From Classical to Runtime Aware Architectures

Prof. Mateo Valero
BSC Director
IEEE-CS Charles Babbage Award

In Recognition of Significant Contributions in the Field of Parallel Computing

Established in memory of Charles Babbage in recognition of significant contributions in the field of parallel computation. The candidate would have made an outstanding, innovative contribution or contributions to parallel computation. It is hoped, but not required, that the winner will have also contributed to the parallel computation community through teaching, mentoring, or community service.

Mateo Valero Named Recipient of 2017 IEEE Computer Society Charles Babbage Award

Citation: “contributions to parallel computation through brilliant technical work, mentoring PhD students, and building on incredibly productive European research environment.”
Once upon a time ...

1986 and 1988, UPC multiprocessor prototypes
Our Origins...


- IBM RS-6000 SP & IBM p630 192+144 Gflop/s
- SGI Altix 4700 819.2 Gflops
- SL8500 6 Petabytes
- Maricel 14.4 Tflops, 20 KW
- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- Our Origins...

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster

- IBM PP970 / Myrinet MareNostrum 42.35, 94.21 Tflop/s
- BULL NovaScale 5160 48 Gflop/s
- Compaq GS-160 32 Gflop/s
- Compaq GS-140 12.5 Gflop/s
- Parsys Multiprocessor 4.45 Gflop/s
- Parsytec CCI-8D 0.64 Gflop/s
- Convex C3800
- Connection Machine CM-200
- SGI Origin 2000
- Research prototypes
- Transputer cluster
**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**

<table>
<thead>
<tr>
<th>Consortium</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanish Government</td>
<td>60%</td>
</tr>
<tr>
<td>Catalan Government</td>
<td>30%</td>
</tr>
<tr>
<td>Univ. Politècnica de Catalunya (UPC)</td>
<td>10%</td>
</tr>
</tbody>
</table>
Mission of BSC Scientific Departments

**Computer Sciences**
To influence the way machines are built, programmed and used: computer architecture, programming models, performance tools, Big Data, Artificial Intelligence

**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 supercomputing capabilities (biomedical, geophysics, atmospheric, energy, social and economic simulations)
The MareNostrum 4 Supercomputer

Total peak performance: 13.7 Pflops/s

- More than 11 Pflops/s
- 12 times more powerful than MareNostrum 3

Compute:
- General Purpose, for current BSC workload
- More than 11 Pflops/s
- 3 times more than MareNostrum 3
- 12 times more than MareNostrum 3

Emerging Technologies, for evaluation of future Exascale systems
- More than 0.5 Pflops/s
- 3 systems, each of more than 0.5 Pflops/s

Storage:
- More than 10 PB of GPFS Elastics Storage System

Network:
- IB EDR/OPA Ethernet

Operating System:
- SuSE
Mare Nostrum 4
Design of Superscalar Processors

Decoupled from the software stack

Applications

ISA

Fetch → Decode → Rename → Instruction Window → Wakeup+ select → Register file → Bypass → Data Cache → Register Write → Commit

Programs “decoupled” from hardware

Simple interface Sequential program

CPU
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
  - Locality/reuse
    - Cache Memory With Hybrid Mapping (ASTED87). 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)
... 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
- 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

This image is the original one

Performance @ Low Power

Fuzzy Computation

Accuracy

Binary systems (bmp)

Compression protocols (jpeg)

This one only used ~85% of the time while consuming ~75% of the power
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 (cars, planes, trains, …)

**How to achieve it:**
- Controlling how resources are assigned to co-running tasks
- Soft real-time systems
- Hard real-time systems
  - Deterministic resource ‘securing’ (ISCA 2009)
  - Time-Randomised designs (DAC 2014 best paper award)
Vector Architectures... Memory Latency and Power 😊😊😊

- 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 (Micro 1997)
- SMT Vector Architectures (HPCA 1998)
- Architectures with Short Vectors (PACT 1997, ICS 1998)
- Tarantula (ISCA 2002), Knights Corner
- High-Speed Buffers Routers (Micro 2003, IEEE TC 2006)
- Vector Architectures for Data-Base (Micro 2012, HPCA2015, ISCA2016)
Statically scheduled VLIW architectures

- Power-efficient FU
- Clustering
- Widening (MICRO-98)
- $\mu$SIMD and multimedia units (ICPP-05)
- Locality-aware RF (SACK-94)
- Non-consistent (HPCAS-95)
- Two-level hierarchy (MICRO-00)
- Integrated modulo scheduling techniques, register allocation and spilling (MICRO-95, PACT-96, MICRO-96, MICRO-01)
The MultiCore Era

Moore’s Law + Memory Wall + Power Wall

Chip MultiProcessors (CMPs)
## How Multicores Were Designed at the Beginning?

<table>
<thead>
<tr>
<th>Model</th>
<th>Year</th>
<th>Cores</th>
<th>SMT</th>
<th>L2 Cache</th>
<th>L3 Cache</th>
<th>TDP</th>
<th>Mem BW</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IBM Power4 (2001)</strong></td>
<td></td>
<td>2</td>
<td>ST</td>
<td>0.7 MB/core</td>
<td>16 MB/core</td>
<td>115 W</td>
<td>10 GB/s</td>
</tr>
<tr>
<td><strong>IBM Power7 (2010)</strong></td>
<td></td>
<td>8</td>
<td>SMT4</td>
<td>256 KB/core</td>
<td>16 MB/core</td>
<td>170 W</td>
<td>100 GB/s</td>
</tr>
<tr>
<td><strong>IBM Power8 (2014)</strong></td>
<td></td>
<td>12</td>
<td>SMT8</td>
<td>512 KB/core</td>
<td>8 MB/core</td>
<td>250 W</td>
<td>410 GB/s</td>
</tr>
</tbody>
</table>
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
  • Deep and heterogeneous memory hierarchy
    • Non-Uniform Memory Access (NUMA)
    • Multiple address spaces
  • Stringent energy budget
  • Load Balancing

A Really Fuzzy Space

Programmability Wall
Living in the Programming Revolution

Multicores made the interface to leak...

Applications

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

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

Power to the runtime

PM: High-level, clean, abstract interface
History / Strategy

- PERMPAR ~1994
- GridSs ~2002
- CellSs ~2006
- SMPSs V1 ~2007
- SMPSs V2 ~2009
- COMPSs ~2007
- COMPSs ServiceSs ~2010
- GPUSs ~2009
- StarSs ~2008
- COMPSs ServiceSs PyCOMPSs ~2013
- DDT @ Parascope ~1992
- NANOS ~1996
- OmpSs ~2008

Forerunner of OpenMP

OpenMP...3.0...4.0...

2008 2013
OmpSs

A forerunner for OpenMP
OmpSs: data-flow execution of sequential programs

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

Decouple how we write applications from how they are executed

Write

Clean offloading to hide architectural complexities

Execute

#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);
OmpSs: ...Taskified...

```c
#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

```c
#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);
```

```c
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: a possible order of task execution

Decouple how we write form how it is executed

Write

Execute
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
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 # remanings

Killed transfers

Main Memory transfers (capacity)

Main memory transfers (cold)
Data Reuse @ Cell

- Experiments on the CellSs
  - Data Reuse
  - Locality arcs in dependence graph

- Good locality but high overhead → no time improvement

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)

Main memory:
- Cold
- Capacity

Global software cache

Local software cache

DMA Reads

P. Belens et al, “Making the Best of Temporal Locality: Just-In-Time Renaming and Lazy Write-Back on the Cell/B.E.” IJHPC 2010
GPUSs implementation

- **Architecture implications**
  - Large local store $O(\text{GB}) \rightarrow$ large task granularity \(\Leftarrow\) Good
  - Data transfers: Slow, non overlapped \(\Leftarrow\) 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

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
• 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
• E. Ayguadé,: “An Extension of the StarSs Programming Model for Platforms with Multiple GPUs.” Euro-Par 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. Dongarra et al, “The International Exascale Software Project roadmap” *IJHPCA 2011*
- V. Krishnan “OmpSs-OpenCL Programming Model for Heterogeneous Systems” *LCPC 2012*
- A. Fernández,”Task-Based Programming with OmpSs and Its Application.” *Euro-Par 2014*
- Many more since 2014…
The runtime **drives** the hardware design

- **Applications**
- **PM: High-level, clean, abstract interface**
- **Runtime**
- **ISA / 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

<table>
<thead>
<tr>
<th>Superscalar World</th>
<th>Multicore World</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out-of-Order, Kilo-Instruction Processor, Distant Parallelism</td>
<td>Task-based, Data-flow Graph, Dynamic Parallelism</td>
</tr>
<tr>
<td>Branch Predictor, Speculation</td>
<td>Tasks Output Prediction, Speculation</td>
</tr>
<tr>
<td>Fuzzy Computation</td>
<td></td>
</tr>
<tr>
<td>Dual Data Cache, Sack for VLIW</td>
<td>Hybrid Memory Hierarchy, NVM</td>
</tr>
<tr>
<td>Register Renaming, Virtual Regs</td>
<td>Late Task Memory Allocation</td>
</tr>
<tr>
<td>Cache Reuse, Prefetching, Victim Cache In-memory Computation</td>
<td>Data Reuse, Prefetching</td>
</tr>
<tr>
<td>AcCELERATORS, Different ISA’s, SMT</td>
<td>In-memory FU’s</td>
</tr>
<tr>
<td>Critical Path Exploitation</td>
<td>Heterogeneity of Tasks and HW</td>
</tr>
<tr>
<td>Resilience</td>
<td>Task-criticality</td>
</tr>
<tr>
<td>Memory Wall</td>
<td>Resilience</td>
</tr>
<tr>
<td>Power Wall</td>
<td></td>
</tr>
<tr>
<td>Programmability Wall</td>
<td>Load Balancing and Scheduling</td>
</tr>
<tr>
<td>Resilience Wall</td>
<td>Interconnection Network</td>
</tr>
<tr>
<td></td>
<td>Data Movement</td>
</tr>
</tbody>
</table>
Architecture Proposals in RoMoL

Runtime Support Unit
- DVFS
- Light-weight deps tracking
- Task memoization
- Reduced data motion

Cache Hierarchy
- LM usage
- Coherence
- Eviction policies
- Reductions

Vectors
- DB, sorting
- BTrees

Cluster Interconnect
- Priority-based arbitration
- By-pass routing
Runtime Management of Local Memories (LM)

LM Management in OmpSs
- 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 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
  • dramatically reduce coherence traffic (2.26x reduction with respect to the 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
  - dramatically reduce coherence traffic (2.26x reduction with respect to the state-of-the-art).

Graph Algorithms-Driven Partition (RIP-DEP)

Gauss-Seidel TDG

RIP-DEP requires ~90GB of data transfer across a 18-sockets (288 cores) system

I. Sánchez et al, Reducing Data Movements on Shared Memory Architectures (submitted to SC’17)
Runtime Managed Data Locality

• Leveraging runtime knowledge of the HW (NUMA topology) and the SW (task input data)

• Runtime manages co-location of data and computation (PACT’16):
  • NUMA Oblivious (DFT)
  • NUMA Aware Data Allocation only (DI)
  • NUMA Aware Data Allocation and Task Scheduling (NAFT)

- NAFT provides **best performance** (6.7x average speedup) and **lowest data movement** (4.0x average reduction) in a real 288 core ccNUMA SMP (16 sockets x 18 cores)

P. Caheny et al., “Reducing cache coherence traffic with hierarchical directory cache and NUMA-aware runtime scheduling.” PACT 2016
Runtime-Assisted Cache Insertion Policies

• Motivation: Improve LLC hit ratio to reduce costly requests to memory (EuroPAR’17)

• Use information about application semantics provided by the runtime:
  • Task types
  • Task data-dependency types (inputs, outputs, non-dependencies)

• Insertion policies based on Re-Reference Intervals
  • TTIP: uses probabilities per task-type to decide insertion position
    • Best probability is determined by training at the beginning of the execution
  • DTIP: gives output-dependencies a higher priority in the cache
    • Outputs will be reused by the successor task
    • Input- and non-dependencies lower priority

• Average MPKI improvement over LRU: 11.2% (TTIP) and 16.8% (DTIP)
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
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 reconfigures per-core DVFS meeting a global power budget
  - Architectural Support for DVFS: Runtime Support Unit (RSU)
    - Reduces reconfiguration overheads of software solution
      - Serialization in DVFS reconfigurations
      - User-kernel mode switches
    - Runtime system notifies to the RSU task criticality and running core
    - Similar hardware cost to TurboBoost

32-core system with 16 fast cores

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

Dealing with Manufacturing Variability in CPUs

- Manufacturing Variability of CPUs and Power becomes performance heterogeneity in power-constrained environments (ICS’16)
- Typical load-balancing may not be sufficient
- Redistributing power and number of active cores among sockets can improve performance

- Statically trying all possible configurations for each node imposes huge overhead (static).
- Runtime can try different configurations for a segment of the execution and choose a good one for the remaining time.
- Carefully limiting the configuration space to meaningful choices can greatly improve performance within a single run (exhaustive vs scoped).

D. Chasapis et al, “Runtime-Guided Mitigation of Manufacturing Variability in Power-Constrained Multi-Socket NUMA Nodes”. ICS’16
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 (IPDPS’17)


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
- **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
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/SMPSoSs Approach" ICS 2010
Effects on Bandwidth

flattening communication pattern

thus

reducing bandwidth requirements

*simulation on application with ring communication pattern

speedup of 1.6 for the same network bandwidth

for the same execution time 20 times lower needed bandwidth

OmpSs Runtime-based Resilience

• Suitability of OmpSs for resilience
  • Asynchrony – OoO execution, Input/output annotations

• Checkpoint – restart techniques (PDP’15)
  • Per-task inputs checkpointing, task replication to check outputs, asynchronous recovery tasks

• Algorithmic Recovery Routines (SC’15)
  • Conjugate Gradient (CG)
  • Detection
    • Memory Page Retirement
  • Correction
    • Algorithmic
  • Computation/Recovery overlap plus checkpointless techniques → low overhead


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 across LMs

• Runnemedeo 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
• 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/SMPSs approach.” ICS 2010
RoMoL ... papers

- 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., “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


• 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 Re-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.

• Our team:
  • CS Department @ BSC
  • PI:  Project Coordinators:
  • Researchers:  Postdocs:
  • Students:

• Open for collaborations!
Roadmaps to Exaflop

From K computer...
... to Post K
with domestic technology.

From Tianhe-2...
... to Tianhe-2A
with domestic technology.

From the PPP for HPC...
to future PRACE systems...
... with domestic technology?

IPCEI on HPC
HPC is a global competition

“The country with the strongest computing capability will host the world’s next scientific breakthroughs”.

US House Science, Space and Technology Committee Chairman
Lamar Smith (R-TX)

“Our goal is for Europe to become one of the top 3 world leaders in high-performance computing by 2020”.

European Commission President
Jean-Claude Juncker (27 October 2015)

“Europe can develop an exascale machine with ARM technology. Maybe we need an AIRBUS consortium for HPC and Big Data”.

Seymour Cray Award Ceremony  Nov. 2015
Mateo Valero
HPC: a disruptive technology for Industry

“The transformational impact of excellent science in research and innovation”

Final plenary panel at ICT - Innovate, Connect, Transform conference, 22 Oct 2015, Lisbon.

“...Europe has a unique opportunity to act and invest in the development and deployment of High Performance Computing (HPC) technology, Big Data and applications to ensure the competitiveness of its research and its industries.”

Günther Oettinger, Digital Economy & Society Commissioner
“Europe needs to develop an entire domestic exascale stack from the processor all the way to the system and application software”

Mateo Valero, Director of Barcelona Supercomputing Center

The transformational impact of excellent science in research and innovation

Final plenary panel at ICT - Innovate, Connect, Transform conference, 22 October 2015 Lisbon, Portugal.
# Mont-Blanc HPC Stack for ARM

## Industrial applications

- Pharmcelera™
- MUREX™
- Termo Fluids
- Cenoero

## Applications

- GENCI
- CINECA
- UNIGRAZ
- HLR
- University of Bristol

## System software

- ARM
- ETHzürich
- Inria

## Hardware

- ARM
- BSC
- Bull
- CEA
- UC
BSC Accelerator

512 RiscV cores in 64 clusters, 16GF/core: 8TF
4 HBM stacks (16GB, 1TB/s each): 64GB @ 4TB/s
16 custom SCM/Flash channels (1TB, 25GB/s each): 16TB @ 0.4TB/s

RISC-V ISA
Vector Unit
- 2048b vector
- 512b alu (4clk/op)
- 1 GHz @ Vmin

OOO

4w Fetch
- 64KB I$
- Decoupled I$/BP
- 2 level BP
- Loop Stream Detector

4w Rename/Retire
D$
- 64KB
- 64B/line
- 128 in-flight misses
- Hardware prefetch

1MB L2 per core
D$ to L2
- 1x512b read
- 1x512b write

L2 to mesh
- 1x512b read
- 1x512b write
Cluster holds snoop filter
HPC European strategy & Innovation

A window of opportunity is open:
- Basic industrial and scientific know-how is available
- Excellent funding opportunities exist in H2020 at European level and in the member state structural funds

It's time to invest in large Flagship projects for HPC to gain critical mass

Do we need an Airbus-type consortium for HPC and Big Data?

MareNostrum 3
Are we planning to upgrade?.. Negotiating our next site ;)

[Images of Sagrada Familia]
THANK YOU!

www.bsc.es