Modelling a Multi-Core Media Processor Using JCSP

Anna Kosek, Napier University, UK
Jon Kerridge, Napier University, UK
Aly Syed, NXP, The Netherlands
Motivation

• Use CSP model capabilities to create a parallel system
• Use JCSP to simulate a multi-core audio/video processor
• Use of mobile processes to implement allocation of tasks to cores
• Use allocation to control heat generation
  – For hand held devices
Original architecture

Multimedia stream

Channel decode -> Demux -> Video decode

Demux -> Audio decode

Sync -> Display
### Data stream

<table>
<thead>
<tr>
<th>Block k</th>
<th>Block 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_m ... a_2 a_1 a_0</td>
<td>a_0 ... a_2 a_1 a_0</td>
</tr>
<tr>
<td>v_m ... v_2 v_1 v_0</td>
<td>v_n ... v_2 v_1 v_0</td>
</tr>
</tbody>
</table>

- Demux
- Audio subtask
- Video subtask

Data Stream

**NAPIER UNIVERSITY EDINBURGH**
# Data structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Special values</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Block number</td>
<td>Starts from 0</td>
</tr>
<tr>
<td>t</td>
<td>Task number</td>
<td>Starts from 0</td>
</tr>
</tbody>
</table>
| c    | Subtask category | Audio = 0  
                  | Video = 1 |
| r    | Requested subtask type | Starts from 0 |
| w    | Amount of work of requested subtask type (units of work) | Starts from 1 |
| o    | Variable reserved for actual core performance (outcome) time needed to execute requested subtask | By default equals 0, but changes after subtask processing |
JCSP Components

- Processes
- Channels
- Alternative
- Network Channels (jcsp.net)
- Mobile processes
Mobile process

- Moves around the system
- Can be connected to any process and be run in parallel with it
- Mobile process carries data set representing a subtask
Channel Decode

- Reads data stream
- Sends the task, divided into subtasks, to the Demux process
Architecture

Channel decode

Demux

Core 0 Core 1 Core 2
Core 3 Core 4 Core 5
Core 6 Core 7 Core 8
...

Sync

Display
Demux

- Requests core location from the Control process
- Creates connection and sends subtask to that Core
- Sends information about subtask ordering to the Sync process to enable synchronisation
Control

- Decides where to allocate a subtask
- Cores register their state and capabilities with the Control process
Architecture

Channel decode

Demux

Core 0
Core 1
Core 2

Core 3
Core 4
Core 5

Core 6
Core 7
Core 8

Sync

Display

Control

Core 0 → Core 1 → Core 2 → Core 3 → Core 4 → Core 5 → Core 6 → Core 7 → Core 8 → Sync → Display

NAPIER UNIVERSITY
EDINBURGH
Core

- Has its own capabilities
- Sends information to the Control process about: capabilities, availability and location
- Receives a subtask and executes it in a time depending on the core’s capabilities
- All cores run in parallel
- A core gains heat if the subtask’s execution takes more than 0.1 sec
- A hot core takes longer to process a subtask
- Heated cores cool down
Sync

- Synchronises audio and video subtasks
- Sends synchronised subtasks to the Display process
- Outputs some basic information concerning where subtasks were processed
Display

- Displays output information such as:
  - Block number
  - Audio / Video subtask
  - Subtask number
  - Optimum time
  - Actual time
## Typical output

<table>
<thead>
<tr>
<th>Block</th>
<th>A/V</th>
<th>subtask</th>
<th>optimum time</th>
<th>actual time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>0</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>V</td>
<td>0</td>
<td>100</td>
<td>700</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>V</td>
<td>1</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>2</td>
<td>70</td>
<td>140</td>
</tr>
<tr>
<td>0</td>
<td>V</td>
<td>2</td>
<td>200</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>50</td>
<td>500</td>
</tr>
<tr>
<td>1</td>
<td>V</td>
<td>0</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>1</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>V</td>
<td>1</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>2</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>1</td>
<td>V</td>
<td>2</td>
<td>200</td>
<td>1400</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>3</td>
<td>70</td>
<td>700</td>
</tr>
<tr>
<td>1</td>
<td>V</td>
<td>3</td>
<td>200</td>
<td>400</td>
</tr>
</tbody>
</table>
Corresponding Sync output

Sync: task: 0 (task type: 0, category: 0), block: 0, core: 7, processing time: 100
Sync: task: 0 (task type: 10, category: 1), block: 0, core: 7, processing time: 700
Sync: task: 1 (task type: 6, category: 0), block: 0, core: 7, processing time: 100
Sync: task: 1 (task type: 19, category: 1), block: 0, core: 1, processing time: 200
Sync: task: 2 (task type: 4, category: 0), block: 0, core: 1, processing time: 140
Sync: task: 2 (task type: 11, category: 1), block: 0, core: 7, processing time: 1800
Sync: task: 0 (task type: 9, category: 0), block: 1, core: 4, processing time: 500
Sync: task: 0 (task type: 14, category: 1), block: 1, core: 4, processing time: 1000
Sync: task: 1 (task type: 7, category: 0), block: 1, core: 4, processing time: 1000
Sync: task: 1 (task type: 19, category: 1), block: 1, core: 1, processing time: 200
Sync: task: 2 (task type: 6, category: 0), block: 1, core: 7, processing time: 70
Sync: task: 2 (task type: 10, category: 1), block: 1, core: 7, processing time: 1400
Sync: task: 3 (task type: 1, category: 0), block: 1, core: 4, processing time: 700
Sync: task: 3 (task type: 12, category: 1), block: 1, core: 1, processing time: 400
Conclusions

• A basic implementation capturing:
  – Distribution of work
  – Selection of cores
  – Core heating and cooling
  – Synchronisation
  – Limited output and performance capture
Future work

• Improve the input format
• Improve the monitoring capability
• Automating functional testing
• Core failure
• More complex heating and cooling strategies
• Multiple input streams
• Switching between cores to complete one subtask
• Removing need for a single Control process