A Monitoring System for NoCs

Leandro Fiorin
ALaRI, Faculty of Informatics - University of Lugano
Lugano, Switzerland

Gianluca Palermo, Cristina Silvano
Politecnico di Milano - DEI
Milano, Italy

NoCArc'10 – December 4, 2010, Atlanta, Georgia, USA
Outline

- Motivations
- Monitoring in NoCs
- Our contributions
- Event categories
- Monitoring architecture
  - Programmable probes
  - Data management & collection
- Experimental results
- Conclusions and future work
Motivations

- Next generation MPSoC platforms will integrate a large number of processing cores, storage elements, and I/O peripherals, interconnected by NoCs.

- A high number of complex concurrent applications will share available resources providing users with new services and functionalities.

- Platform-based design will allow to reduce the cost per single item by giving the system the possibility to easily adapt to different application requirements.
Motivations

How can we exploit efficiently available resources?

How can we understand system behaviour?

New tools are needed for helping designers in these tasks, exploiting information derived by measurements taken on the running system.
Motivations

- Modern, high performance processors use dedicated on-chip hardware event detectors and counters

- Performance Counters are hw registers dedicated to counting events within the processor or system

- Each register has an associated control register that tells it what to count and how to do it

Taken from “Philip J. Mucci, Hardware Performance Analysis on the Opteron with PAPI ClusterWorld 2004, San Jose, CA”
Monitoring in NoCs

- NoCs monitoring was proposed for:

  **Debugging**

  **Testing**

  **Detecting congestion**

  **Platform run-time management**

  **Security**
Our contributions

- to perform a comprehensive study of the most common events in NoCs
- to propose the utilization of a multipurpose programmable monitoring probe
- to propose and discuss an efficient and automatic collection and storage of the information related to the events detected, and to evaluate the intrusiveness of the components and activities of the monitoring system
- we propose an architecture for a monitoring system for NoCs
- while mainly focusing on performance tuning, the system could be easily adapted to provide information useful for debugging, run-time management of system resources, and security
Event categories

- We focus on events of cores and NoC resources related to the communication system
  - Throughput characterization
  - Timing and Latency
  - Resources utilization
  - NoC Events and Messages characteristics
Monitoring architecture

- Probes
- Probes Management Unit
- Data collection and storage
Programmable probe

- Event detector
- Accumulator
- Preprocessing modules
- Configuration registers
- Message generator
- Output queue
Event detectors

- The event detector observe OCP/IP, and NI, and router signals and monitor events selected by the configuration registers.

- We use a programmable multipurpose probe, able to monitor all the events of the system.

- Depending on the area budget, several multipurpose probes can be deployed for each NI.

- Event detectors operate in parallel with NI kernel, not interfering with its operations (not intrusiveness).
Event detectors

- Throughput detector
- Keeps track of incoming/outgoing traffic
- Choice of connections
- Choice of period of collection

- Timing/Latency detector
- Measures time proprieties of transactions
- Different types of measurements: I2I, I2T, EXEC, T2I
- Collaboration between probes at initiator and target
- Collection for different transactions and connections
Event detectors

- Resources utilization detector
  - Monitors status of internal queue of NI and router

- Message characteristics detector
  - Detects user configuration events
  - NoC configuration events
Data Preprocessing

- We implement the possibility to pre-process data for reducing traffic

- Time windows
  - Messages sent at the end of time window
  - Generated using 32 bit counter

- Threshold
  - Messages generated only if $>$, $<$, $=$, $=>$, $<=$ of threshold value
  - Only critical information is sent

- Average calculation
  - Values of samples are collected during the execution, together with number of occurrences
  - Values sent at the end of collection
Message generator e Probes configuration

- The Message generator creates packets to be sent to the PMU
- Data collection triggered at the end of the time frame or for occurrence of events
- It acts as initiator, writing in memory address associated to during configuration
- Possibility to aggregate data, reducing traffic generated of up to 92%
- Configuration registers are memory mapped to the PMU
- PMU keeps track of all the configurations
Intrusiveness of monitoring system should be limited in collection and storage

We performed an analysis of bandwidth needed by each probe

<table>
<thead>
<tr>
<th>Event</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput for execution</td>
<td>$\frac{64 \text{bits}}{\text{execution time}}$</td>
</tr>
<tr>
<td>Throughput for time window</td>
<td>$\frac{64 \text{bits}}{\text{time_window}}$</td>
</tr>
<tr>
<td>T2I latency for transaction</td>
<td>$\frac{64 \text{bits} \times \text{number_transactions}}{\text{execution time}}$</td>
</tr>
<tr>
<td>T2I latency above threshold</td>
<td>$\max\left(\frac{64 \text{bits} \times \text{number_transactions}}{\text{execution time}}\right)$</td>
</tr>
<tr>
<td>queue utilization</td>
<td>$\frac{64 \text{bits}}{\text{queue_sampling_time} \times \alpha}$</td>
</tr>
<tr>
<td>Average queue utilization in time window</td>
<td>$\frac{96 \text{bits}}{\text{time_window}}$</td>
</tr>
</tbody>
</table>
Data collection

- **PMU Local memory**
  - Used for event generating a limited number of messages for execution
  - Fast access to information
  - Local storage important for analysis of run-time system behaviour and adaptive systems

- **Streaming memory**
  - For data exceeding allocated space in PMU local memory, and for data with unknown dimension
  - All the message packet is stored, and retrieved when elaborated
 Probe Management Unit

- Programs the configuration registers (before execution)
- Retrieves and elaborates collected data (after execution)
- These tasks can be implemented as software routines (no overhead associated)
- For run-time management, a third task should be active during execution in order to implementing adaptivity based on the information detected
Experimental results

<table>
<thead>
<tr>
<th></th>
<th>Area (mm^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single multipurpose probe</td>
<td>0.042</td>
</tr>
<tr>
<td>Timing/latency probe</td>
<td>0.037</td>
</tr>
<tr>
<td>4 multipurpose probes</td>
<td>0.156</td>
</tr>
<tr>
<td>6 multipurpose probes</td>
<td>0.231</td>
</tr>
<tr>
<td>Full monitoring system</td>
<td>0.577</td>
</tr>
<tr>
<td>Router (5 p)</td>
<td>0.143</td>
</tr>
<tr>
<td>NI</td>
<td>0.141</td>
</tr>
<tr>
<td>ARM920T</td>
<td>4.7</td>
</tr>
</tbody>
</table>

- We implemented the monitoring system and synthesized with Synopsys, using a 0.13um technology library, and targeting 500MHz.
- Adding reconfigurability (multipurpose) costs around 13%.
- For 4 multipurpose probes, we save around 73% with respect to complete monitoring system.
- 4 probes counts for around 35% of area NI (buffers long 8)+router (buffers long 4) (generated for architecture with 10 initiator and 1 target).
- Overhead with regard to NoC elements for 4 probes is around 55%, while if we consider also a typical embedded processor (ARM920T), it is 3%.
Experiments

- We implement a cycle accurate simulator in SystemC, where generation of transactions is driven by memory requests generated by the application.
- As use case, we considered a *ray tracing* application.
- We consider an architecture with 10 initiators and one L2 shared memory, mapped on a 4 x 3 mesh (L2 in [1,1]).
- 4 probes for each tile.
- We measure:
  - total traffic from each initiator
  - Average I2I latency
  - Number of times I2I over threshold (95 cycles)
  - Throughput generated in each time window.
Experiments
Experiments

<table>
<thead>
<tr>
<th>Event</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>0.26 KB/s</td>
</tr>
<tr>
<td>Average I2I latency</td>
<td>0.39 KB/s</td>
</tr>
<tr>
<td>I2I latency over threshold</td>
<td>4.62 MB/s</td>
</tr>
<tr>
<td>Throughput in time window</td>
<td>119 KB/s</td>
</tr>
</tbody>
</table>

- By using the threshold functionality the number of messages is reduced in average of the 91%

- The traffic generated by the probes is around the 5% of the traffic generated by the application, and 0.2% of the link bandwidth

- Assumption about non intrusiveness verified in the experiment
Conclusions and future work

- We approached the problem of monitoring NoC based systems
- We performed a comprehensive study on the type of events
- We propose a programmable multipurpose probe able to detect a large number of events, with a relatively small overhead

Future work will focus on the analysis of other possible monitoring functionalities, to the exploration of alternative for the data collection, and the implementation of tools for management of configuration registers and analysis of collected data.
Thanks for your attention!

fiorin@alari.ch