Research on MultiThreaded architectures at BSC-CNS

Francisco J. Cazorla
Senior PhD Researcher/Group Leader
francisco.cazorla@bsc.es

Jueves 31 de enero de 2008
Lo que muestro a continuación es tan solo una parte de la investigación en arquitecturas multi-hilo que se hace en el departamento de arquitectura de computadores del BSC.

Para un listado completo de los temas que se abarcan en dicho departamento visitar: http://www.bsc.es/plantillaF.php?cat_id=55
Agenda

- Motivation
- Implicit Resource Allocation
- Explicit Resource Allocation (ERA)
- ERA for High-Performance Systems
  - POWER5 Project (IBM)
- ERA for Real-Time Systems
  - MERASA Project
- Massive Multithreading architectures
  - Niagara T2 Project (Sun Microsystems)
- Conclusions
Basic equation to measure the performance of processors:

\[
\text{Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Cycle Time}
\]

(1) Data and control dependences limit the instruction-level parallelism (ILP).

- Extracting remaining IPC is too costly.
- The performance achievable by traditional superscalar processor designs has almost saturated.

(2) Processor frequency has also saturated.

- Power constraints, design constrains have reduced the increment in processor frequency.
Technology advances have increased the number of available transistors for processor designers.

However, it is difficult to extract more ILP from a single program.

Thread-level parallelism has become a common strategy for improving processor performance.

Multithreaded processors rely on using the additional transistors to obtain more parallelism by simultaneously executing several tasks.

A possible classification of MT processors:

- **SMT**: Intel Pentium 4 Xeon
- **CMP**: Intel dual-core, Intel quad-core, IBM POWER4
- **SMT/CMP**: IBM POWER5, IBM POWER6
- **FMT/CMP**: Sun Niagara T1, Sun Niagara T2
Multi-Threaded Processors

- MT processors are used in a wide range of computing systems:
  - High-performance: IBM Power5, Intel/AMD Quad-Core
  - Real-Time: Infineon Tricore, Imagination Meta
  - Network: Sun Niagara T1, Sun Niagara T2

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cores</th>
<th>Ways</th>
<th>L2 Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core Duo</td>
<td>2</td>
<td>8/16</td>
<td>2MB L2 cache</td>
</tr>
<tr>
<td>Niagara T2</td>
<td>8</td>
<td>8</td>
<td>3MB L2 cache</td>
</tr>
<tr>
<td>Power5</td>
<td>2</td>
<td>2</td>
<td>2MB L2 cache</td>
</tr>
</tbody>
</table>
Some of the main reasons for this wide use of MT processors are the following:

- MT processor has good performance/cost ratio
  - Simple cores are replicated on the chip
  - Easy to verify
  - Easier to design than complex cores
- MT processors have a good performance/energy consumption ratio
  - The watt per committed instruction rate is better in MT processor than in super-scalar processors
One of the main differences between ST and MT processors is resource sharing.

- ST: only one thread uses all processor resources.
- MT: Several threads share the processor resources at the same time.
- The degree of sharing changes between processor types.
- **Super-Scalar**: Only 1 thread uses resources at a time.

- **CMP**: Core is not shared. High-levels of the cache are can be.

- **CMT**: On a long-latency event execution is switched to a new thread (Blocking Multithreading).
FMT: On short-latency event the execution changes to another thread

SMT: Every cycle all threads running share most of the resources.

Agenda

- Motivation
- Implicit resource allocation
- Explicit Resource Allocation
- ERA for High-Performance Systems
  - POWER5 Project (IBM)
- ERA for Real-Time Systems
  - MERASA Project
- Massive Multithreading architectures
  - Niagara T2 Project (Sun Microsystems)
- Conclusions
Resource Sharing

- It represents one of the main difference between ST processors and MT processors.

- Good news about Resource Sharing:
  - Provides higher-performance by allowing threads make a better use of resources.
  - Lower energy consumption.

- Bad news about Resource Sharing:
  - Execution Time Unpredictability
    - At execution time running threads dynamically share processor resources.
    - Before hand, we do not know the amount of resources that each thread will receive.
    - As a consequence, we cannot estimate the execution time of programs.
  - Let's see the effect of Unpredictability with an example.
Performance Unpredictability

- Let’s say that in your laptop you are running several applications at the same time.
  - Text processing application
  - Playing a movie

- In order to properly display the movie let’s say we need a rate of 50 frames per second.
  - $\frac{1}{50} \rightarrow$ every 20ms 1 frame has to be processed
  - Something similar happens with the audio

- In a Teleconference we have four of these real-time flows
This chart shows gzip’s IPC when executed in an 4-context SMT in different workloads.

There is a large variability in its IPC.
The underlying problem

- OS perceives contexts of an MT as…
  - Independent processing units (PUs)
  - These units are assumed to have similar resources

- However, in an SMT…
  - Contexts are not truly independent
  - Contexts share same resources
  - Threads interact between them
The underlying problem

- In a MT processor there are several hardware mechanisms that determine how to prioritize threads
  - Example: The cache replace policy like LRU

- This resource allocation is not under the supervision of the OS
  - This is what we call **Implicit Resource Allocation**

- OS runs two tasks in a MT processor.
  - One with very high priority and
  - The other with low priority

- If the low-priority task misses more frequently in cache than the high-priority, it will get LRU priority reducing the performance of the other thread
OS/MT architecture may have opposing objectives

- Hardware mechanisms mainly focused on improving throughput

- Hardware mechanisms lack flexibility:
  - They assign resources to threads in order to meet its own objectives rather than those of the OS.
  - If it would serve its purpose to allocate fewer resources to a OS high-priority thread, hardware mechanisms would do so.

- In SMTs Ifetch policy *implicitly* controls resource allocation
  - Assumes part of the responsibility of the OS in prioritizing jobs
  - Ifetch policy designed to fulfill its own objectives: Throughput

- In CMPs the same happens with the Cache replacement policy
  - Optimized to reduce number of misses $\rightarrow$ throughput

- Hardware mechanisms objectives may be opposed to OS objectives
Several problems because of Implicit Resource Allocation:

- This main one: **priority inversion**:
  - A program with higher OS-priority is given less resources than a program with lower OS-priority
  - OS cannot control the execution speed of threads

- Performance lost
Our objective

- Each computing system has its own objectives (requirements)
  - Real-time: execute tasks before a given deadline
  - High-performance: increase throughput

- We call these objectives, *Quality of Service requirements*

- Our target:
  - Can we design a MT processor able to deal with all these requirements?
  - Can MT processors provide QoS?
  - How can we design an MT to do so?
Agenda

- Motivation

- Implicit resource allocation

- **Explicit Resource Allocation**

- ERA for High-Performance Systems
  - POWER5 Project (IBM)

- ERA for Real-Time Systems
  - MERASA Project

- Massive Multithreading architectures
  - Niagara T2 Project (Sun Microsystems)

- Conclusions
ERA: Explicit Resource Allocation

- What if we explicitly control resource allocation?
  - Directly controlling resource management
    - In addition to resource-usage indicators: L1, L2 data cache misses
  - Making the OS aware of resource allocation
    - Resource allocation is made explicit to the OS

- Ideally we want a hardware/software mechanism that enables the OS to satisfy system requirements
  - Control the execution of certain critical threads
  - Improve throughput
  - ...
OS and MT processor talk different languages

- The Operating System Job Scheduler works with ‘time constraints’
  - 70% of execution time → 70% of total performance
  - Time a task is scheduled onto a CPU

- MT hardware works with ‘resource allocation’:
  - Percentage of resources given to a thread

- How do we translate time into resource allocation?
  - Definition of an interface between the OS and the SMT

![Diagram of the interface between the OS and the SMT processor]
Currently the OS (job scheduler) just composes the workload

In our proposal, the MT processor provides ‘levers’ through which the OS can control the processor internal resource allocation.
Agenda

- Motivation
- Implicit resource allocation
- Explicit Resource Allocation

- ERA for High-Performance Systems
  - POWER5 Project (IBM)

- ERA for Real-Time Systems
  - MERASA Project (EU FP7)

- Massive Multithreading architectures
  - Niagara T2 Project (Sun Microsystems)

- Conclusions
Explicit Resource Allocation in HPC

- Collaborative project between IBM and BSC:

- Objectives:
  - Analyze the interaction between the operating system and the IBM POWER5 processor
    - Interaction between the OS priorities and hardware priorities
  - Understand the effect of the IBM POWER5 hardware prioritization on performance
POWER 5

- 2-core CMP processor where each core is 2-context SMT

- Each core has a hardware prioritization mechanism that the OS can modify
  - The higher the priority of a thread the more resources it receives
  - Prioritization levels: 7, 6, 5, 4, 3, 2, 1, 0

<table>
<thead>
<tr>
<th>Priority difference</th>
<th>R</th>
<th>Decode cycles for P</th>
<th>Decode cycles for S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>31</td>
<td>1</td>
</tr>
</tbody>
</table>
MareNostrum Applications

- MareNostrum Applications are:
  - Parallel Applications
  - MPI
  - Applications are SPMD (Single Program Multiple Data)
  - It is a common situation that applications have some kind of barrier where all applications communicate each other
  - Applications are repetitive
The problem

- Like MareNostrum applications, many HPC applications are SPMD (Single Process-Multi Data)
  - Each process in the parallel application executes the same code on different data set
  - Synchronization points (barriers, collective operations, etc.) ensure the correct result

- In theory all the processes should complete their tasks at the same time:
  - In this case it does not really make sense to prioritize one process respect to another, since they are supposed to execute the same code with the same performance
  - Anyhow, processes operate on different data (let's say a block of a sparse matrix) and they may require different time to complete their execution
  - Still, the synchronization points (barriers, collective operations, etc.) hold

- In practice, the execution time of each process could be different

- We could use the hardware priority capabilities to reduce this “application-intrinsic” imbalancing
HPC Applications

Which are the sources of imbalancing?

- **Input set**: data to be processed by each thread is different
  - i.e., when processing a matrix each process receives a sub-matrix and the matrix is sparse

- **Domain**: solution is divided in sub-domains, each process receive a sub-domain and some sub-domains require more computation power than others

- **OS noise/user daemons**: in some computing nodes the OS or user daemons could delay the running process

- **Network topology**: exchanging data in the same node takes shorter than between nodes
Sparse linear algebra:

- The application is executing some operation using sparse linear algebra (for example it is using a sparse matrix)
- All the processes perform the same operation on their piece of the matrix
- The execution time depends on the number and distribution of not-null values in the sparse matrix

- Each process receives a block of the matrix
- Some process could receive blocks with few or no elements inside
  - The execution of those processes is much faster but they need to wait for the other in order to continue with the next iteration

Non-null values
(Picture from [3])
Iterative methods

- Iterative methods try to solve a problem (for example a Partial Differential Equation, PDE) computing an approximation of the solution and the error made during the computation

  - The solution computed at step i, $S_i$, is combined with the error computed in the same step, $E_i$, to compute $S_{i+1}$

  - The application stops when the error is less than a minimum threshold (i.e., the solution is accurate)

- Every process receives a sub-domain and has to compute the solution in this sub-domain

  - The function can have picks in some sub-domains and be smooth in others

  - It is easier (i.e., faster) to compute an accurate values if the function is smooth in a sub-domain

    - The solution converges earlier (with less iterations)

- The application is unbalanced because of domain/function characteristics
Iterative methods (2)

(Pictures from [3])
Border data

- In order to compute the value \( f(x) \) the process A may require the values of the function computed around \( x \).
  - If \( x \) is a point on the border of the data set computed by A, some required value could involve several communications with other processes.
  - While the process A is waiting for the data, it is not running.

- The required data can be:
  - On the same node (several MPI processes running on the same SMP node)
  - On near neighbours
  - On far neighbours

- The delay introduced by the data exchange depends on the distance from the sending node.

(Picture from [3])
Large clusters

- When scaling on large cluster the application unbalancing could dramatically reduce performance.

(Picture from [1])
Related Work

- Several approaches have proposed to reduce imbalance

- Dynamic data distribution:
  - Open-MP: loop scheduling algorithms assign iterations to threads dynamically
    - Big granularity
    - In a 8-core node, assigning 1 thread implies a variation of 12.5% in the computational power
  - MPI: more complex, because communications are defined explicitly by the programmers

- Dynamic Resource distribution: Assigning more resources to those threads computing longer
  - OpenMP: assign more threads to those groups computing longer
  - MPI: this process is more complex as the number of processes is statically assigned

- Our proposal: Use ERA to reduce imbalance
ERA in HPC Computing

Recently IBM POWER5 processor introduce such an ERA mechanism

- POWER5 is a dual-core where each core is 2-thread SMT

- POWER5 offers 7 levels of priorities for each hardware thread controllable by the software

- In each core, priorities control the rate at which each thread decodes instructions
  - Higher priority $\rightarrow$ Higher decode rate
  - Higher decode rate $\rightarrow$ Higher performance

<table>
<thead>
<tr>
<th>Priority</th>
<th>Priority level</th>
<th>Privilege level</th>
<th>or-nop inst.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Thread shut off</td>
<td>Hypervisor</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Very low</td>
<td>Supervisor</td>
<td>31,31,31</td>
</tr>
<tr>
<td>2</td>
<td>Low</td>
<td>User/Supervisor</td>
<td>1,1,1</td>
</tr>
<tr>
<td>3</td>
<td>Medium-Low</td>
<td>User/Supervisor</td>
<td>6,6,6</td>
</tr>
<tr>
<td>4</td>
<td>Medium</td>
<td>User/Supervisor</td>
<td>2,2,2</td>
</tr>
<tr>
<td>5</td>
<td>Medium-high</td>
<td>Supervisor</td>
<td>5,5,5</td>
</tr>
<tr>
<td>6</td>
<td>High</td>
<td>Supervisor</td>
<td>3,3,3</td>
</tr>
<tr>
<td>7</td>
<td>Very high</td>
<td>Hypervisor</td>
<td>7,7,7</td>
</tr>
</tbody>
</table>

$$R = 2^{|PrioP - PrioS|} + 1.$$
ERA in HPC Computing

- Assume a 4 process MPI application running on a POWER5
  - Further assume that P1 computes longer than P2, P3, P4
  - P1, P2 run on one core and P3, P4 in the other core
- By increasing P1’s priority the application execution time decreases

(a) Unbalanced HPC application
(b) More balanced HPC application
Our proposal: increase the priority of the threads executing longer

This solution fits in the ‘dynamic resource distribution’ group

The advantages with respect to previous solution is that is granularity

- Processor resource allocation provides a finer granularity than assigning/deassigning threads
- It is transparent to the user
- One-time effort

Experiment:

- We run a real a 4-process unbalanced application on a POWER5 processor executing Linux
- We played with priorities as see the effect on its performance

### ERA in HPC Computing

- **Initialization Phase**
- **Dark-grey** → computation
- **Light-grey** → communication
- **Iterative Behavior**
- **Great Imbalance**
- **P4 is the bottleneck of the application**

<table>
<thead>
<tr>
<th>Test</th>
<th>Proc</th>
<th>Core</th>
<th>P</th>
<th>Comp %</th>
<th>Sync %</th>
<th>Imb %</th>
<th>Exec. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>P1</td>
<td>1</td>
<td>7</td>
<td>17.63</td>
<td>82.32</td>
<td></td>
<td>82.88s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>7</td>
<td>28.91</td>
<td>71.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>P1</td>
<td>1</td>
<td>4</td>
<td>52.33</td>
<td>47.49</td>
<td>70.93</td>
<td>127.91s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>3</td>
<td>99.64</td>
<td>0.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>6</td>
<td>53.78</td>
<td>46.11</td>
<td>14.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>1</td>
<td>6</td>
<td>85.88</td>
<td>14.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>P1</td>
<td>1</td>
<td>3</td>
<td>65.32</td>
<td>34.48</td>
<td>45.99</td>
<td>75.62s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>4</td>
<td>99.68</td>
<td>0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>6</td>
<td>53.78</td>
<td>46.11</td>
<td>14.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>1</td>
<td>6</td>
<td>85.88</td>
<td>14.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>P1</td>
<td>1</td>
<td>4</td>
<td>82.73</td>
<td>17.10</td>
<td>33.38</td>
<td>66.88s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>4</td>
<td>73.68</td>
<td>26.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>5</td>
<td>66.40</td>
<td>33.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>1</td>
<td>6</td>
<td>99.72</td>
<td>0.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>P1</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ERA in HPC Computing

- P1, P4 same core
- P2, P3 same core
- Priority 4 to P1 and P2 and
- Priority 6 to P3 and P4
- Imbalanced has been reduced from 82% to 46%
- ET improvement of 7%

<table>
<thead>
<tr>
<th>Test</th>
<th>Proc</th>
<th>Core</th>
<th>P</th>
<th>Comp %</th>
<th>Sync %</th>
<th>Imb %</th>
<th>Exec. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>P1</td>
<td>1</td>
<td>7</td>
<td>17.63</td>
<td>82.32</td>
<td></td>
<td>82.88s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>P1</td>
<td>1</td>
<td>4</td>
<td>28.91</td>
<td>71.02</td>
<td>82.23</td>
<td>81.64s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>4</td>
<td>66.47</td>
<td>33.4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>4</td>
<td>99.72</td>
<td>0.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>P1</td>
<td>1</td>
<td>3</td>
<td>52.33</td>
<td>47.49</td>
<td>70.93</td>
<td>127.91s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>3</td>
<td>99.64</td>
<td>0.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>6</td>
<td>28.87</td>
<td>71.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>2</td>
<td>6</td>
<td>46.26</td>
<td>53.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P1</td>
<td>1</td>
<td>6</td>
<td>53.78</td>
<td>46.11</td>
<td>45.99</td>
<td>75.62s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>6</td>
<td>99.68</td>
<td>0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>6</td>
<td>85.88</td>
<td>14.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>P1</td>
<td>1</td>
<td>4</td>
<td>65.32</td>
<td>34.48</td>
<td>45.99</td>
<td>75.62s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>4</td>
<td>99.68</td>
<td>0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>6</td>
<td>53.78</td>
<td>46.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>1</td>
<td>6</td>
<td>85.88</td>
<td>14.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>P1</td>
<td>1</td>
<td>4</td>
<td>82.73</td>
<td>17.10</td>
<td>33.38</td>
<td>66.88s</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>2</td>
<td>4</td>
<td>73.68</td>
<td>26.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>2</td>
<td>5</td>
<td>66.40</td>
<td>33.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>1</td>
<td>6</td>
<td>99.72</td>
<td>0.09</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agenda

- Motivation
- Implicit resource allocation
- Explicit Resource Allocation
- ERA for High-Performance Systems
  - POWER5 Project (IBM)
- ERA for Real-Time Systems
  - MERASA Project
- Massive Multithreading architectures
  - Niagara T2 Project (Sun Microsystems)
- Conclusions
Performance Unpredictability

This chart shows gzip’s IPC when executed in an 4-context SMT in different workloads.

There is a large variability in its IPC: from 1 to 2.5.
ERA in Real-Time Systems

- MT processors presents the problem of performance unpredictability execution

- Why we should use MT processors in **Soft/Hard** real-time systems?

- Increasing demand for functionality in current and future real-time embedded systems is driving an increase in performance of processors

- We need a solution, with the following characteristics:
  
  - Does not increase CPU clock speeds → High Frequency → High consumption
  
  - Maintains low chip costs → This prevents from us using of High-performance cores
  
  - Low power consumption etc → Increase Battery Life

- MT architectures represent such a solution
Standard single-thread real-time scheduler:

- We focus on systems with periodic task sets
- CTs executed periodically, period P (1 frame every 4ms)
- Deadline, d: any instance of a task must be executed before next instance (d=P)
- Worst Case Execution Time (WCET) (2 ms)
  - Computed assuming access to all resources
  - At run time, thread will access only to a part of the resources

\[
S = \frac{d}{WCET} = 2 \quad p = \frac{1}{S} = 0.5
\]
ERA in Real-Time Systems

- How the OS translate from time to resource allocation?
  - Exploits performance/resource relation of applications
  - This relation is: Not linear, variable

![Graph showing the relationship between relative IPC and amount of resources](image)

40% resources $\rightarrow$ 70% full speed

Super-linear relation

$X\%$ resources $\rightarrow$ $\geq X\%$ performance
We work with 2-thread workloads:

- 1 critical-thread (CT): Thread with time constraints
  - MediaBench benchmark suite

- 1 non-critical thread (NCT): Thread with no time constraints, but high resource requirements
  - SPEC CPU 2000 benchmark suite
  - Makes more difficult to accomplish with the time constraints of CT

Objective:

- CT meets its deadline and high performance for the NCT
Low-Variability Performance

- We execute pairs of threads (CT, NCTᵢ)

- Objective: Reduce the variability in the Execution Time of the CT

- In this experiment we run the CT against 8 different NCT

- By controlling MT processor internal resource allocation we obtain a low variability

(a) IPC when we control the fetch (gray bars), and the fetch, the IQs and regs. (black bars).
Predictable Performance

- In this experiment we run the CT against different NCT in the same MT processor
- Controlling MT internal resource allocation, we obtain a high success rate

- We obtain a good performance for the non-critical task
Soft and Hard Real-Time Systems

- So far we have talked about soft real-time systems
- This type of systems allow some deadlines to be missed
  - Playing a movie
  - Cell phones
- There are another kind of systems where deadlines cannot be missed, hard real-time systems
  - Automotive
  - Space
  - Airplanes
Processors in use today in embedded applications with hard real-time requirements are:

- characterised by simpler architectures than desktop processors,
- encompassing short pipelines and in-order execution.

This is required to make the computation of the WCET easier.

However, there is a growing requirement for real-time systems to host applications with increasingly high data throughput rates.

Meeting these demands means that processors must, crucially, be able to provide higher performance than current embedded processors.
The challenge

- Simple architectures
  - Ease the computation of WCET
  - Do not provide the performance required by current and future real-time systems

- MT architectures
  - Provide the throughput required by current and future real-time systems
  - Make the computation of the WCET complex

- There is a clear trade-off between analysability and performance
MERASA

- European STREP Project under the umbrella of FP7
  - Multi-Core Execution of Hard Real-Time Applications Supporting Analysability
  - www.merasa.org

- There are 5 participants:
  - UAU – University of Augsburg - Germany
  - UPS – University Paul Sabatier - France
  - RPT - Rapita Systems - UK
  - HON – Honeywell spol. s.r.o. - Czech Republic
  - BSC – Barcelona Supercomputing Center - Spain

- Objective: allow the use of MT processor in systems with hard-real time constrains
Providing higher performance than what state-of-the-art embedded processors can deliver today will increase:

- safety, comfort, number and quality of services, lower emissions and fuel demands

Automotive applications:

- Controls for internal combustion engines → Lower emissions
- Automatic emergency-braking triggered by collision avoidance techniques
- ABS

Aerospace/Space applications:

- Support for ever increasing demands of additional functionality on board or a higher comfort by a better control of the actuators

Construction machinery applications
High-performance processors are not suitable for real-time environments:

- Too expensive
- Consume a lot of energy

We need processors providing high-performance with low cost, low power consumption.

MT processors seem a possible solution but we have to deal with the unpredictability wall.
Agenda

- Motivation
- Implicit resource allocation
- Explicit Resource Allocation
- ERA for High-Performance Systems
  - POWER5 Project (IBM)
- ERA for Real-Time Systems
  - MERASA Project
- Massive Multithreading architectures
  - Niagara T2 Project (Sun Microsystems)
- Conclusions
Massive Multithreading Architectures

- In this type of architectures the TLP is more important than the ILP
  - The IPC of each thread is not the main concern
  - The main concern is to improve total throughput (performance of all threads)
  - These architectures support more in-flight threads than other architectures

- Normally used in networking
  - Web Server:
    - The latency of the network makes per-thread performance a less important metric
    - What really matters is how many threads (requests) can be served in parallel
    - Threads normally are short (thousand of instructions)
  - Routing
    - Target metric number of packets that can be processed per second
Massive Multithreading Architectures

- Chips can allocate ‘a lot of’ threads at the same time

- Example: Sun Niagara T2
  - 8 cores
  - 8 threads per core
  - 64 threads in total

Niagara T2

http://www.opensparc.net/
In this type of processor the binding of threads to strands is critical

- Threads have different resource-utilization patterns
- We should put in the same core threads using different ‘sets’ of resources

Example:

- Let’s assume a 2-core chip each core having 4 threads
- Also assume a workload of 5 threads (T0, T1, T2, T3, T4)
- There are 15 possible ways of assigning these 5 threads to the cores
We can put:

- 4 threads in one core and 1 in the other (4,1)
- 3 threads in one core and 2 in the other (3,2)

**Group (4,1):**

{\{T0 T1 T2 T3\}, \{T4\}}
{\{T0 T1 T2 T4\}, \{T3\}}
{\{T0 T1 T3 T4\}, \{T2\}}
{\{T0 T2 T3 T4\}, \{T1\}}
{\{T1 T2 T3 T4\}, \{T0\}}

**Group (3,2):**

{\{T0 T1 T2\}, \{T3 T4\}}
{\{T0 T1 T3\}, \{T2 T4\}}
{\{T0 T1 T4\}, \{T2 T3\}}
{\{T0 T2 T3\}, \{T1 T4\}}
{\{T0 T2 T4\}, \{T1 T3\}}
{\{T0 T3 T4\}, \{T1 T2\}}
{\{T1 T2 T3\}, \{T0 T4\}}
{\{T1 T2 T4\}, \{T0 T3\}}
{\{T1 T3 T4\}, \{T0 T2\}}
{\{T2 T3 T4\}, \{T0 T1\}}
If the architecture is homogeneous:

- \((4,1) = (1,4)\)
- \((3,2) = (2,3)\)
The number of combinations increase exponentially.
Massive Multithreading Architectures

- The number of combinations increase exponentially.
The large number of combinations makes it impossible to use a Brute Force Approach

Different heuristics metrics have to be used in order to provide a good scheduling

This is one of the targets of Collaborative project between Sun and BSC
Agenda

- Motivation
- Implicit resource allocation
- Explicit Resource Allocation
- ERA for High-Performance Systems
  - POWER5 Project (IBM)
- ERA for Real-Time Systems
  - MERASA Project
- Massive Multithreading architectures
  - Niagara T2 Project (Sun Microsystems)
- Conclusions
Conclusions

- MT architecture are here to stay

- They are used in a wide-range of computing systems
  - High-performance: IBM Power5, Intel Pentium 4 Xeon, Intel Quad-Core
  - Real-Time: Infineon Tricore, Imagination Meta
  - Network: Sun Niagara T2

- MT architectures are an open research area in computer architecture
BSC-CNS had 170 members at the end of 2006 and hailed from 21 different countries:

- Argentina
- Mexico
- Belgium
- Lebanon
- Brazil
- Poland
- Bulgaria
- Russia
- Colombia
- Serbia
- China
- Turkey
- Cuba
- UK
- France
- USA
- Germany
- Italy
- India
- Spain
- Ireland
References


Research on MultiThreaded architectures at BSC-CNS

Francisco J. Cazorla  
Senior PhD Researcher/Group Leader  
francisco.cazorla@bsc.es  

Jueves 31 de enero de 2008