Application-Platform Mapping in Multiprocessor Systems-on-Chip

Leandro Soares Indrusiak

lsi@cs.york.ac.uk
http://www-users.cs.york.ac.uk/lsi
Outline

- Motivation
- Application-Platform Mapping
- Multiprocessor System-on-Chip Platforms
- Application Modelling
- System Validation
- Conclusions and Future Work
Motivation

- Application-Platform mapping isn’t really a problem when dealing with sequential software and uniprocessor hardware.
Motivation

- It has been studied for decades by researchers in parallel and distributed computing.
Motivation

- CMOS limits can be felt in many ways
  - number of transistors
  - clock frequency
  - power dissipation

- Expectations of performance increase must be met somehow
  - multiprocessing seems to be the best shot
Motivation

- Application-Platform mapping is a critical issue when it comes to multiprocessor platforms.
Motivation

Sources: Gordon ASPLOS’06, Kudlur PLDI’08
Motivation

- Application-Platform mapping is a critical issue when it comes to multiprocessor platforms must be able to explore concurrency at the application level
Motivation

- Application-Platform mapping is a critical issue when it comes to multiprocessor platforms can be done dynamically to improve performance or to increase dependability.
Application-Platform Mapping

- The simplest formulation of the mapping problem resembles a graph isomorphism problem
  - application is a graph $G = G(A, C)$, where $a_i \in A$ is an application task and $c_{i,j} \in C$ represents the communication from $a_i$ to $a_j$
  - platform is a graph $G' = G(B, D)$, where $b_i \in B$ is a processor and $d_{i,j} \in D$ represents the channel from $b_i$ to $b_j$
  - objective is to map tasks onto processors such that task that communicate with each other lie on adjacent processors

- There is no known polynomial time solution for the graph isomorphism problem
Application-Platform Mapping

- More sophisticated problem formulations may include additional information to the application and platform graphs, so that different objectives can be met.

**platform:**
- processing power of $b_i$
- power consumption and latency of $d_{i,j}$

**application:**
- computational cost and deadline of $a_i$
- volume, max latency and required bandwidth of $c_{i,j}$

- objectives: reduce bandwidth requirements on the platform, reduce power consumption, balance thermal dissipation, etc.
Application-Platform Mapping

- Platform models over-simplify the complex design space of on-chip multiprocessor platforms

- All approaches disregard the temporal dimension
  - application and platform models must include further details on time and concurrency
Multiprocessor System-on-Chip Platforms

- Many architectural alternatives
  - homogeneous vs. heterogeneous processing
  - shared memory vs. distributed memory
  - on-chip interconnect
    - point-to-point, crossbar
    - on-chip bus
    - network-on-chip

source: Intel
Multiprocessor System-on-Chip Platforms

- **Cell Processor**
  - It has nine processors: one Power Processor Element (PPE) and eight Synergistic Processing Elements (SPE)
  - PPE has separated I & D L1 cache (32KB each)
  - Each SPE can only access its 256KB of local storage (LS) and uses its memory flow controller (MFE) to perform DMA operations to/from LS (non-blocking)
  - Bandwidth (3.2 GHz)
    - SPE <-> LS = 2x 25.6 GB/s
    - MFC <-> EIB = 2x 25.6 GB/s
    - MIC <-> EIB = 2x 25.6 GB/s
    - L1 <-> L2 = 2x 51.2 GB/s
Multiprocessor System-on-Chip Platforms

- **ARM Cortex-9 MPCore**
  - Can have 1-4 Cortex-A9 cores with separated I & D L1 cache (32KB each)
  - Data caches of all cores are fully coherent
  - Interface to external components be made cache-coherent
  - On-chip interconnect based on AMBA standard
Multiprocessor System-on-Chip Platforms

- As the number of cores increase, on-chip communication becomes a major issue

Source: ITRS Roadmap
Multiprocessor System-on-Chip Platforms

- **Current solution: reusability**
  - standard processors
  - reusable IP cores
  - reusable on-chip interconnects

- **Networks-on-Chip (NoC)**
  - multi-hop, packet-based interconnect
  - scalable, high bandwidth
  - low power (shorter wires)
  - regular, reusable
Multiprocessor System-on-Chip Platforms

- Networks-on-Chip: design decisions
  - topology
  - packetisation
  - flow control
  - routing
  - switching
  - arbitration
  - buffering
Multiprocessor System-on-Chip Platforms

- Networks-on-Chip for 3D architectures
  - all previous alternatives and more
  - may integrate dies produced with different technologies
  - asymmetry between horizontal and vertical channels
  - increased locality
  - thermal issues become critical
    - load balancing
Multiprocessor System-on-Chip Platforms

- Design decisions must be evaluated using abstract models of the platform
  - cycle-accurate
  - cycle-approximate
  - untimed/functional
  - structural/graph

- Modeling trade-offs
  - accuracy
  - observability
  - speed
Multiprocessor System-on-Chip Platforms

- Cycle-accurate and cycle-approximate models

  - Flit-level accuracy
  - Observability: buffer occupation, latency, throughput, ...
  - Graphical interface to experiment with topology, routing, buffering, ...
Multiprocessor System-on-Chip Platforms

- Coding techniques to reduce bit transition activity over NoC interconnects

experimentation of alternative coding techniques aiming to reduce the power consumption in NoC interconnects

reductions of up to 60% of bit transition were achieved
Multiprocessor System-on-Chip Platforms

- Simplified models
  - packet-level accuracy, 1-position buffers
    - complete packet is abstracted by header and trailler
    - header's way through the network is fully simulated, trailler latencies are calculated
    - less than 5% error for average latency (compared with cycle-accurate HDL model)
  - fully analytical
    - latencies of packet delivery are calculated upon packet creation and refined until packet delivery
    - function of network occupation, route, buffer sizes, switching and arbitration techniques
  - real-time analysis
    - static analysis of worst-case interference between network flows
Application Modeling

- Compilers can only go so far on extracting parallelism from sequential code.
- Developers must have the means to specify the inherent concurrency of each particular application:
  - avoiding pitfalls such as deadlocks or undesired non-determinism.
Application Modeling

- Many concurrent programming models available
  - threads
  - concurrent processes with message-passing
  - actors
  - streams/dataflow
  - CSP
  - timed automata
  - <add your favorite here>
Application Modeling

- The choice of which concurrent programming model should be adopted depends on
  - application domain
  - familiarity by the designer/developer
  - availability of stable flows and tools

<table>
<thead>
<tr>
<th>model</th>
<th>tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>threads</td>
<td>libraries in Java and C++</td>
</tr>
<tr>
<td>message passing</td>
<td>OpenMP</td>
</tr>
<tr>
<td>actors</td>
<td>Simulink, Ptolemy</td>
</tr>
<tr>
<td>streams / dataflow</td>
<td>StreamIt, CUDA</td>
</tr>
<tr>
<td>timed automata</td>
<td>UPAAL</td>
</tr>
</tbody>
</table>
Application Modeling

- Application models are abstract representations of a program, which can be used both at design time and at runtime
  - can be strongly influenced by the programming model
Application Modeling

- Executable models
  - fast execution
  - rich set of modeling constructs, different abstraction levels
  - clearly defined execution semantics, so that the dynamic behavior of the application can be properly validated

- Formal properties
  - models of concurrent computation
  - concurrency constraint through the definition of ordering relations
  - polymorphic type systems
Concurrent Models of Computation

- Actor orientation, as proposed by E. A. Lee, UC Berkeley
  - revisited fundamental concepts from Atkinson & Hewitt (MIT, 1977) and Gul Agha (MIT, 1986)

- Actor orientation basics
  - execution semantics of a given system model was factored out from the individual components of that model
  - execution semantics is associated to a well defined Model of Computation (MoC)
  - heterogeneous models can be created through the hierarchical composition of MoCs
Actor-orientation

execution semantic of the top level model

execution semantic of the composite model
Application Modeling

- Extending actor-orientation with types and explicit ordering
  - UML suitable visual representation for the definition of polymorphic type systems (class diagrams) and ordering relations (sequence diagram)
  - but
  - UML is not an executable specification language
UML sequence diagrams within actors

- Recall the formal definition of MSC tuple \( \langle P, E, C, l, m, < \rangle \) where
  - \( P \) is a finite set of processes
  - \( E \) is a finite set of events
  - \( C \) is a finite set of names for messages
  - \( l: E \rightarrow T = \{ p!q(a), p?q(a), p(a) \mid p \neq q \in P, a \in C \} \)
  - \( m: S \rightarrow R \)
  - \( < \subseteq E \times E \) is a acyclic relation between events consisting of:
    - a total order on \( E_p \) for every \( p \in P \), and
    - \( s < r \), whenever \( m(s)=r \)
UML sequence diagrams within actors

- Recall the formal definition of MSC
  - order of events (message occurrence) within a process (lifeline) is a total order
  - the reflexive-transitive closure of < (denoted as <*) is a partial order on the complete set E
  - enough for the definition of an untimed model of computation

- different possibilities were explored and integrated as a library of directors on an extended version of Ptolemy II
Case study

- Application modeling based on behavioral patterns and polymorphic type systems

  - Application functionality described using actors
  - Constraints to concurrent execution described using UML
System Validation

- Joint validation of application and platform models
- Application is back-annotated with communication costs and power consumption data from the execution of platform models
System Validation

- Mapping heuristic becomes a design decision
  - experiments show that in large multiprocessor platforms, the average communication latency of a given application can vary by 2 orders of magnitude according to the employed mapping heuristic
Case study

- Application and platform modeling on Ptolemy II
  - multiple MoCs
  - supports the evaluation of how different platforms execute a given application
  - supports the evaluation of different mapping heuristics

synthetic example: autonomous vehicle
Conclusions and Future Work

- Extensions to the state-of-the-art on system-level specification and validation
  - combination of formalism, simulation/execution and analytical solution

- Extensions on application and platform models allow for richer mapping mechanisms
  - temporal constraint based on concurrent MoC and explicit ordering
  - spacial constraint based on functional types
Conclusions and Future Work

- Further progress must be done in mapping to fully explore information in application and platform models
  - time and concurrency
  - functional constraints

- Must explore different implementation styles, aiming to achieve more reliable platforms
  - static vs. dynamic mapping
  - centralised vs. distributed mapping