ADAPTABILITY TECHNIQUES FOR QOS AND DISTRIBUTED MANAGEMENT OF MPSOCS
Adaptability techniques for QoS and distributed management of MPSoCs

- Introduction
- MPSoC Architecture
- Monitoring and “QoS API”
- QoS Techniques
- Hierarchical Management
- Publications
Introduction

MPSoCs with dozens or hundreds of PEs interconnected through NoCs is a reality in current design of embedded systems.

Our research focus:

- General purpose MPSoC – dynamic workload with applications starting at any time
- Hardware and software infrastructure may have mechanisms to ensure at runtime QoS requirements
What are our MPSoC assumptions?

- **Homogeneous MPSoC (SMP)**
  - Simplifies tasks mapping, task migration, code generation

- **NoC as the communication infrastructure**
  - Scalability and parallelism

- **Processing Element with one processor**
  - Avoids complex clustered architectures (more than one processor per node)
  - Cost of the router is small

- **Message passing communication**
  - No shared memory
  - Reduced traffic into the communication infrastructure

- **Memory organization**
  - Only local memories (scratchpad memory)
  - Paged memory
Why such assumptions?

Research in several topics related to MPSoCs

- Task mapping / migration
- Fault-tolerance
- NoC architectures
- Abstract Modeling
- Power management (aging, reliability)
- Monitoring and QoS
- System management
Adaptability

- **Self-adaptation**: system is able to change at runtime its behavior according to the application requirements
  - this capability is as a key feature to meet runtime application requirements in dynamic systems

- Examples of self-adaptation techniques
  - priority-based task scheduling
  - communication priorities
  - task migration
  - circuit switching
  - DVFS

- These techniques are managed by a **monitoring** mechanism
Outline

Adaptability techniques for QoS and distributed management of MPSoCs

- Introduction
- MPSoC Architecture
- Monitoring and “QoS API”
- QoS Techniques
- Hierarchical Management
- Publications
MPSoC Architecture

- Homogeneous MPSoC

HeMPS

Plasma-IP SL

Task Repository

Modified Hermes NoC

Plasma-IP SL

Router

PLASMA

Network Interface

DMA

RAM

Plasma-IP SL

Plasma-IP SL

Plasma-IP MP
The MPSoC contains two types of PEs:

- **Manager PE**
  - system management, mapping, monitoring, etc.
  - access to the external devices (e.g. the application repository)
  - Do not executes user tasks

- **Slave PEs**
  - responsible for task execution
PE Architecture

- **Network Interface**
  - Interface with the NoC – decouples communication from computation

- **Plasma Processor**
  - MIPS architecture

- **DMA**
  - Direct access to the memory without passing through the processor

- **Memory**
  - scratchpad memory
  - true dual port
  - contains the kernel and the application's tasks

- **Router**
  - belongs to the PE
Task Repository

- External memory
  - Contains
    - Application description (list of tasks, dependencies, code size, …)
    - All object codes of the tasks to be executed in the system
NoC

- 2D-mesh NoC, with the following features:
  - **duplicated** 16-bit physical channels, assigning high priority to channel 0 and low priority to channel 1 (high priority packets may use both channels)
  - deterministic **Hamiltonian** routing in channel 1 and partially adaptive Hamiltonian routing in channel 0
  - input buffering
  - credit-based flow control
  - **simultaneous** PS and CS
  - support to multi-cast communication
NoC

- **Router**
  - **Red resources** *(channel 0)*
    - High priority packets
  - **Blue resources** *(channel 1)*
    - High/Low priority packets
    - Increases the support to the high priority traffic
  - **Local ports**
    - Channel 0: Connections
    - Channel 1: Priorities
NoC

- Hamiltonian path
- two acyclic paths
NoC

- Partially adaptive version of the routing
- Used for high priority flows
NoC

- Deterministic version of the routing
- Used for low priority flows
Adaptability techniques for QoS and distributed management of MPSoCs

- Introduction
- MPSoC Architecture
- Monitoring and “QoS API”
- QoS Techniques
- Hierarchical Management
- Publications
Hybrid Monitoring requirement
- Small intrusion, i.e., minimal disturbing in the applications’ performance
- Hardware and software impl.

Packet Monitor (at Slave PEs)
- Implemented in hardware
- Extracts information from packets, sending them to the manager PE (size, timestamp)
- Only packets from RT applications are transmitted to the MP

Throughput and latency monitors
- Implemented in software in the manager PE
Performance Monitors

- **Throughput Monitor**
  - bits per packet are extracted from the packet sent by the packet monitor
  - at the end of the **window monitoring** the throughput is computed and compared to the specified flow requirement

- **Latency Monitor**
  - the latency is computed according to the timestamps sent by the packet monitor
  - there is no window monitoring for latency monitoring

- **For both monitors**
  - **Violation**: when a throughput or latency deadline is not respect
  - **Event**: after \( n \) violations the QoS manager is notified
Evaluation of the monitoring approach

- Our hybrid approach (hw+sw) was compared to a pure software implementation.

**Execution time**

- Hybrid monitoring + 1.9%
- Software monitoring + 10.6%

**Link utilization**

- For all RT-packets a packet is transmitted to the manager PE.

- Small performance penalty
- Small link usage

A packet is transmitted to the manager PE when an event occurs.

6x6 MPSoC with nine RT applications instances: 3 MJPEG, 1 audio_video, 3 DTW, and 2 FFT.
QoS API

- The API enables the monitoring between a pair of communicating tasks
- Constraints: defined by the application and verified with the application running alone in the platform (**profiling**)

```c
/* Called in consumer task: define for real time task the resolution of each iteration, default is 1 millisecond */
setRTResolution(uint producer, uint resolution_time)

/* Called in consumer task: create a new communicant task pair, and setup latency and throughput deadline to them */
setQoSProducer(uint producer, uint lat_deadline, uint thr_deadline)

/* Called in producer task: informs to kernel of producer task the consumer task pair */
setQoSConsumer(uint consumer)
```
These functions are *system calls* that:

- send the QoS parameters to the manager PE
- program the local *packet monitor* to monitor all packets of the task that executed the QoS function

```c
/* Called in consumer task: define for real time task the resolution of each iteration, default is 1 millisecond */
setRTResolution(uint producer, uint resolution_time)

/* Called in consumer task: *create a new communicant task pair, and setup latency and throughput deadline to them* */
setQoSProducer(uint producer, uint lat_deadline, uint thr_deadline)

/* Called in producer task: informs to kernel of producer task the consumer task pair */
setQoSConsumer(uint consumer)
```
QoS Manager

Summary of the method

- User defines the application requirements (QoS API)
- The packet monitor monitors the application behavior at the hardware level (NI)
- Those informations are sent to the manager PE which apply a high level monitoring (SW)
- If deadlines are missed the QoS Manager is called (event), the following actions may be taken:
  1. change the task scheduling priority
  2. increase the task time-slice
  3. change the application communication priority
  4. start task migration
  5. change the switching mode
Outline

Adaptability techniques for QoS and distributed management of MPSoCs

- Introduction
- MPSoC Architecture
- Monitoring and “QoS API”
- QoS Techniques
- Hierarchical Management
- Publications
Adaptive Management Techniques

- 4 techniques are employed

<table>
<thead>
<tr>
<th>Technique</th>
<th>Computation</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow priority</td>
<td>-</td>
<td>application traffic is prioritized</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– soft guarantee</td>
</tr>
<tr>
<td>Circuit Switching</td>
<td>-</td>
<td>exclusive reservation of network resources</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- hard guarantee</td>
</tr>
<tr>
<td>Task Scheduling</td>
<td>increase the CPU time, by reducing the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>processor sharing among several tasks</td>
<td></td>
</tr>
<tr>
<td>Task Migration</td>
<td>move a given task to a free PE (load</td>
<td>move a task to a non-</td>
</tr>
<tr>
<td></td>
<td>balancing)</td>
<td>congested NoC region</td>
</tr>
</tbody>
</table>
Scheduling (1)

- Round-Robin Scheduling with Priority-Based Preemption
  - Priority defines the time-slice (amount of clock cycles a task execute)

- A suspended task may preempt a task with lower priority
  - A preempted task has its remaining time-slice stored in its TCB, which is restored when the task is re-scheduled

- If an application is not meeting required performance
  - MP sends a multicast packet to all applications tasks, increasing the priority and the time-slice (+50%) of each application task
Communication Priority (2)

- Default communication mode:
  - low priority and deterministic routing

- If an application is not meeting required performance
  - the MP increases the flow priority of the communicating pair
  - the flow returns to low priority when deadlines are not missed in a parameterizable time window
Task Migration (3)

- **Task migration**
  - If multiple tasks share a same PE, a given task can be moved to improve performance.
  - Moving a task near to its communicating tasks reduces communication energy and packet latency.

- **Task migration features**
  - Complete task migration, including code, data and context.
  - The task may be migrated at any moment — no migration checkpoints required.
  - In-order message delivery, i.e., tasks communicating with the migrated tasks will receive messages in the order they were created.
Task Migration (3)

QoS Manager sends a TASK_MIGRATION order with a target processor ID

A MIGRATION_CODE packet is sent to target processor with object code of TASK_A

1. QoS Manager
2. SOURCE PROCESSOR
3. TARGET PROCESSOR
4. SOURCE PROCESSOR
5. TARGET PROCESSOR

SOURCE PROCESSOR
- TASK_B ready
- TASK_A running
- MICROKERNEL

TARGET PROCESSOR
- FREE_PAGE
- TASK_C
- MICROKERNEL

SOURCE PROCESSOR
- TASK_B ready
- TASK_C
- MICROKERNEL

TARGET PROCESSOR
- TASK_A running
- MICROKERNEL

SOURCE PROCESSOR
- TASK_B ready
- MICROKERNEL

TARGET PROCESSOR
- TASK_C
- MICROKERNEL

SOURCE PROCESSOR
- TASK_A running
- MICROKERNEL

TARGET PROCESSOR
- MICROKERNEL

SOURCE PROCESSOR

TARGET PROCESSOR

TASK_A keeps running until the condition to migrate the data memory is satisfied

Data memory of TASK_A (bss, data and stack) is migrated to PAGE 2 in target processor
Established using channel 0

Reserves a path between the producer and consumer PEs

All packets produced in a given switching mode must be consumed before changing the switching mode

Originality: the MP keeps a data structure to manage the connections, avoiding waste time trying to establish connections
Flow and CPU Adaptation

- Monitoring events determine the system state

- Latency events:
  - Initial Stage
  - Performance
  - Critical State

- Throughput events:
  - Initial Stage
  - Performance
  - Critical State

- Flow adaptation:
  - High Priority Flow
  - Circuit Switching

- CPU adaptation:
  - Task Migration
  - Scheduling Priority
Latency events trigger flow adaptations
- increased latency is mostly due to congestion

First action:
- Increase the flow priority of the communicating pair

Second action:
- Change the switching mode for CS
- Ensure maximum throughput, without jitter
struct {  // communicating pair structure  
  char mode; // { CS, HIGH, LOW }  
  uint timer; // value to keep a communication mode  
  uint CSt; // circuit switch timer value  
  uint FCt; // flow counter timer value  
  int producer; // task that generate packets  
  int consume; // task that receives packets  
} comm_pair;

main_OS_master() {  
  initialization functions  
  while(1) {  
    if( system_time > last_monitored_time + WINDOW_LENGHT)  
      Time_Out_Monitor()  
    
    if( received_noc_packet(pk)) {  
      switch(pk.service) {  
        case QoS_event: Monitor_Event_Handler(pk)  
          ...  
        ...  
      }  
    }  
  }  
}

Time_Out_Monitor() {  
  forall communicating pairs (comm_pair) {  
    if (comm_pair.mode = LOW) continue;  
    else if (comm_pair.mode=CS AND comm_pair.timer > comm_pair.CSt)) {  
      Send_Flow_Adaptation_Packet(comm_pair.producer, HIGH)  
      comm_pair.timer = 0  
    }  
    else if (comm_pair.mode=HIGH AND comm_pair.timer > comm_pair.FCt)) {  
      Send_Flow_Adaptation_Packet(comm_pair.producer, LOW)  
      comm_pair.timer = 0  
    }  
    else  
      comm_pair.timer += WINDOW_LENGHT;  
  }

Monitor_Event_Handler (input: incoming packet pk){  
  comm_pair = get_communicating_pair(pk.producer, pk.consumer)  
  comm_pair.timer = 0  
  switch(pk.event){  
    case LATENCY:  
      if (comm_pair.mode=HIGH AND Path_CS_Available(comm_pair)=TRUE)  
        Send_Flow_Adaptation_Packet(comm_pair.producer, CS)  
      else if (comm_pair.mode=LOW)  
        Send_Flow_Adaptation_Packet(comm_pair.producer, HIGH)  
    case THROUGHPUT:  
      if (Path_CS_Available(comm_pair)=TRUE )  
        Send_Flow_Adaptation_Packet(comm_pair.producer, CS)  
      else if (comm_pair.mode=LOW)  
        Send_Flow_Adaptation_Packet(comm_pair.producer, HIGH)  
  }  
}
Increasing the network priority

- **Changing the network priority** for a given flow
  - The OS of the producer task enable or disable the priority field

- Each packet contains in its header the network priority
  - The router reads the packet and insert it in the appropriate channel (0/1)
Changing the switching mode

- **Problem:** the QoS management must avoid CS flows using the same channels
  - Before the adaptation order, an algorithm verifies if some flow intersect with the new flow
  - If not, the CS is established by sending an order to the OS of the producer task

- All PS packets must be consumed before passing to CS (and vice-versa)
Flow adaptation

```java
case LATENCY:
    if (comm_pair.mode=HIGH AND Path_CS_Available(comm_pair)=TRUE)
        Send_Flow_Adaptation_Packet(comm_pair.producer, CS)
    else if (comm_pair.mode=LOW)
        Send_Flow_Adaptation_Packet(comm_pair.producer, HIGH)

case THROUGHPUT:
    if (Path_CS_Available(comm_pair)=TRUE )
        Send_Flow_Adaptation_Packet(comm_pair.producer, CS)
    else if (comm_pair.mode=LOW)
        Send_Flow_Adaptation_Packet(comm_pair.producer, HIGH)
```

- **Time_Out_Monitor ()**
  - Responsible for restoring the flow priority when no event is received
  - In CS defines the period in which the flow remains in this switching mode

- procedure that checks in a lookup table the feasibility of CS
- table has all CS paths
- complexity is \( \Theta (n) \)
Flow adaptation – synthetic applications

SR2 latency without adaptation

SR2 latency with adaptation
Flow adaptation — synthetic applications

Throughput for the SR2 application, without and with QoS adaptation
Flow adaptation – real benchmarks

violations: reduced from 131 to 50
jitter: 13 of the 50 violations have a latency superior to 10% of the deadline

Latency without adaptation (IDCT → PRINT)

Latency with adaptation (IDCT → PRINT)
Flow adaptation – real benchmarks

Dynamic flow adaptation

- Start Disturbing Applications
- Maximum Throughput to MJPEG and DTW
- End of Applications

- START → IVLC
- IVLC → IQUANT
- IQUANT → IDCT
- IDCT → PRINT
- BANK → P1
- BANK → P2
- BANK → P3
- BANK → P4

TIME

- Low Priority Flow
- High Priority Flow
- Circuit Switching

0 ms

380,8 ms
Flow adaptation – real benchmarks

- CS established to ensure hard RT

Graph showing latency over time with CS establishment and deadline markers.
Throughput events trigger CPU adaptation
- reduced throughput is mostly due to CPU sharing or non-optimal mapping

Acts indirectly over the QoS application
- Instead to migrate the RT task, the BE tasks are migrated
- Minimize the effect of the BE tasks disturbing the RT application performance
CPU Adaptation

First action: move BE tasks by migration

Second action: Increase the scheduler priority

Third action: migrate the RT task

Fourth action: no space in the cluster without sharing the CPU with RT tasks – increase scheduler priority
(1) and (2) BE tasks shares resources with RT tasks, the BE tasks are migrated to PEs with no others RT tasks. The throughput is restored.

(3) Cluster become full, an BE application ends and releases resources to migrate the BE task which are sharing resources with the RT task. The throughput is restored.
Application DTW (bank → p2)

1. The monitoring detects throughput - task p2 migrates to a free PE
2. A BE task starts, reducing the throughput. The BE task is migrated
3. Other disturbing tasks starts, the scheduling priority increases
Flow adaptation
- Adaptive techniques provided QoS guarantees
- The use of CS using a software algorithm avoids attempts to establish connections
- Hard RT applications can extend CS up to the end of the application

CPU Adaptation
- Showed an important throughput improvement in congested scenarios
- The techniques tries to map RT applications in one processor, while grouping BE tasks
- Good performance of the task migration method
Adaptability techniques for QoS and distributed management of MPSoCs

- Introduction
- MPSoC Architecture
- Monitoring and “QoS API”
- QoS Techniques
- Hierarchical Management
- Publications
Introduction

• The constant growth in the number of cores implies an important issue: scalability

• Despite the scalability offered by NoCs and distributed processing, the MPSoC resources must be managed to deliver the expected performance

• An alternative to ensure scalability is to decentralize or distribute the management functions of the system
The distributed resource management assumes an MPSoC divided in $n$ regions, named clusters.
The MPSoC contains three types of PEs

- **Global Master PE (GMP):**
  - contains all functions of the LMP, and functions related to the overall system management
  - only PE with access to the external devices (e.g., the application repository)

- **Local Master PEs (LMP):** responsible to control the cluster

- **Slave PEs (SP):** responsible for task execution
Reclustering

- According to user requests, new applications can be loaded at runtime.
- If an application does not fit in a given cluster, the LMP of the cluster may request resources to neighbor clusters.
Task Migration

- Used to restore the cluster size
  - When resources became available, the foreign tasks may migrate
Total Execution Time results

- Execution time normalized w.r.t the centralized management in a 12x12 MPSoC, with an MPSoC load equal to 75%
- Distributed management leads to a total execution time reduction
- The reduction in the total execution time reduction comes from:
  - Several PEs execute the task mapping in parallel
  - Each manager treats a smaller number of control packets compared to the centralized approach

<table>
<thead>
<tr>
<th>Cluster Size</th>
<th>Nb of Clusters</th>
<th>Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPEG</td>
<td>Synthetic</td>
</tr>
<tr>
<td>12x12</td>
<td>1,00</td>
<td>1,00</td>
</tr>
<tr>
<td>12x6</td>
<td>0,94</td>
<td>0,78</td>
</tr>
<tr>
<td>6x6</td>
<td>0,90</td>
<td>0,67</td>
</tr>
<tr>
<td>6x4</td>
<td>0,88</td>
<td>0,58</td>
</tr>
<tr>
<td>6x3</td>
<td>0,86</td>
<td>0,57</td>
</tr>
<tr>
<td>4x4</td>
<td>0,88</td>
<td>0,58</td>
</tr>
<tr>
<td>3x3</td>
<td>0,87</td>
<td>0,54</td>
</tr>
</tbody>
</table>

MPSoC, described in RTL cycle accurate modeling (SystemC)
The evaluation of the average hop number is a key parameter to evaluate the mapping quality.

Higher values of hop number on the other side penalize the performance of applications, since disturbing traffic may interfere in the communication.

<table>
<thead>
<tr>
<th>Cluster Size</th>
<th>Nb of Clusters</th>
<th>MPEG (5 tasks)</th>
<th>Synthetic (6 tasks)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>min</td>
<td>avg</td>
</tr>
<tr>
<td>12x12</td>
<td>1</td>
<td>4</td>
<td>5.14</td>
</tr>
<tr>
<td>6x3</td>
<td>8</td>
<td>4</td>
<td>4.05</td>
</tr>
<tr>
<td>4x4</td>
<td>9</td>
<td>4</td>
<td>4.45</td>
</tr>
</tbody>
</table>
Reclustering results

- Evaluates monitoring with task migration. Two scenarios were evaluated:
  - main application with disturbing applications, without task migration
  - main application with disturbing applications and two task migrations (tasks E and D)
- Execution time reduction of 2.67%, considering two task migrations
  - Even if task migration momentarily increases the execution time, the final result is an improvement in the overall performance
Distributed management conclusion

- The distributed management technique reduced the distance among tasks, resulting in an important reduction in the total execution time.

- Task migration is an effective adaptive method to improve the system performance.
Outline

Adaptability techniques for QoS and distributed management of MPSoCs

- Introduction
- MPSoC Architecture
- Monitoring and “QoS API”
- QoS Techniques
- Hierarchical Management
- Publications
Main publications related to QoS

A Self-adaptable Distributed DFS Scheme for NoC-based MPSoCs
In: SBCCI, 2011

Dynamic Task Mapping for MPSoCs
CARVALHO, Ewerson; CALAZANS, Ney; MORAES, Fernando Gehm
http://dx.doi.org/10.1109/MDT.2010.106

Adaptive QoS Techniques for NoC-Based MPSoCs
RUARO, M.; CARARA, E.; MORAES, F.G.
In: SOC, 2013

Differentiated Communication Services for NoC-Based MPSoCs
CARARA, E.; CALAZANS, N.; MORAES, Fernando Gehm
http://dx.doi.org/10.1109/tc.2012.123

Runtime Adaptive Circuit-Switching and Flow Priority in NoC-Based MPSoCs
RUARO, R.; CARARA, Everton Alceu; MORAES, Fernando Gehm
IEEE Transactions on Very Large Scale Integration (VLSI) Systems, PREPRINT, 2014
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6849463&queryText%3DRuntime +Adaptive+Circuit-Switching+and+Flow+Priority+in+NoC-Based+MPSoCs
Publications related to distr. manag.

A Framework for MPSoC Generation and Distributed Applications Evaluation
CASTILHOS, Guilherme ; WACHTER, Eduardo ; MADALOZZO, Guilherme ; ERICHSEN, Augusto ; MONTEIRO, Thiago ; MORAES, Fernando Gehm

Distributed Resource Management in NoC-Based MPSoCs with Dynamic Cluster Sizes
CASTILHOS, Guilherme, MANDELLI, Marcelo, MADALOZZO, Guilherme, MORAES, Fernando Gehm
In: ISVLSI, 2013, pp. 153-158

Enhancing Performance of MPSoCs through Distributed Resource Management
MANDELLI, Marcelo ; CASTILHOS, Guilherme ; MORAES, Fernando Gehm

Proposal and Evaluation of a Task Migration Protocol for NoC-based MPSoCs
MORAES, Fernando Gehm ; MADALOZZO, Guilherme ; CASTILHOS, Guilherme ; CARARA, Everton Alceu

HeMPS - A Framework for NoC-Based MPSoC Generation
CARARA,E.; OLIVEIRA,R.; CALAZANS,N.; MORAES,F.G.
In: ISCAS, 2009
Thanks to my PhD students

Eduardo Wachter  Marcelo Mandelli  Guilherme Madalozzo  Guilherme Castilhos  Marcelo Ruaro

Professor Everton Carara (former PhD)

Thank you for your attention!