Proposal for Noc benchmarking methodology

Erno Salminen, Ari Kulmala
DACI research group
Tampere University of Technology
02.01.2007
Noc WG goals (from white paper)

- Benchmarking environment should exercise
  - network size (small, medium, large)
  - IP core composition (amount of processing, memory cores, other)
  - Topology (regular, irregular)
  - Traffic characteristics (spatial and temporal)
  - QoS requirements

- Benchmark types
  - programs or program models (e.g. H.263)
  - micro-benchmarks (e.g. uniform traffic)
Proposed methodology

- Separate
  a) application functionality
  b) mapping of application onto resources
  c) hardware resources
  d) benchmarked Noc

- Application model is fixed in benchmark suite
  - Includes dependencies

- Architecture and mapping may be modified during measurements in some cases
Proposed methodology (2)

- Application model (a) is interpreted by traffic generator (c) that models the resources.
- Not all transfers directed to Noc.
- E.g. transfer:
  - A \rightarrow B via Noc
  - E \rightarrow C internal to PE3
- They have different cost.
Application model

- Model similar to Kahn Process Network (KPN)
  - Vertices represent computation tasks
  - Edges represent communication channels
  - Channels carry the data tokens between tasks
- Model includes dependencies, timing, destination, and size of data transfers
- No actual computation, only external behavior of application tasks
  - Unlike approaches that include also computation in KPN (e.g. Artemis by Pimentel et al.)
- The task is triggered according to a condition that
  - depends on the received data tokens
  - and internal state of the task
Application model (2)

- Internal state of the task is expressed with
  - an execution counter
  - and a status variable: {RUN, READY, WAIT, FREE}.

- The dependence of the multiple inputs can be either
  a) OR: triggered when any of the input receives a token
  b) AND: triggered when there is data in all inputs
  - Note: trad. KPN uses only AND dependency

- A function of a task defines the
  - execution time in cycles
  - output port(s) (their ID numbers)
  - data amount to send in bytes
Application model (3)

- A trigger includes one or several functions
- The function that is evaluated depends on the value of the execution counter.
- The execution time and data amount is expressed with either
  a) a statistical distribution (uniform or normal)
  b) a polynomial function which depend on the received data amount (scalar value is subset of this)
- Events (timers) can be used for environment modeling
- Some tasks may model memory accesses instead of computation
Possible task models

- Statistical
  - e.g. destinations:
    - send to task 5 with 10% prob, to task 4 with 90% prob and so on
  - most compact format
  - Either set probability to 100% or neglect dependencies to avoid locking

- Modulo (model loops)
  - e.g. two destinations:
    - to task 5 at every 10th iteration (modulo)
    - to task 4 otherwise

- Trace
  - runtime and data amount can be different for each execution
  - large file
Resource model

- Very coarse-grain
- Modeled by traffic generator
- PE has a performance of "X operations/cycle"
  - Possibly also area, power, max. frequency etc.
  - For NoC benchmarking, all resources may be identical (e.g. 1 ops/cycle)
  - Heterogeneous models needed for design space exploration and SoC modeling
- PEs may have different frequencies
- Example
  - task \( i \) has 500 operations
  - PEs \( A \) and \( B \) can execute 1 and 2 ops/cycle
  - task \( i \) takes
    - 500 cycles when mapped to PE \( A \) (=500/1)
    - 250 cycles when mapped to PE \( B \) (=500/2)
Proposed file format

- Based on XML (eXtensible Markup Language)
  - ASCII text with user-specified tags
    - Larger file size but more reader-friendly than binary format
  - Format rules defined with schema file
  - Free parsers available for C, Tcl, Python
  - Free validators also available

- Benchmark tools convert XML to their internal format if needed

- Application, architecture, and mapping may be separate or combined into single file
Example of task

- Triggered when data at input port 2
- Condition is the same for all exec counts
- Always sends 384 bytes to outport 0
  - function: 0x + 384
- Complexity is always 13571 integer operations
  - function: 0x + 13571
- Process is not finished for ever, but can be executed again (state=ready)
Example of task (2)

- Two inports: 7 and 21
  - 7 receives data from outport 6 (of another task), 21 from port 20

- Two outports: 11 and 20

- Event "VideoInput" executes periodically every 0.028 seconds, it writes 1 byte to outport 8
  - Also "one-shot" events possible

- Number of events, tasks, ports, or connections is not restricted
  - 32-bit integers allow 4e9 unique IDs
Task mapping

- Suitable mapping depends on
  - task and PE type
  - PE load (runtime)
  - induced communication
  - required memory vs available memory

- Tasks with (class=general) can be freely mapped to general purpose resources (CPUs)

- HW accelerator resource can execute only type of tasks
  - External interfaces (e.g. radio interface) are similar
**Transaction generator**

- SystemC program that models the PEs
- Converts operator counts to time
- Sends, receives, and checks data
- Schedules the tasks mapped to same PE
  - Non-preemptive policy
  - FIFO, priority, round-robin

*Fig. 30. An overview of Transaction Generator implementation.*

*Fig. 31. Block diagram of the communication network class.*

Institute of Digital and Computer Systems
Modeling DMA

- Direct memory access allows
  - To send data to network while PE is processing
  - To receive data while processing
- Of course, DMA can be disabled
  - Processing not possible during the transfers

Fig. 32. Processing element model for Transaction Generator with and without DMA modeling.
Task state machine

- After reset, state usually *Wait*
- Data token (tg_packet) structure shown in Fig 34
- An internal transfer
  - communication between tasks located in the same PE
  - pass-by-reference (pointer)
- External transfers
  - carried out through an OCP TL2 interface
  - master port is utilized for outgoing, slave port for incoming transfers.

![Task execution scheduling state diagram.](image)

**Fig. 33.** Task execution scheduling state diagram.

```c
class tg_packet {
    unsigned int transfer_id; // transfer id
    unsigned int source_id; // source process id
    unsigned int target_id; // target process id
    unsigned int n_bytes; // amount of payload data
    unsigned char* data; // pointer to payload
}
```

**Fig. 34.** The packet structure of a data transfer.
Statistics

- Some general statistics are collected automatically
- These are complemented with Noc-specific metrics

**Table 9. Statistics produced by Transaction Generator.**

<table>
<thead>
<tr>
<th>Processing element</th>
<th>Task</th>
<th>Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task execution count</td>
<td>Execution count</td>
<td>Packet amount</td>
</tr>
<tr>
<td>Task execution time</td>
<td>Time in READY state</td>
<td>Min transfer latency</td>
</tr>
<tr>
<td>Read/write time</td>
<td>Time in RUN state</td>
<td>Avg transfer latency</td>
</tr>
<tr>
<td>Read/write packet count</td>
<td>Time in WAIT state</td>
<td>Max transfer latency</td>
</tr>
<tr>
<td>Read/write packet header amount</td>
<td>Time in FREE state</td>
<td></td>
</tr>
<tr>
<td>Read/write packet payload amount</td>
<td>Last execution time</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 35. Block diagram of the task class.

Additionally: required buffering
Performance/real-time requirements

- Somewhat open subject currently
- E.g. define limits for certain paths
  - max time "task A starts executing" -> "task F has sent all the data to Noc" is 3.5 ms (= application runtime)
  - max time "A starts sending data" -> "B has received all the data from Noc" 0.6 ms
  - time limits in absolute time (1ms after reset this has happened, after 5 ms that has happened...)
  - These can be automatically checked
    - Define criticality level for requirements
      - Stop at violation
      - Continue despite
- If separate to application description
  - "Clean" application model
  - Problematic if data should be sent in different manner depending on requirements
  - Problematic if scheduler at the PE must react to requirements
Accuracy

- Seems adequate for network comparison
- Not as issue with purely synthetic workloads

Fig. 57. The clock cycle results obtained with XSM compared to ISS with Seamless for INTRA and INTER frames.
"Traditional" traffic model

- No dependencies
  - PEs might send results before receiving source data
- Traffic generated either
  a) directly to Noc at runtime
  b) to file at compile-time
  - The latter (b) allows exact reproduction easily
- PE type, task grouping and scheduling fixed
  - However, PE location within Noc may still be changed
- Not easy to determine PE load or application runtime
- Suitable for environment modeling (noise traffic, external IO)
Benefits of proposed methodology

- Trad. case with one functional model per resource
  - Suitable for environment, HW accelerator modeling (e.g. DCT) and for micro-benchmarks
  - Not suitable for CPU modeling
    - multithreading and context switch overhead?

- Proposed method is a superset of trad. and allows
  - modeling dependencies between tasks
  - more tasks than resources: one resource may execute several tasks in time-multiplexed manner
  - CPU (+RTOS) modeling
  - real trace or statistical (e.g. avg. runtime) runtime
  - design space exploration (allocation, mapping, scheduling, Noc params)
  - also micro-benchmarks
    - tasks have self-loop, i.e. they trigger themselves repeatedly
      - same as neglecting dependencies
Some related publications


