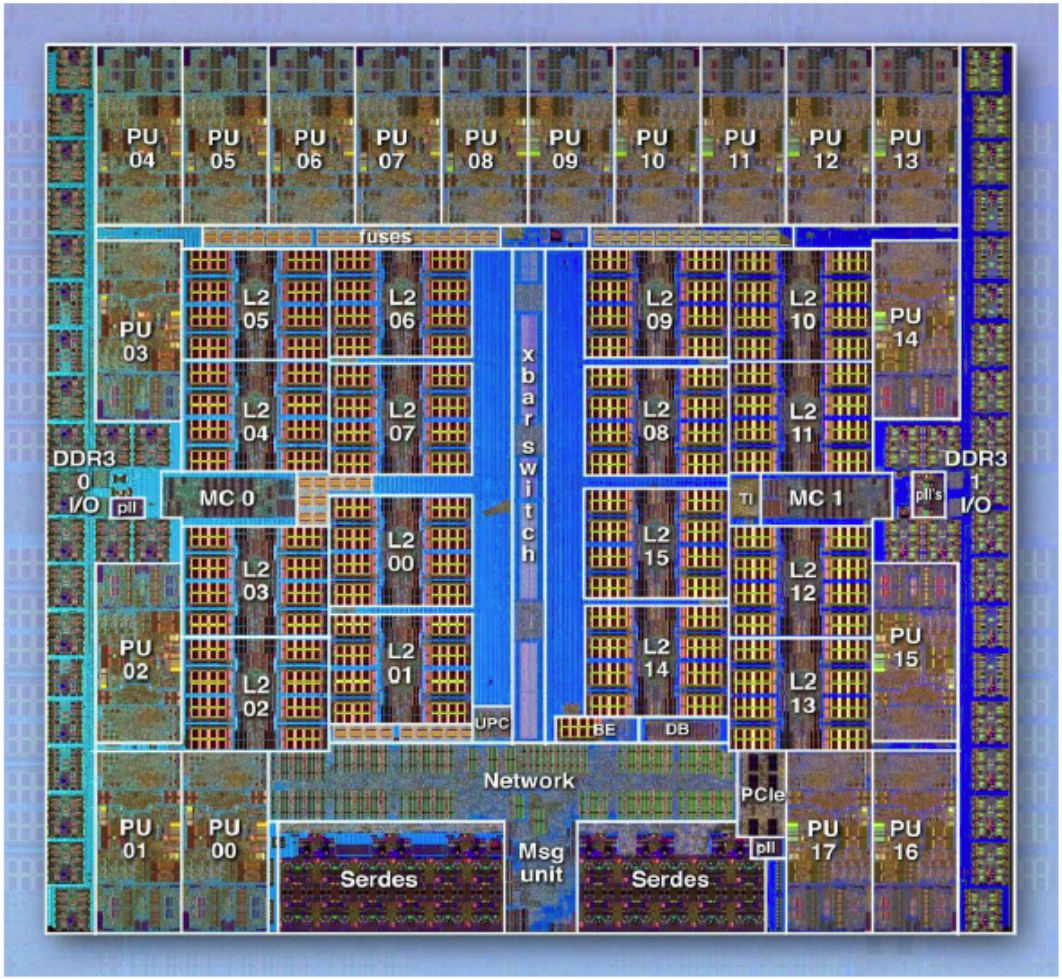
Runtime drives HW design



Efficient HW usage

BlueGene/Q Compute chip

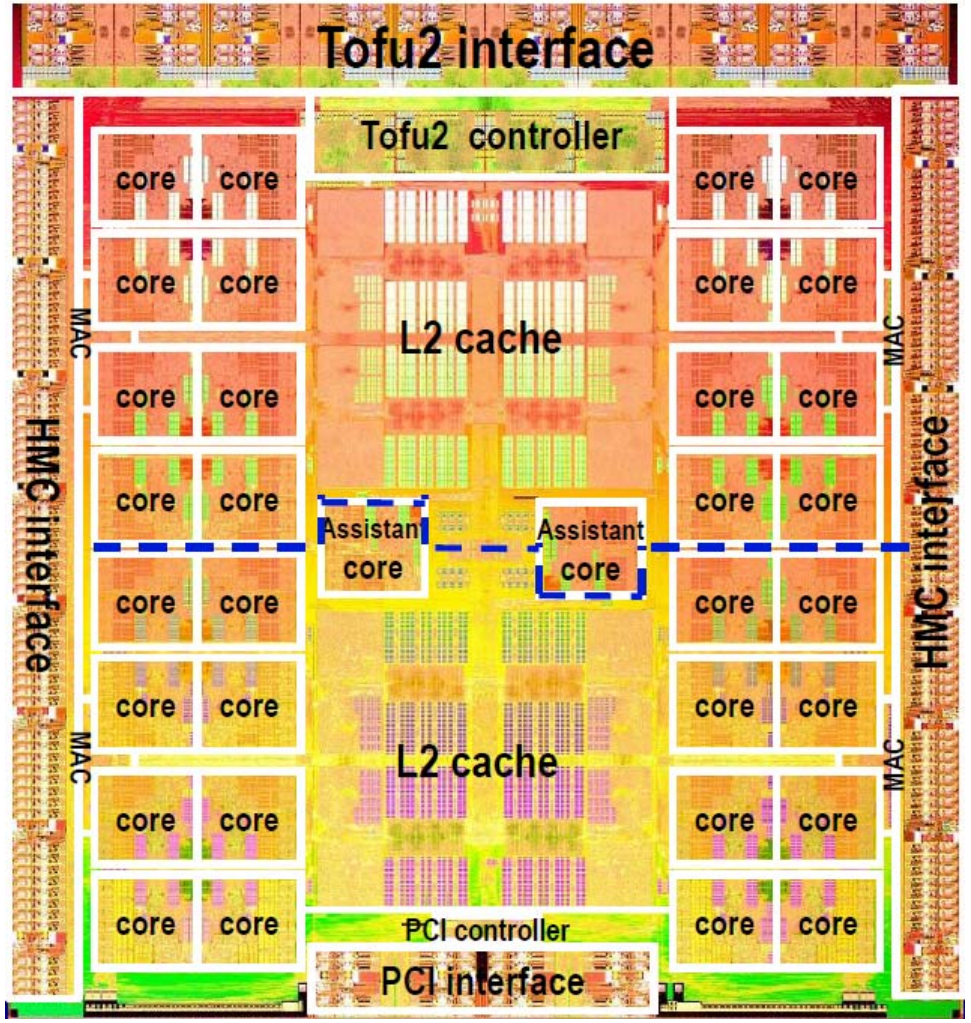
System-on-a-Chip design : integrates processors, memory and networking logic into a single chip



- **360 mm² Cu-45 technology (SOI)**
  - ~ 1.47 B transistors
- **16 user + 1 service processors**
  - plus 1 redundant processor
  - all processors are symmetric
  - each 4-way multi-threaded
  - 64 bits PowerISA™
  - 1.6 GHz
  - L1 I/D cache = 16kB/16kB
  - L1 prefetch engines
  - each processor has Quad FPU (4-wide double precision, SIMD)
  - peak performance 204.8 GFLOPS@55W
- **Central shared L2 cache: 32 MB**
  - eDRAM
  - multiversioned cache will support transactional memory, speculative execution.
  - supports atomic ops
- **Dual memory controller**
  - 16 GB external DDR3 memory
  - 1.33 Gb/s
  - 2 * 16 byte-wide interface (+ECC)
- **Chip-to-chip networking**
  - Router logic integrated into BQC chip.
- **External IO**
  - PCIe Gen2 interface

Fujitsu SPARC64 XIfx

- 32 computing cores (single threaded) + 2 assistant cores
- 24MB L2 sector cache
- 256-bit wide SIMD
- 20nm, 3.75M transistors
- 2.2GHz frequency
- 1.1TFlops peak performance
- High BW interconnects
  - HMC (240GB/s x 2 in/out)
  - Tofu2 (125GB/s x 2 in/out)



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

Superscalar vision at Multicore level

Superscalar World	Multicore World
Out-of-Order, Kilo-Instruction Processor, Distant Parallelism	Task-based, Data-flow Graph, Dynamic Parallelism
Branch Predictor, Speculation	Tasks Output Prediction, Speculation
Fuzzy Computation	
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
Memory Wall / **Power Wall**	Load Balancing and Scheduling
Programmability Wall / **Resilience Wall**	Interconnection Network
	Data Movement

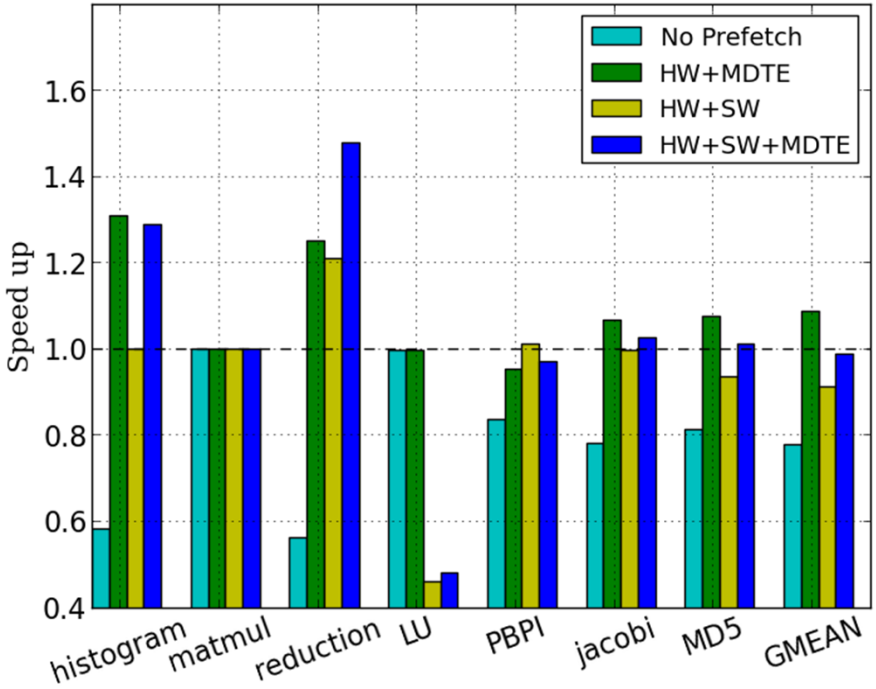
Runtime Aware Architectures (RAA)

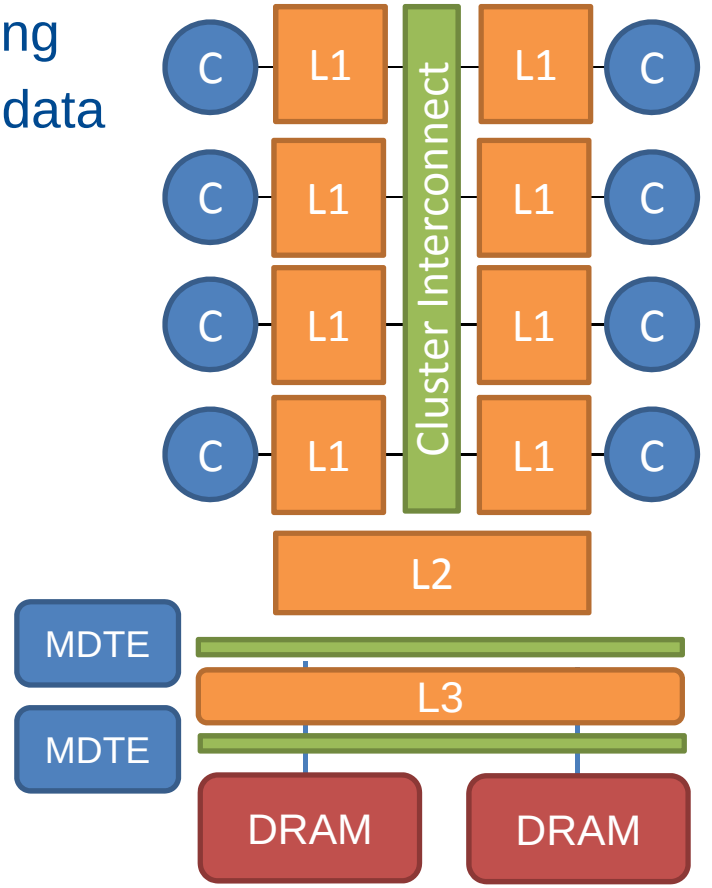
- Re-design memory hierarchy
  - Hybrid (cache + local memory)
  - Non-volatile memory, 3D stacking
  - Simplified coherence protocols, non-coherent islands of cores
- Exploitation of data locality:
  - Reuse, prefetching, in-memory computation

Memory Wall

Runtime-Assisted Data Prefetching

- Hide off-chip latency via data block prefetching
- Leveraging runtime knowledge of task input data
- Explicit data transfer without programmer intervention
  - Command to Data Transfer Engine (MDTE)
  - Requests to memory to store data into L2
- Hardware support for block prefetching

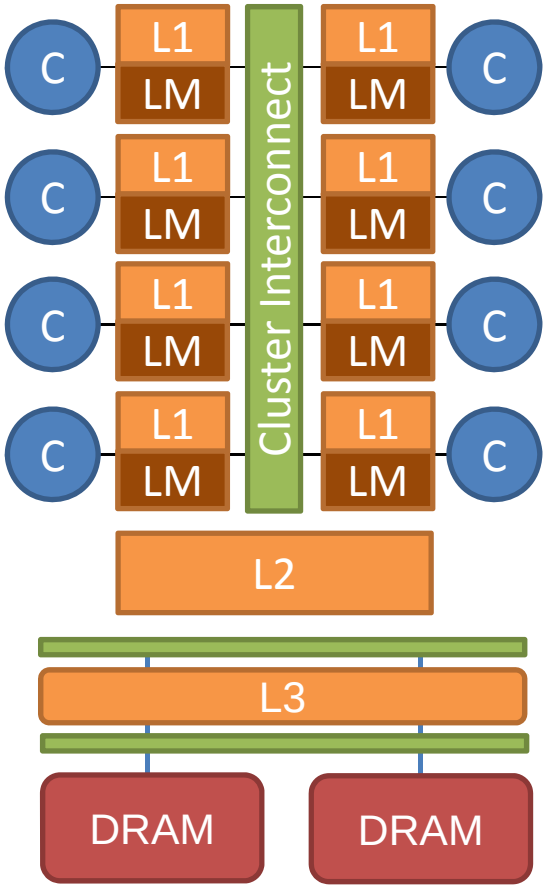


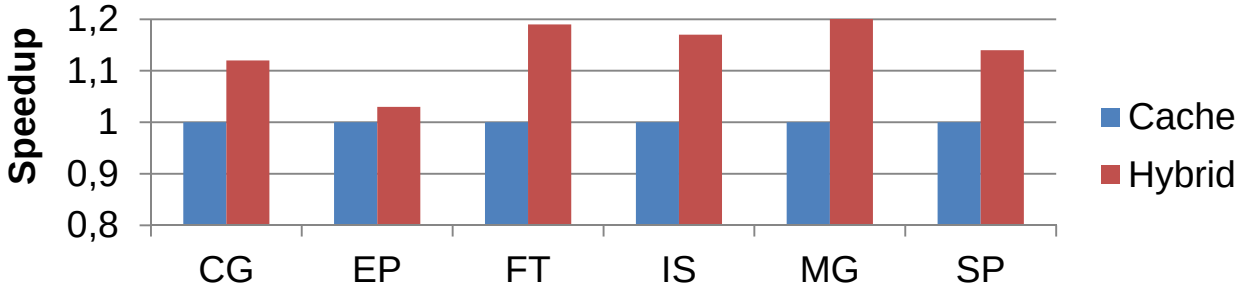


V. Papaefstathiou et al.: Prefetching and cache management using task lifetimes. ICS 2013.
V. Garcia et al.. OMHI 2014, Euro-Par workshop. (Best paper award)

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





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

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

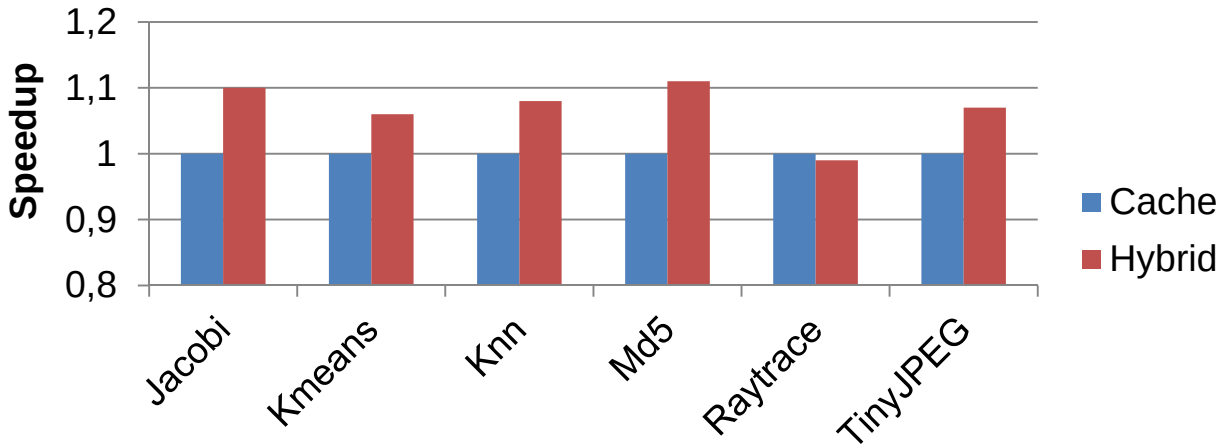
Transparent Management of Local Memories

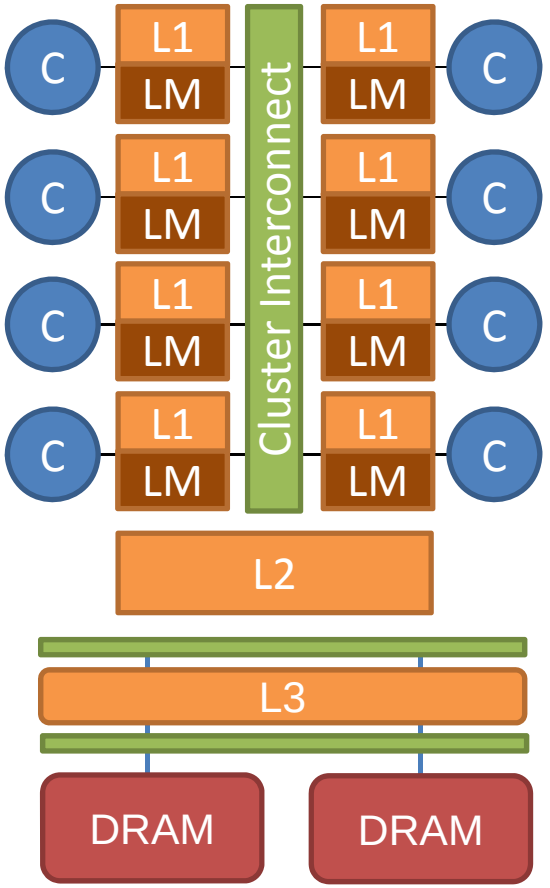
- LM Management in OmpSs (PACT'15)
  - Task inputs and outputs mapped to the LMs
  - Runtime manages DMA transfers

8.7% speedup in execution time

14% reduction in power

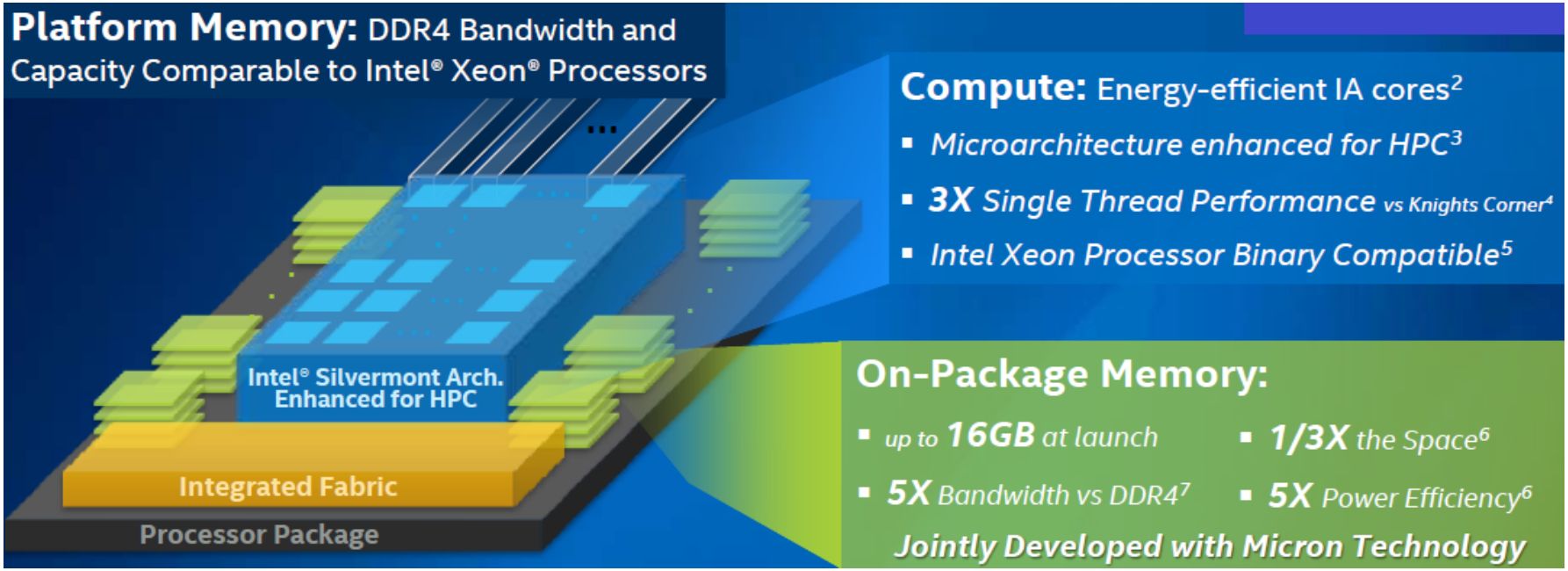
20% reduction in network-on-chip traffic





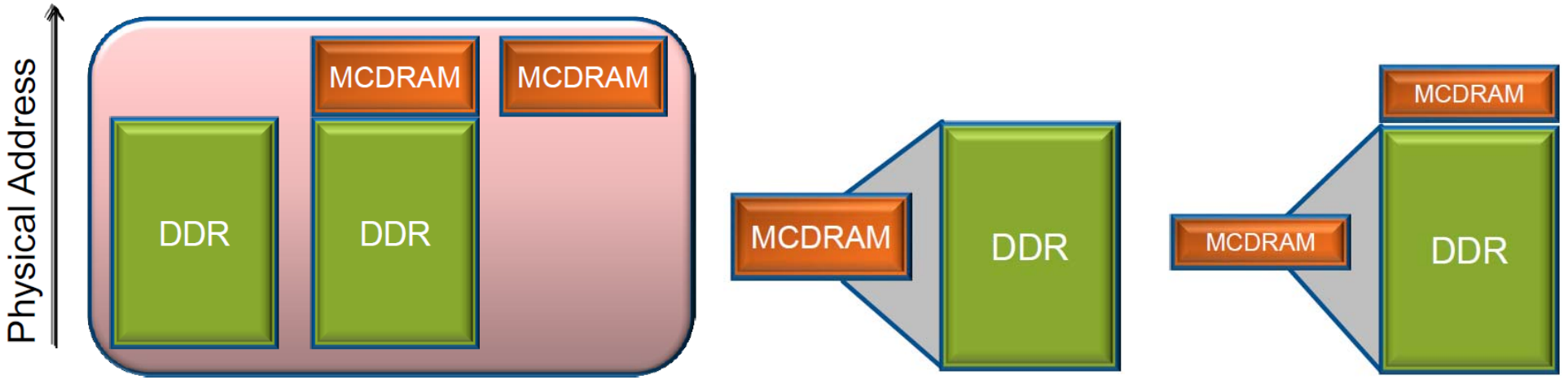
Ll. Alvarez et al. Runtime-Guided Management of Scratchpad Memories in Multi-core Architectures . PACT 2015

Intel Knights Landing

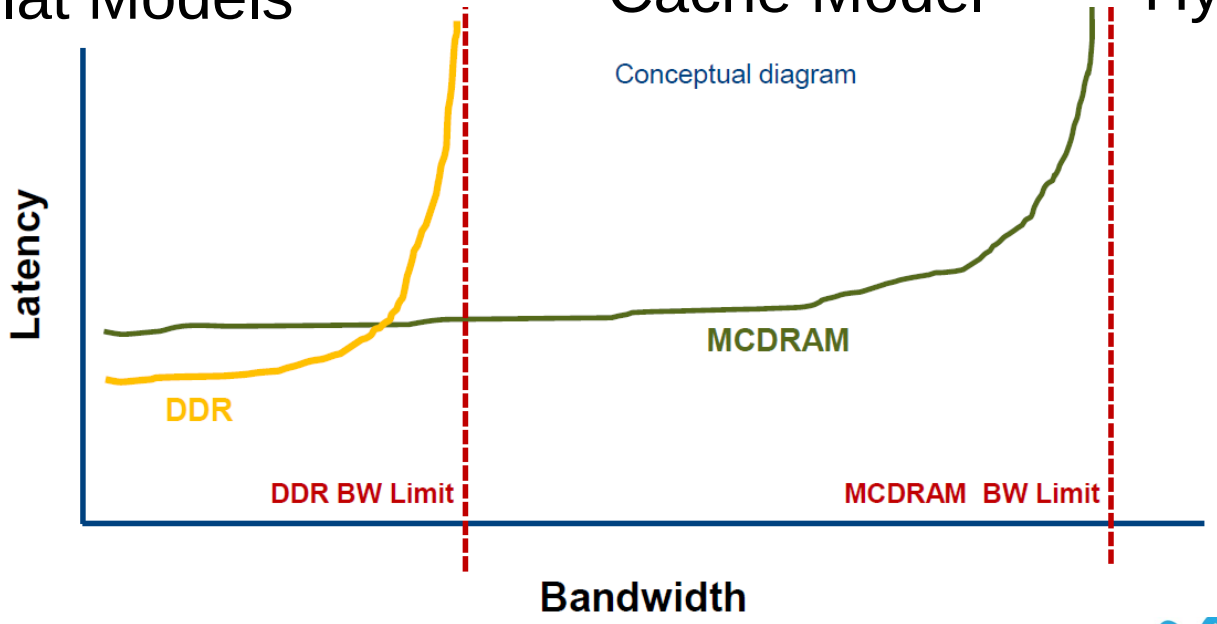


- Intel's Knights Landing has a hybrid and configurable memory hierarchy

Memory Models & Technological Characteristics @ KNL



Flat Models | Cache Model | Hybrid Model



Source: A. Sodani. "Intel Xeon Phi Processor Knights Landing Architectural Overview". Keynote at IXPUG workshop organized at ISC 2015.

High Bandwidth (HBW) Malloc API

```
HBWMALLOC(3)                HBWMALLOC                HBWMALLOC(3)

NAME
       hbwmalloc - The high bandwidth memory interface

SYNOPSIS
       #include <hbwmalloc.h>

       Link with -ljemalloc -lnuma -lmemkind -lpthread

       int hbw_check_available(void);
       void* hbw_malloc(size_t size);
       void* hbw_calloc(size_t nmemb, size_t size);
       void* hbw_realloc (void *ptr, size_t size);
       void hbw_free(void *ptr);
       int hbw_posix_memalign(void **memptr, size_t alignment, size_t size);
       int hbw_posix_memalign_psize(void **memptr, size_t alignment, size_t size, int pagesize);
       int hbw_get_policy(void);
       void hbw_set_policy(int mode);
```

- Allows determining the desired “kind” of memory to use
- Our approach: Power to the runtime to decide where to allocate memory for a particular application!

Source: A. Sodani. “Intel Xeon Phi Processor Knights Landing Architectural Overview”. Keynote at IXPUG workshop organized at ISC 2015.

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

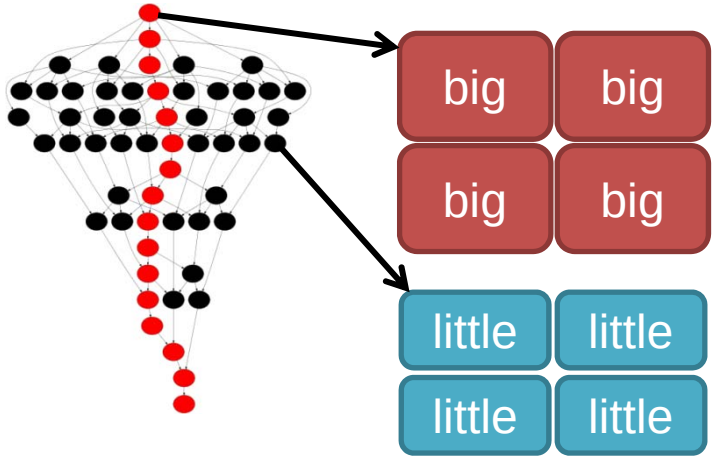
Memory Wall

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

Power Wall

OmpSs in Heterogeneous Systems

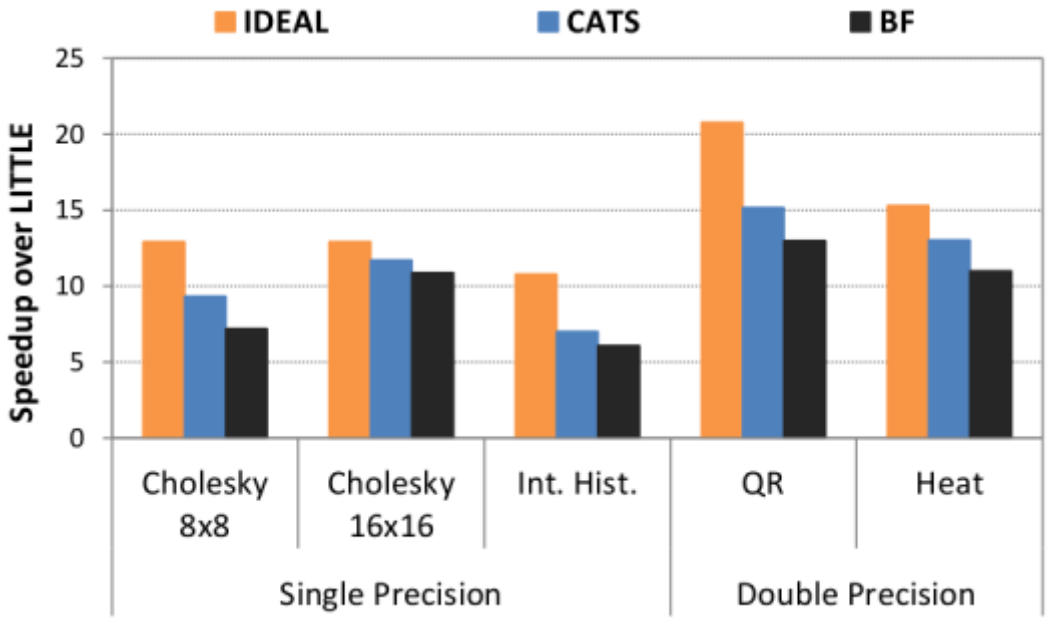
- Heterogeneous systems
  - Big-little processors
  - Accelerators
  - Hard to program



- 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 core of balancing the load
  - Same performance with less power consumption

Discovering Task Criticality in the Runtime

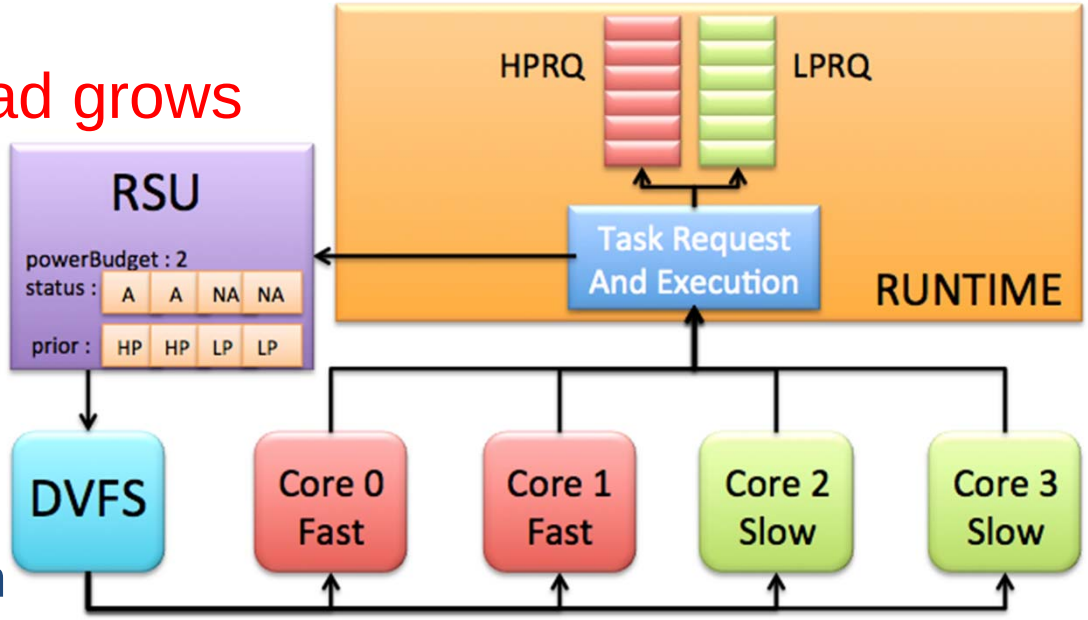
- Criticality-Aware Task Scheduler (CATS, ICS 2015)
  - Critical tasks are those in the longest path in the task dependence graph
- Comparison to default OmpSs scheduler: Breadth First (BF)
- Run on Exynos 5422 big.LITTLE
  - 4xCortex-A15 @ 2GHz
  - 4xCortex-A7 @ 1.4GHz
- Results running on 8 cores
- CATS allows applications to scale better and get closer to ideal speedup
- Speedup for double precision apps is larger
  - Difference between big and LITTLE is larger for double than for single-precision floating-point computation



K. Chronaki et al. Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures. ICS 2015.

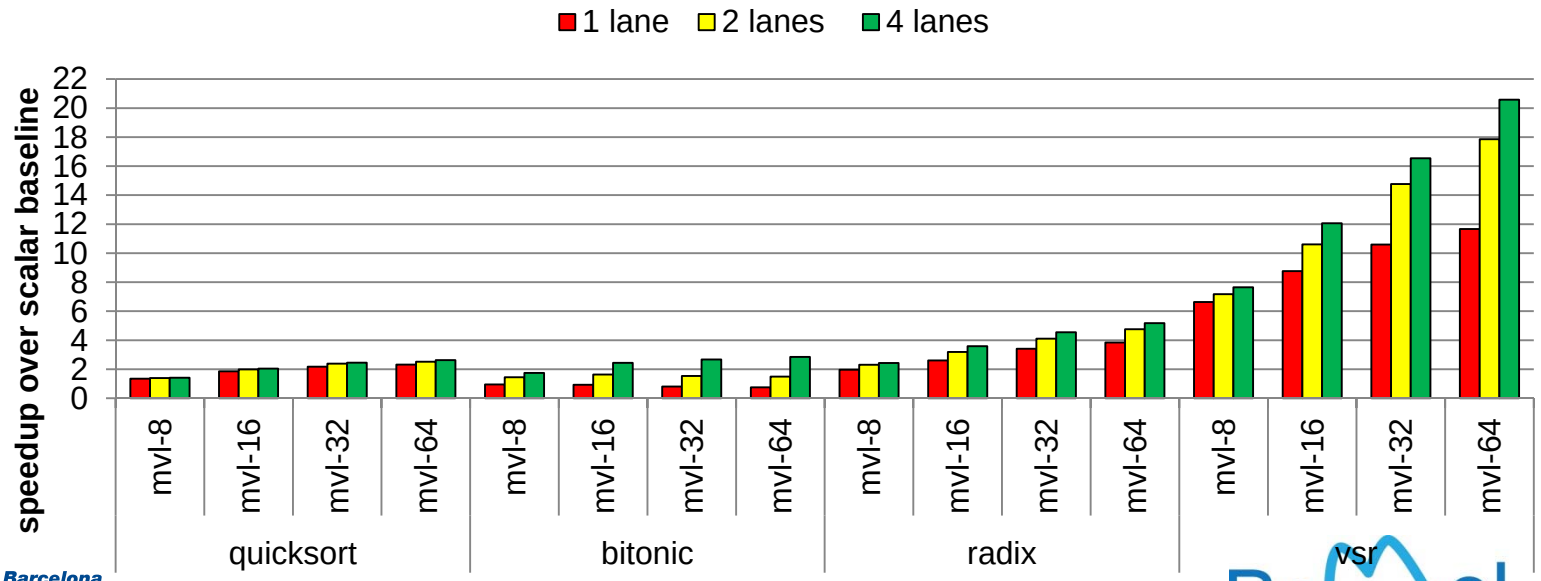
Task Criticality Aware Acceleration

- Cores computation power can be reconfigured
  - Runtime drives cores reconfigurations using DVFS according to task criticality and available power budget
- Reconfigurations lead to 16% and 45% average improvements in execution time and EDP over static techniques
- Reconfigurations overhead grows with the number of cores
  - Barriers
  - Reconfig frequency
- Hardware Runtime Support Unit (RSU)
  - The ISA is augmented with instructions to notify task execution.
  - The RSU reconfigures the core speed



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
    - Elminating replicated data structures
- Ongoing work on aggregations
  - Reduction to a group of values, not a single scalar value
  - Building from VSR work



BSC proposal for the Exaflop

- Exaflop @ 20MW
- Target: using a 10TF node then
  - Need ~100K nodes
  - Each node ~200W, including memory & network
  - Assume 80%:20% for cores : memory
- Using 800Mhz vector units, we need
  - 100 cores, 2 v.u./core, 64 lanes/v.u., muladd/lane
  - Why 2 v.u.? Assuming a 4wide ARM core driving the v.u., 2 v.u. seems reasonable
- Hence, about 1.6W for a “core+V.U.”
  - Including the ARM front end, the v.u., the caches, the interconnect slice

BSC proposal 2010

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
- Management of shared resources

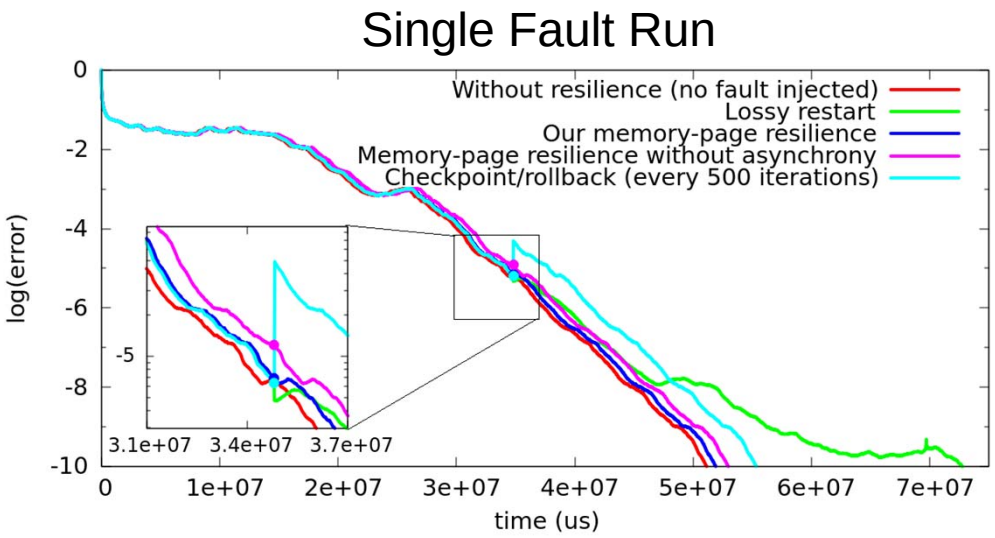
Power Wall

Resilience Wall

- Task-based checkpointing
- Algorithmic-based fault tolerance

OmpSs Runtime-based Resilience

- Advantages of OmpSs for resilience
  - Asynchrony, OoO execution, Input/output annotations
- Algorithmic Recovery Routines
  - Conjugate Gradient
  - Detection
    - Memory Page Retirement
  - Correction
    - Algorithmic
  - Computation/Recovery overlap plus checkpointless techniques → low overhead



L. Jaulmes et al, "Exploiting Asynchrony from Exact Forward Recovery for DUE in Iterative Solvers". SC'15… Nominated to the Best paper Award

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
- Management of shared resources

Power Wall

Programmability Wall

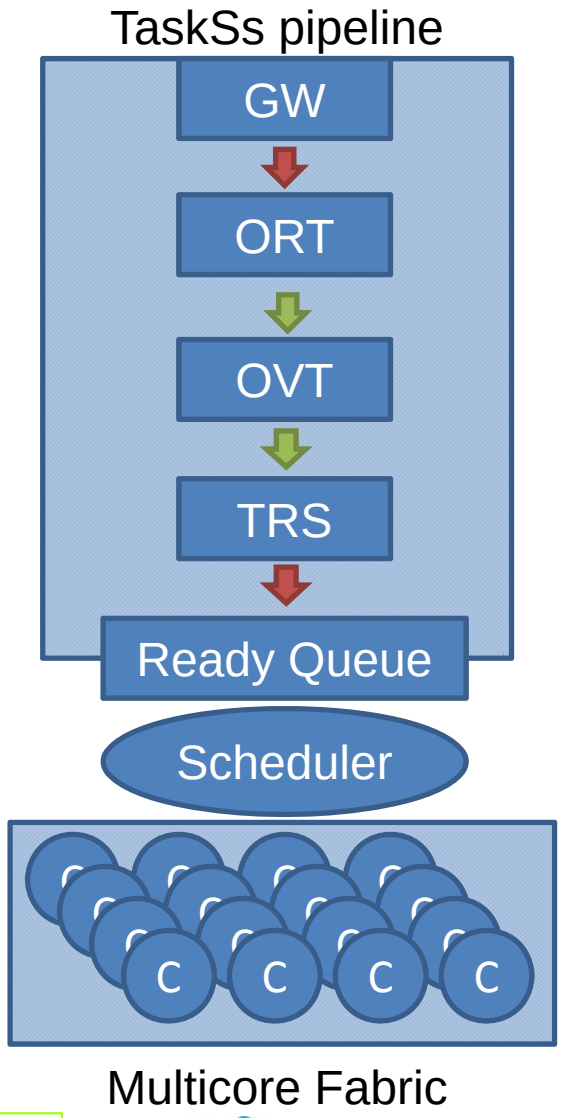
- Hardware acceleration of the runtime system
  - Task dependency graph management
- Load balancing and scheduling
  - Asynchrony and critical path exploitation

Resilience Wall

- Task-based checkpointing
- Algorithmic-based fault tolerance

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-flght meta-data
- Implementing TaskSs @ Xilinx Zynq



Y. Etsion et al, "Task Superscalar: An Out-of-Order Task Pipeline" MICRO-43, 2010

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

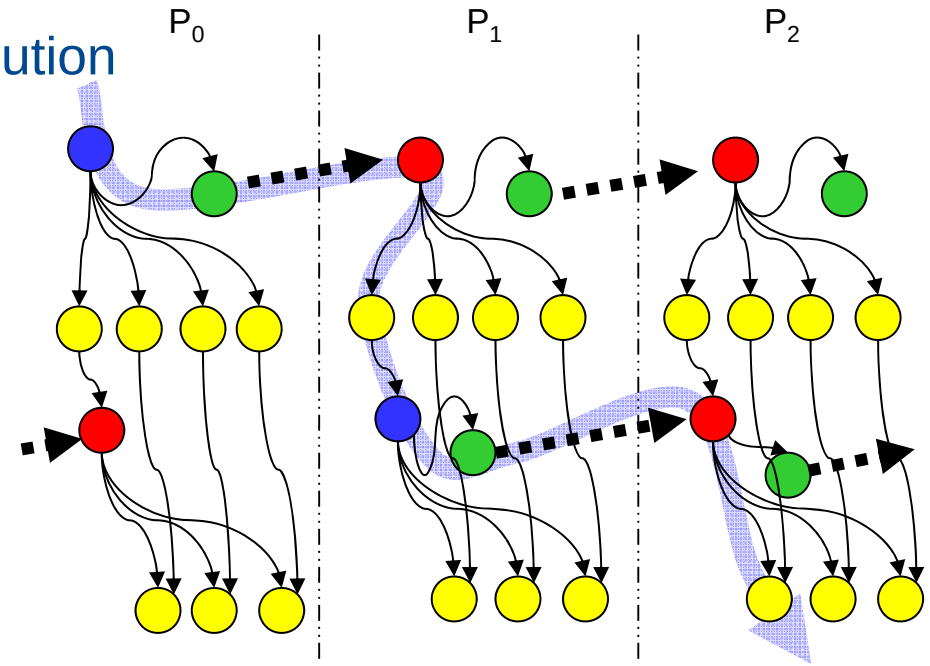
- Hardware acceleration of the runtime system
  - Task depency graph management
- Load Balancing and scheduling
  - Asynchrony and critical path exploitation

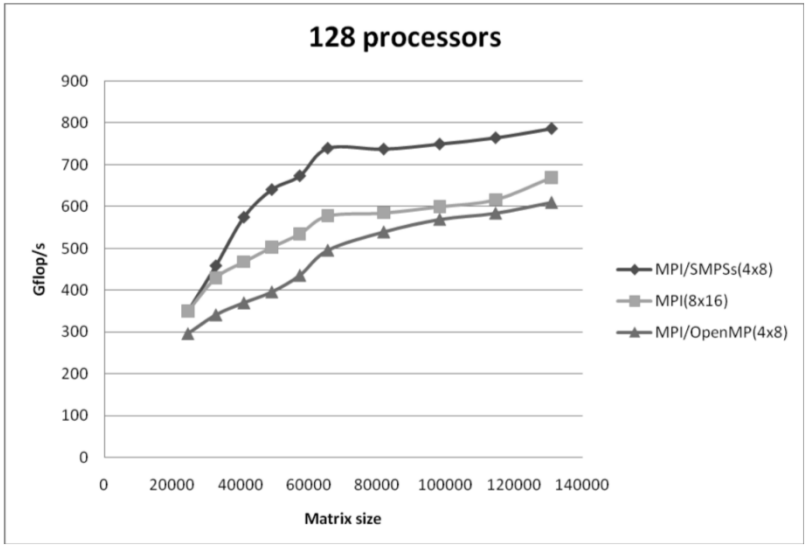
Resilience Wall

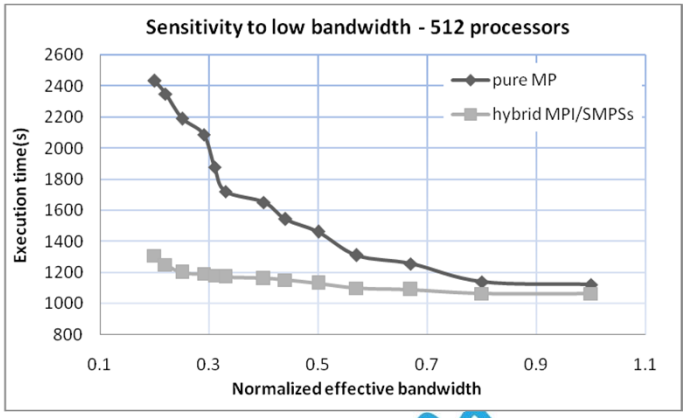
- Task-based checkpointing
- Algorithmic-based fault tolerance

Overlap Communication and Computation

- Hybrid MPI/OmpSs: Linpack example
- Extend asynchronous data-flow execution to outer level
  - Taskify MPI communication primitives
- Automatic lookahead
- Improved performance
- Tolerance to network bandwidth
- Tolerance to OS noise

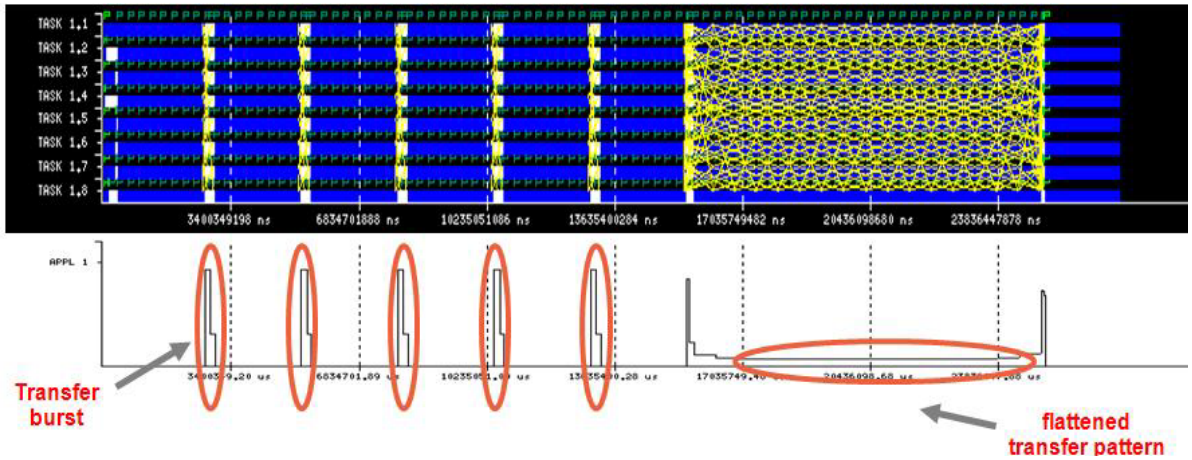






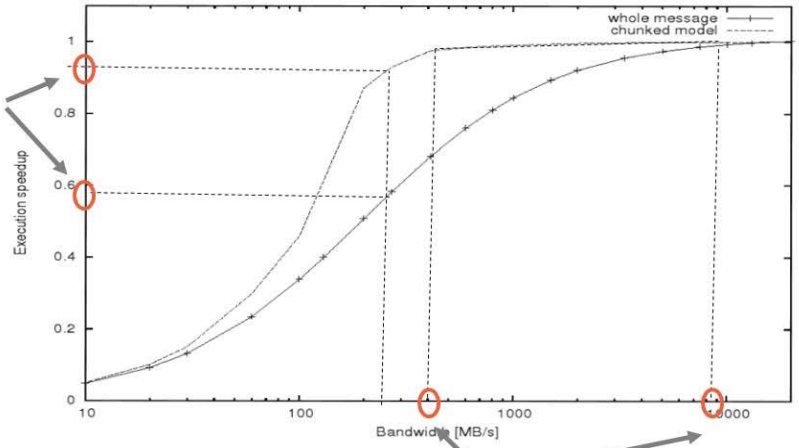
V. Marjanovic et al, "Overlapping Communication and Computation by using a Hybrid MPI/SMPSs Approach" ICS 2010

Effects on Bandwidth



flattening communication pattern

thus



speedup of 1.6 for the same network bandwidth

for the same execution time 20 times lower needed bandwidth

reducing bandwidth requirements

*simulation on application with ring communication pattern

V. Subotic et al. "Overlapping communication and computation by enforcing speculative data-flow", January 2008, HiPEAC

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
- Runnemede 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)

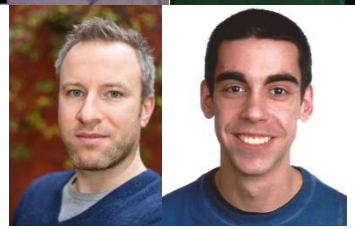
RoMoL … papers

- V. Marjanovic et al. “Effective communication and computation overlap with hybrid MPI/SMPSs.” **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 Trans. Parallel Distrib. Syst. 2010**
- V. Marjanovic et al. “Overlapping communication and computation by using a hybrid MPI/SMPSs approach.” **ICS 2010**
- T. Hayes, Oscar Palomar, Osman S. Unsal, Adrián Cristal, Mateo Valero: 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**
- L. Alvarez,et al. “Hardware-Software Coherence Protocol for the Coexistence of Caches and Local Memories.” **IEEE Trans. Comp. 2015**

RoMoL … papers

- 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 15. Candidate to the best paper award**

RoMoL Team

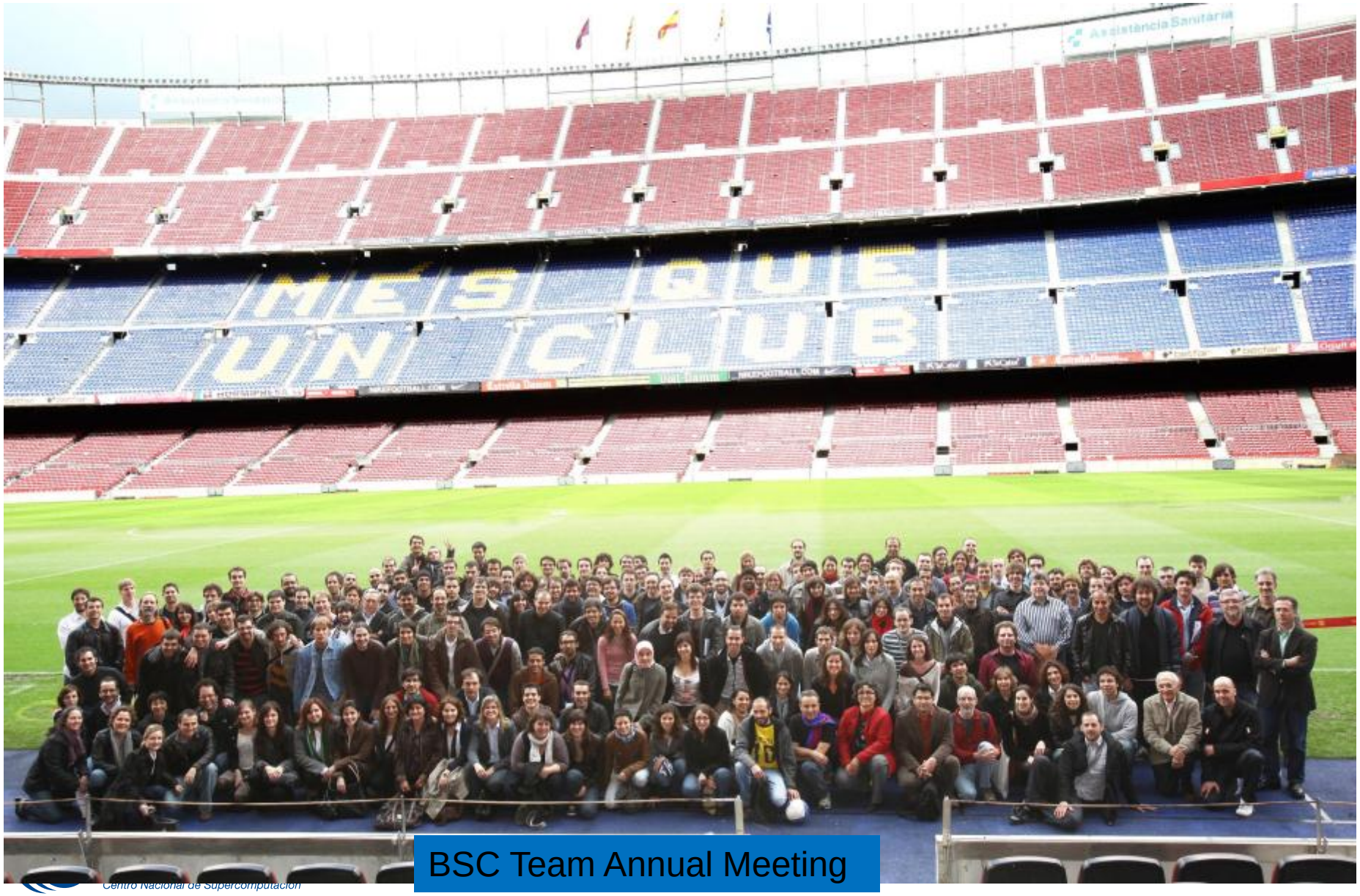
- Riding on Moore's Law (RoMoL, http://www.bsc.es/romol)
  - ERC Advanced Grant: 5-year project 2013 – 2018.
- Our team:
  - **CS Department @ BSC**
  - PI: 
  - Project Coordinators:  
  - Researchers:  
  - Staff:  
  - Students: 
- **Open for collaborations!** 

European Research Council
Established by the European Commission

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

RoMoL Project

Thank you!



BSC Team Annual Meeting

MareNostrum 3

Are we planning to upgrade?.. Negotiating our next site ;)


www.bsc.es
BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
THANK YOU!

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