Multimedia Multiprocessor Systems: Analysis, Design and Management

Akash Kumar
Modern Multimedia Embedded Systems
Trends in Multimedia Systems

- Increasing number of features i.e. applications
- Simultaneously active applications
- Power increasingly becoming more important
- Short time-to-market, new devices released every few months
- Multiple standards to be supported
- Multiprocessors being used increasingly
Challenges in Multimedia System Design

- Ensuring all applications can meet their performance
- Handle the huge number of use-cases i.e. combinations of applications
  - Each possible set of applications leads to a new use-case
  - For 10 applications there are over a thousand use-cases!
- Limit the design time
  - Late launch of products directly hurts profits
  - Increased design-time implies higher design costs
- Deal with dynamism in the applications
Contributions

- **Analysis**
  - Accurately predict performance of multiple applications executing concurrently
  - Basic and iterative probabilistic techniques

- **Design**
  - Synthesizing MPSoC for multiple applications
  - Synthesizing MPSoC for multiple use-cases

- **Management**
  - Resource manager for MPSoC systems
  - Admission control and budget enforcement
Assumptions

- Heterogeneous MPSoC used increasingly more
  - Different levels of parallelism in application
  - uProc – better for control-flow
  - DSP – better for signal processing
  - Dedicated hardware blocks needed for certain parts
  - Improves efficiency and saves power

- Applications modeled as SDF
- First-come-first-serve arbiter at cores
- Non-preemptive system – tasks can not be stopped

Copyright © 2010 Akash Kumar
Non-Preemptive Systems

- State-space needed is smaller
- Lower implementation cost
- Less overhead at run-time
- Cache pollution, memory size
Outline

- Introduction – Multimedia Multiproc Systems
- Introduction to SDF
- Analysis
  - Basic Probabilistic Performance Prediction
  - Iterative Probabilistic Performance Prediction
- Design
  - Synthesizing MPSoC for multiple applications
  - Synthesizing MPSoC for multiple use-cases
- Management
  - Resource Management for MPSoC systems
Synchronous Dataflow Graphs

- First proposed in 1987 by Edward Lee

- SDF Graphs used extensively
  - SDFG: Synchronous Data Flow Graphs
  - DSP applications
  - Multimedia applications

- Similar to task graphs with dependencies
Synchronous Dataflow Graphs

1. Fire A

2. Execution time

3. Actor

4. Rate

5. Token

6. Channel
Synchronous Dataflow Graphs

![Diagram of a synchronous dataflow graph]

1. A → B → C
2. Fire B
3. A → B → C

Copyright © 2010 Akash Kumar
Synchronous Dataflow Graphs

- Example – H263 Decoder

![Diagram of H263 Decoder]

- VLD: 120,000
- Reconstruction: 30,000
- IQ: 96,000
- IDCT: 28,800

Copyright © 2010 Akash Kumar
Synchronous Dataflow Graphs

- **Advantages**
  - Easily allows performance analysis of single applications
  - Communication buffers can be easily modeled

- **Disadvantages**
  - Sharing of resources is hard to model
  - Only static resource arbitration can be modeled: infinite possibilities with multiple applications
  - Difficult to analyze performance of multiple applications executing concurrently
  - Unable to handle dynamism in the application
Problem: Predicting Multiple Application Performance

- Two applications – each with three actors
- Mapped on a heterogeneous platform
- Non-preemptive scheduler

Mapping & Scheduling
## Considering Only Actors on a Processor

### Task Only Actors

<table>
<thead>
<tr>
<th>Task</th>
<th>Only Actors</th>
<th>Individual Graph</th>
<th>Worst Case</th>
<th>Static</th>
<th>Priority Based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A pref.</td>
</tr>
<tr>
<td>A</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
<td>B pref.</td>
</tr>
<tr>
<td>B</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Iteration count for each task for 3,000 cycles*
Considering Only Applications

<table>
<thead>
<tr>
<th>Task</th>
<th>Only Actors</th>
<th>Individual Graph</th>
<th>Worst Case</th>
<th>Static</th>
<th>Priority Based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A pref.</td>
</tr>
<tr>
<td>A</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Iteration count for each task for 3,000 cycles
Worst Case Waiting Time

A

B

P1
P2
P3

Calculate waiting time

Copyright © 2010 Akash Kumar
Worst Case Waiting Time

![Diagram of worst case waiting time with nodes A and B and processes P1, P2, P3.]
Worst Case Waiting Time

<table>
<thead>
<tr>
<th>Task</th>
<th>Only Actors</th>
<th>Individual Graph</th>
<th>Worst Case</th>
<th>Static</th>
<th>Priority Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
<td>A pref.</td>
</tr>
<tr>
<td>B</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td></td>
<td>B pref.</td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Iteration count for each task for 3,000 cycles

Unrealistic!
Lower Bound

Copyright © 2010 Akash Kumar
Static Order Arbitration

Add ordering dependencies (edges)

Steady state

t₀  t₁  t₂  Steady state  t₃
## Problem: Predicting Performance

### Task Only Actors

<table>
<thead>
<tr>
<th>Task</th>
<th>Individual Graph</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>30</td>
<td>20</td>
</tr>
</tbody>
</table>

### Iteration count for each task for 3,000 cycles

<table>
<thead>
<tr>
<th>Task</th>
<th>Static Priority Based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A pref.</td>
</tr>
<tr>
<td>A</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
</tr>
</tbody>
</table>

### Total

| Total | 60 | 40 | 20 | 30 |
Problem: Predicting Performance – Priority Based

\[ \begin{align*}
\text{P1} & \quad & \text{P2} & \quad & \text{P3} \\
A & \quad & B & \quad & \\
\text{50} & \quad & \text{Steady State} & \quad & \text{50}
\end{align*} \]
### Problem: Predicting Performance

#### Task Only

<table>
<thead>
<tr>
<th>Task</th>
<th>Only Actors</th>
<th>Individual Graph</th>
<th>Worst Case</th>
<th>Static</th>
<th>Priority Based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A pref.</td>
</tr>
<tr>
<td>A</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

Iteration count for each task for 3,000 cycles

Copyright © 2010 Akash Kumar
Problem

No good techniques exist to analyze performance of multiple applications on non-preemptive heterogeneous systems.

Use probabilistic approach to estimate the performance of multiple applications running on an MPSoC platform.
Analyzing Multiple Applications Performance

- When resources need to be shared, the actor execution may be delayed
- Determining this waiting time is the key

\[ t_{\text{resp}} = t_{\text{exec}} + t_{\text{wait}} \]
Probability Distribution

- Compute the probability distribution of a resource being blocked by an actor

\[ E(x) = \int_0^{50} x \cdot \frac{1}{150} \, dx \]

\[ = \left[ \frac{1}{150} \cdot \frac{x^2}{2} \right]_0^{50} = 8 \]

\( x \) denotes the time other actors have to wait for respective resources to be free from actors of A

E\((x)\) provides the expected time an actor will need to wait when sharing resources with actors of A
Updated Response Time

A

50

50

50

A

B

50

50

50

B

58

58

58

58

Copyright © 2010 Akash Kumar
Basic P³ Algorithm

- Compute throughput of all applications
- Compute the probability of blocking a resource
- Estimate the waiting time for all actors
- Update the response time for all actors
  - Response time = execution time + waiting time
- Re-compute the application throughput
So if actor $a_i$ and $b_i$ are mapped on the same resource, $b_i$ on average will need to wait for

$$\mu_{a_i} \cdot \rho_{a_i}$$

$$P_{AB} = P_A + P_B - P_A \cdot P_B$$

$$\mu_{AB} \cdot P_{AB} = \frac{1}{2} \cdot P_A \cdot P_B \cdot (\mu_A + \mu_B) + \mu_A \cdot P_A + \mu_B \cdot P_B$$

$$= \mu_A \cdot P_A \cdot (1 + \frac{1}{2} P_B) + \mu_B \cdot P_B \cdot (1 + \frac{1}{2} P_A)$$

$$\mu_{a_1 \ldots a_n} \cdot P_{a_1 \ldots a_n} = \sum_{i=1}^{n} \mu_{a_i} \cdot P_{a_i} \left( 1 + \sum_{j=1}^{n-1} \frac{(-1)^{j+1}}{j + 1} \right)$$

$$\prod_j (P_{a_1} \ldots P_{a_{i-1}} P_{a_{i+1}} \ldots P_{a_n})$$
Complexity Reduction

- Overall complexity is $O(n^n)$ – $n$ is the number of actors mapped on a processing resource
- Higher order probability products
  - Limit the equation to only second or fourth-order
- Complexity reduces significantly

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>$O(n^n)$</td>
</tr>
<tr>
<td>Second-order</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Fourth-order</td>
<td>$O(n^4)$</td>
</tr>
</tbody>
</table>
Probabilistic Performance Prediction (P³)

- **Basic P³ technique**
  - Looks at all possible combinations of other actors blocking a particular actor
  - Results in exponential possibilities

- **Iterative P³ technique**
  - Looks at how an actor can contribute to waiting time of other actors
  - Results in linear complexity
  - Iterating over the algorithm while updating throughput improves the estimate further
Determining the Waiting Time

- Three states of an actor
  - **Not ready** – data not present
    - Actors arriving in this state, are not affected by this actor
  - **Ready and waiting** – data present, but resource is busy
    - Actors arriving in this state have to wait for the full execution of this actor
  - **Ready and executing** – data and resource available
    - Waiting time for other actors depend on where the actor is in its execution
    - Uniform distribution assumed
A’s Waiting Time Due to B

- B not in queue
- B being served
- B waiting in queue

Arbiter
Processor

Copyright © 2010 Akash Kumar
Updated Probability Distribution

\[ E(x) = P_w t_{\text{exec}} + P_e \frac{t_{\text{exec}}}{2} \]

When the actor is not ready

When the actor is in queue

When the actor is executing

Copyright © 2010 Akash Kumar
Updated Probability Distribution – Conservative

When the actor is not ready

\[ E(x) = P_w \cdot t_{exec} + P_e \cdot t_{exec} \]
\[ = (P_w + P_e) \cdot t_{exec} \]

When the actor is in queue

When the actor is executing
Iterative Probability

- Iterate until the analysis estimate stabilizes
  
  Updating the throughput in one iteration
  - Compute **throughput** of all applications
  - Compute the probability of blocking a resource – both while waiting and executing
  - Estimate the waiting time for all actors
  - Update the response time for all actors
    - Response time = execution time + waiting time
  - Re-compute the application **throughput**
Experimental Results

- SDF³ tool used to generate random graphs
  - Ten graphs generated
  - Each had 8-10 actors
  - Over 1000 use-cases generated

- Simulations performed using POOSL – Parallel Object Oriented Specification Language

- 28 hours for simulation

- 10 min for analysis using all approaches
Iterative Analysis – all applications together

Applications

Copyright © 2010 Akash Kumar
Iterative Analysis – all applications together

Applications

Copyright © 2010 Akash Kumar
### FPGA Implementation Results

<table>
<thead>
<tr>
<th>Algorithm/Stage</th>
<th>Clock cycles</th>
<th>Error (%age)</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>Load from CF Card</td>
<td>1903500</td>
<td></td>
<td>O(N.n.k)</td>
</tr>
<tr>
<td>Throughput Computation</td>
<td>12688</td>
<td></td>
<td>O(N.n.k)</td>
</tr>
<tr>
<td>Worst Case</td>
<td>2090</td>
<td>72.6</td>
<td>83.1</td>
</tr>
<tr>
<td>Second Order</td>
<td>45697</td>
<td>22.3</td>
<td>44.5</td>
</tr>
<tr>
<td>Fourth Order</td>
<td>1740232</td>
<td>9.9</td>
<td>28.9</td>
</tr>
<tr>
<td>Iterative - 1 Iteration</td>
<td>15258</td>
<td>12.6</td>
<td>36</td>
</tr>
<tr>
<td>Iterative - 1 Iteration*</td>
<td>27946</td>
<td>12.6</td>
<td>36</td>
</tr>
<tr>
<td>Iterative - 5 Iterations*</td>
<td>139730</td>
<td>2.2</td>
<td>3.4</td>
</tr>
<tr>
<td>Iterative - 10 Iterations*</td>
<td>279460</td>
<td>1.9</td>
<td>3.0</td>
</tr>
</tbody>
</table>

N-number of applications
n-number of actors in an application
k-number of throughput equations for an application
m-number of actors mapped on a processor
M-number of processors

- **19 ms with 100 MHz**
- **2.8 ms with 100 MHz**
Outline

- Introduction – Multimedia Multiproc Systems
- Introduction to SDF
- Analysis
  - Basic Probabilistic Performance Prediction
  - Iterative Probabilistic Performance Prediction
- Design
  - Synthesizing MPSoC for multiple applications
  - Synthesizing MPSoC for multiple use-cases
- Management
  - Resource Management for MPSoC systems
Problem

- Current Design Practice for multiple applications
  - Manual or Semi-automated

- Which is
  - Error Prone
  - Time Consuming
Current Tools - Example

- **Xilinx**
  - Automatic tool chain limited to single processors
  - No Support for multiple applications
  - Design space exploration is manual
Solution

- Multi Application Multi-Processor Synthesis
  - A design-flow that takes in application(s) specifications
  - Generates the entire MPSoC hardware
  - Creates the software models for it
  - Real C-program can also be run

- Provides two main benefits
  - Fast design space exploration
  - Support for multiple applications
MAMPS Overview

Application Specification

SDF

Design Description

Software Project for Processors
Hardware Topology
Xilinx Project

Generated Design

Proc 0
a0, a1

Proc 1
b0, b1

Proc 3
d0

Proc 2
c0, c1

AO FSL
A1 FSL
- **Software Arbitration**
  - Static Scheduling
  - Dynamic Scheduling
Example – H263 Decoder

Diagram:
- VLD: 120,000
- IQ: 96,000
- 30,000
- 2376
- 1
- 2
- IDC T
- 28,800
- 1188
- 1188
- 1188
- 1188
- Reconstruction

Copyright © 2010 Akash Kumar
• Example – H263 Decoder

[Diagram showing the connection between Pro 0 VLD, Pro 1 IQ, Pro 2 IDCT, and Pro 3 Recon, with links to Timer, UART, CF Card, DDR RAM, and FIFO LINKS.]
Standalone Automated DSE Data Collection
DSE Case Study – Buffer-throughput trade-off

- JPEG and H263 decoders
## Design Time

<table>
<thead>
<tr>
<th></th>
<th>Manual Design</th>
<th>Generating Single Design</th>
<th>Complete DSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Generation</td>
<td>~2 days</td>
<td>40ms</td>
<td>40ms</td>
</tr>
<tr>
<td>Software Generation</td>
<td>~3 days</td>
<td>60ms</td>
<td>60ms</td>
</tr>
<tr>
<td>Hardware Synthesis</td>
<td>35:40 min</td>
<td>35:40 min</td>
<td>35:40 min</td>
</tr>
<tr>
<td>Software Synthesis</td>
<td>0:25 min</td>
<td>0:25 min</td>
<td>10:00 min</td>
</tr>
<tr>
<td>Total time</td>
<td>~5 days</td>
<td>36:05 min</td>
<td>45:40 min</td>
</tr>
<tr>
<td>Iterations</td>
<td>1</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>Average time/ iteration</td>
<td>~5 days</td>
<td>36:05 min</td>
<td>1:54 min</td>
</tr>
<tr>
<td>Speed-Up</td>
<td>-</td>
<td>1x</td>
<td>19x</td>
</tr>
</tbody>
</table>

*Speedup!*
Used by following people

- Ahsan Shabbir – TUe.
- Michiel Rooijakkers – TUe.
- Thom Gielen – TUe and NUS, Singapore.
- Priyantha Desilva – NUS, Singapore.
- Shakith Fernando – NUS, Singapore.
- Zhonglei – TU Munchen, Germany.
- James Young - Brigham Young University.
- Amit Kumar Singh – Nanyang Technical University, Singapore.
- Guan Yu – IMEC, Belgium.
Handling Multiple Use-cases

- For rapid prototyping, hardware synthesis time is the bottleneck
  - Limits the design space exploration

- For real system, more use-cases implies
  - More memory to store the configuration
  - Increased switching

- Use-case merging and partitioning
  - Reduces the number of partitions
  - Reduces the synthesis time
  - Better for DSE, and run-time memory
Use-case Merging

Use-case A

Proc 0 ➔ Proc 1 ➔ Proc 2

Use-case B

Proc 0 ➔ Proc 1 ➔ Proc 2 ➔ Proc 3

Merged Design

Proc 0 ➔ Proc 1 ➔ Proc 3 ➔ Proc 2

Copyright © 2010 Akash Kumar
Use-case Partitioning
## Use-case Merging and Partitioning Results

<table>
<thead>
<tr>
<th></th>
<th>Random Graphs</th>
<th>Mobile Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Partitions</td>
<td>Time (ms)</td>
</tr>
<tr>
<td><strong>Without Reduction</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Without Merging</td>
<td>853</td>
<td>-</td>
</tr>
<tr>
<td>Greedy</td>
<td>Out of Memory</td>
<td>Out of Memory</td>
</tr>
<tr>
<td>First-Fit</td>
<td>126</td>
<td>400</td>
</tr>
<tr>
<td><strong>With Reduction</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Without Merging</td>
<td>178</td>
<td>100</td>
</tr>
<tr>
<td>Greedy</td>
<td>112</td>
<td>3,300</td>
</tr>
<tr>
<td>First-Fit</td>
<td>116</td>
<td>300</td>
</tr>
<tr>
<td>Optimal Partitions</td>
<td>≥110</td>
<td>-</td>
</tr>
<tr>
<td>Reduction Factor</td>
<td>7</td>
<td>-</td>
</tr>
</tbody>
</table>
Outline

- Introduction – Multimedia Multiproc Systems
- Introduction to SDF

Analysis
- Basic Probabilistic Performance Prediction
- Iterative Probabilistic Performance Prediction

Design
- Synthesizing MPSoC for multiple applications
- Synthesizing MPSoC for multiple use-cases

Management
- Resource Management for MPSoC systems
Dynamism in Applications

- Multimedia applications are often dynamic
- SDF assumes worst-case-execution-time – not realistic
- Analysis results may be pessimistic – lead to waste of resources & energy
- Dynamic execution time may lead to unpredictable application performance
Unpredictability – Variation in Execution Time

P1

P2

P3

A

B

Steady State

Copyright © 2010 Akash Kumar
Resource Manager

- **Budget enforcement**
  - When running, each application signals RM when it completes an iteration
  - RM keeps track of each application’s progress
  - Operation modes
    - ‘Polling’ mode
    - ‘Interrupt’ mode
  - Suspends application if needed
Budget Enforcement (Polling)

Performance goes down!

Resource Manager

Better than required!

New job enters!

Job suspended!

Job resumed!
Performance without Resource Manager
Performance with RM – I (2.5m cycles)

Throughput (iterations per unit time)

Time (cycles)

Copyright © 2010 Akash Kumar
Performance with RM – II (500k cycles)
Conclusions

- Modern multimedia systems support a number of applications executing concurrently.
- A number of challenges remain for designers
- Probabilistic performance prediction presented for multiple applications executing concurrently
- The approach is fast, yet accurate: ideal for DSE
- A design methodology is proposed that take application(s) specification and generates the MPSoC platform
- Handle multiple use-cases by merging and partitioning
- Resource manager presented: admission control and budget enforcement
Future Work

- Support for hard real-time applications: both analysis and design-flow
- Provide soft real-time guarantee: analysis
- Mixing hard and soft real-time tasks
- Extend MAMPS to CSDF, SADF models
- Achieving predictability in suspension
- Considering the use-case usage when partitioning them
Relevant Publications – Journals (first author)

- Akash Kumar et al. Multi-processor Systems Synthesis for Multiple Use-Cases of Multiple Applications on FPGA. Transactions on Design Automation in Electronic Systems (ToDAES), 2008. ACM.


Relevant Publications – Conferences (first author)


