Adaptive Runtime Resource Management of Heterogeneous Resources

Roel Wuyts
Principal Scientist, imec
Professor, KUL (Distrinet)
## Career Overview

### Studies: Licentiaat Informatica (VUB, 1991-1995)

<table>
<thead>
<tr>
<th>Year</th>
<th>Position</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>Doctoral Researcher</td>
<td>VUB</td>
</tr>
<tr>
<td>2001</td>
<td>Postdoc</td>
<td>University of Bern, Switzerland</td>
</tr>
<tr>
<td>2004</td>
<td>Chargé de cours</td>
<td>ULB</td>
</tr>
<tr>
<td>07</td>
<td>Principal Scientist</td>
<td>imec</td>
</tr>
<tr>
<td>08</td>
<td>Professor (10%)</td>
<td>KUL</td>
</tr>
</tbody>
</table>
Juggling Hats

IMEC
- Embedded devices
- Runtime resource management

ULB
- Object versioning
- AOP

KUL
Language Design
• Research organization located in Leuven
  – world-leading independent research center in nanoelectronics and nanotechnology
  – More Moore research targets semiconductor scaling for the 22nm technology node and beyond.
  – More than Moore research invents technology for nomadic embedded systems, wireless autonomous transducer solutions, biomedical electronics, photovoltaics, organic electronics and GaN power electronics.

• Numbers
  – Budget: ± 200 M€
  – Staff: ± 1700
  – Cleanroom: ± 10,000 m2
The ARES Team

IMEC Ph.D. Students

Maja D’Hondt
Rogier Baert
Carolina Blanch
Paul Coene
Zhe Ma
Roel Wuyts

IMEC M.S. Students

Narasinga Rao Miniskar
Hengjie Song

Master Students

Tipnis Ameya
ARES General Goal

- Software that takes advantage of heterogeneous platforms is becoming the rule.
- Developing such software is hard because:
  - A decision needs to be made regarding what software components can use what resources,
  - that decision varies at runtime as the application’s context changes.
  - moreover the decision needs to result in good performance,
  - And the software needs to run with many possible resource configurations

- ARES solves this problem through adaptive runtime resource management, a solution that monitors applications at runtime and decides the assignment of resources to software components at runtime according to a decision algorithm.
Resource Management at Network, Device and SoC Level
Context: Networked Video Processing

- Video wall
- Embedded display controllers
- Desktop computer
- Network
- Server
- Video processing device

**Diagram:**
- Video wall connected to embedded display controllers via the network.
- Network connects to a server.
- Video processing device is part of the network system.
Video Processing Device

- Previously: custom hardware
- **Now:** Device with off-the-shelf CPU and GPU and optionally DSP-board
  - Many different kinds of CPUs and GPUs -> high variability
  - Hardware evolves rapidly -> high variability
Software Pipelines in Video Processing

- Software to process and analyze video streams
  - encoders, decoders, transcoders, object (e.g. logo) detection, video scalers, color space conversion, ...

- Characteristics
  - Data-dependent: changing workloads (component A in example)
  - User/context interaction: changing pipelines (B triggers pipeline P)
Developing on Heterogeneous Platforms

• Assignment Problem: what runs where when?

Software

Hardware

GPU
CPU multi-core
DSP farm
Related Work

• **Practice**: (manual) design-space exploration + assumptions

• **Task assignment for heterogeneous systems**
  - Finer-granularity imposing only simple assignment strategies

• **Task scheduling on heterogeneous multicore architectures**
  - Only list scheduling and without taking data transfer times into account

• **Static scheduling heuristics for heterogeneous processors**
  - Formal approaches without implementation, no runtime assignment
Static Assignment Problem 1: which is best?

different workloads have different best static assignments on heterogeneous processors

different static assignments on GPU and CPU per workload

6 different best static assignments (sa) for 8 different workloads

GPU bottleneck
Static Assignment Problem 2: scaling

- **experiment (previous graph):**
  - 1 to 8 streams
  - 2 resolutions
  - 8 different load distributions over GPU and CPU
  - # static assignments ~ 100 (points in the graph)

- **professional video processing**
  - 1 to 64 streams
  - 4 resolutions
  - 64 different load distributions over GPU and CPU
  - # static assignments ~ 108
static assignment problem 3: heterogeneity

Variations in configurations of processors have different best static assignments for same workloads.

Best static assignments:

1 GPU and 2x 4-core CPU:
- > 4 tasks: sa A
- ≤ 4 tasks: sa B

1 GPU and 4-core CPU:
- sa A
ARES’ Adaptive runtime resource management

GRES run-time load balancing for all workloads is almost always better than the best static assignment per workload.
ARES’ approach is portable across platforms

ARES run-time load balancing is portable to different configurations of heterogeneous processors:

- exact same software stack adapts to underlying heterogeneous processors and achieves best performance all the time

(horizontal lines)

run-time load balancing on
1 GPU and 2x 4-core CPU:
- 1 to 8 streams
- 720p resolution

run-time load balancing on
1 GPU and 4-core CPU:
ARES Runtime Resource Management: parts

- Monitor resource assignment and usage
- Represent monitored information
- Decide assignment at runtime
  - use monitored information
  - predict, learn, adapt, ...
  - Pluggable strategies with different trade-offs
Monitoring and Representation Examples

• **We monitor:**
  - execution time of a component on a processing element
  - data transfer times between two connected components executing on different type of processing element

• **We represent:**
  - Average time + standard deviation per component and per processing element

<table>
<thead>
<tr>
<th>Task</th>
<th>PE</th>
<th>time</th>
<th>Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>cpu1</td>
<td>50</td>
<td>9,3%</td>
</tr>
<tr>
<td>t1</td>
<td>cpu2</td>
<td>52</td>
<td>5,6%</td>
</tr>
<tr>
<td>t1</td>
<td>gpu1</td>
<td>4</td>
<td>4,1%</td>
</tr>
<tr>
<td>t1</td>
<td>-&gt;gpu</td>
<td>13</td>
<td>12,8%</td>
</tr>
<tr>
<td>t1</td>
<td>gpu-</td>
<td>32</td>
<td>8,4%</td>
</tr>
<tr>
<td>t2</td>
<td>cpu1</td>
<td>134</td>
<td>3,6%</td>
</tr>
</tbody>
</table>
Assignment Strategies

- Can use the following information:
  - Hardware metadata: static and runtime
  - Software metadata: static and runtime

- Have to respond to assignment requests
  - Fast response is required

- Different algorithms are possible
  - Static (up-front) decision: no runtime adaptation (SoA)
  - Generic: fastest available, first finished
  - Domain-specific: prefer-GPU-sequence
  - Machine learning
Example: First Finish Strategy

```
<table>
<thead>
<tr>
<th>Task</th>
<th>PE</th>
<th>time</th>
<th>Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>cpu1</td>
<td>50</td>
<td>9.3%</td>
</tr>
<tr>
<td>t1</td>
<td>cpu2</td>
<td>52</td>
<td>5.6%</td>
</tr>
<tr>
<td>t1</td>
<td>gpu1</td>
<td>4</td>
<td>4.1%</td>
</tr>
<tr>
<td>t1</td>
<td>-&gt;gpu</td>
<td>13</td>
<td>12.8%</td>
</tr>
<tr>
<td>t1</td>
<td>gpu-</td>
<td>32</td>
<td>8.4%</td>
</tr>
<tr>
<td>t2</td>
<td>cpu1</td>
<td>134</td>
<td>3.6%</td>
</tr>
<tr>
<td>t3</td>
<td>cpu1</td>
<td>14</td>
<td>6.2%</td>
</tr>
</tbody>
</table>

...
Implementation

- **ARES Runtime resource manager implementation:**
  - Dynamic library for Unix (Linux, OS-X) and Windows
  - C and C++ header for integrating with applications
  - Uses Boost shared memory to store values
  - Low-overhead (0.01%)

- **Used with:**
  - AVC Encoder (CUDA-accelerated motion estimation)
  - GStreamer applications
  - Imec in-house multimedia framework in .Net on Windows
Making the AVC Encoder runtime managed?

```c
... //ask RRM to decide between GPU or CPU
proc_type = rrm_get_processor(encID);
if ( RRM_PROC_TYPE(proc_type) == RRM_PROC_GPU )
    cuda_me = 1;
else
    cuda_me = 0;

start2 = RDTSC ();
if (cuda_me == 1) {
    start = RDTSC ();
    GPUinit();
    cuda_motion_estimation();
    GPUExit();
    g_total_MEtime = (RDTSC () - start);
}
else {
    ((ARMVCM4P10_MEspec *)encInfo.params.meSpec)->no_gpu_data();
}
...

//update RRM execution time
update_kernel_timing(encID, proc_type, g_total_MEtime);
...```
City + Space: runtime managed
But...

- ... what runtime management strategy works best for my application?
- ... will my existing application benefit from runtime management?
- ... will my new application benefit from runtime resource management?
- ... what if my clients use a dualcore CPU and 2 GPU’s?
Exploration Tool

• Compare different runtime resource management strategies

• How?
  – Model software at high-level (connected components).
  – Decorate nodes with timing information:
    • Average execution times per processing element supported;
    • Data transfer Times between different processing elements.
    • These timings come from the runtime manager, from other profiling tools, from experience, or even from guestimates.
  – Model kind and number of processing elements.
  – Select the strategies you want to compare.

• Result?
  – Exploration tool simulates the execution for each strategy and outputs information that can be plotted (dropped frames, late frames, platform utilization)
Exploration Tool Input

Application 1

Source
Execution time: CPU: 0.011ms

Demux
Execution time: CPU: 0.00163ms

Scaler
Execution time: CPU: 0.115ms
GPU: 0.0129ms
Data Transfer Time: CPU->GPU: 0.079ms
GPU->CPU: 0.17ms

Sink
Execution time: CPU: 0.51ms

Application 2

Encoder
Execution time: CPU: 0.43ms
GPU: 0.074ms
Data Transfer Time: CPU->GPU: 0.085ms
GPU->CPU: 0.46ms

Hardware Description

8 CPU
1 GPU
2 DMA
Plotted output result
Device-level adaptive resource management

- Static assignments exhibit problems
  - Different solutions for different workloads or other runtime variability
  - Do not scale (exploration space explosion)
  - Different solutions for different platforms

- Runtime resource managed solution adapts to different conditions
  - Runtime variability
  - Heterogeneous platforms
Context: Networked Video Processing

Diagram showing networked video processing with a video wall, embedded display controllers, a desktop computer, a server, and video processing devices connected through a network.
Network: Connected heterogeneous devices

What to process when on what device?
System-wide Resource Allocation

Movies

<table>
<thead>
<tr>
<th>CIF</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>HD</td>
<td>1</td>
</tr>
<tr>
<td>CIF</td>
<td>2</td>
</tr>
<tr>
<td>SD</td>
<td>2</td>
</tr>
</tbody>
</table>

Server I

4CPU’s + 1GPU

Server II

2CPU’s + 1GPU

RRM (decision)

Network Switch

10 Gbps

Screen 1

1 Gbps

Screen 2

1 Gbps

Latency Simulation

Scheduling Strategy

Movies

CIF 1
SD 1
HD 1
CIF 2
SD 2

CIF 2
SD 2
HD 2
CIF 2

Movies

1
1
1
2
2
2
2
2
2
2
2
2

Scheduling Strategy

Latency Simulation
Possibility: Everything decoded at server, raw data to client

<table>
<thead>
<tr>
<th>Movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIF 1</td>
</tr>
<tr>
<td>SD 1</td>
</tr>
<tr>
<td>HD 1</td>
</tr>
<tr>
<td>CIF 2</td>
</tr>
<tr>
<td>SD 2</td>
</tr>
<tr>
<td>HD 2</td>
</tr>
<tr>
<td>CIF 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Server I</th>
</tr>
</thead>
<tbody>
<tr>
<td>4CPU’s + 1GPU</td>
</tr>
<tr>
<td>CIF 1</td>
</tr>
<tr>
<td>SD 1</td>
</tr>
<tr>
<td>HD 1</td>
</tr>
<tr>
<td>CIF 2</td>
</tr>
<tr>
<td>SD 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Server II</th>
</tr>
</thead>
<tbody>
<tr>
<td>2CPU’s + 1GPU</td>
</tr>
<tr>
<td>CIF 2</td>
</tr>
<tr>
<td>SD 2</td>
</tr>
<tr>
<td>HD 2</td>
</tr>
<tr>
<td>CIF 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Gbps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RRM (decision)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Screen 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Gbps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Screen 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Gbps</td>
</tr>
</tbody>
</table>

(-) Increases Bandwidth and network latency
(+ ) No processing cost at client
Possibility: Everything trans-coded at server, lower resolution sent to client

(+) Reduces Bandwidth and network latency
(+) Lower processing cost at client
(-) Increases processing cost at server
Possibility 4: Everything fully decoded at client

Movies

<table>
<thead>
<tr>
<th>CIF</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>HD</td>
<td>1</td>
</tr>
<tr>
<td>CIF</td>
<td>2</td>
</tr>
<tr>
<td>SD</td>
<td>2</td>
</tr>
</tbody>
</table>

Server I

4CPU’s + 1GPU

Server II

2CPU’s + 1GPU

Network Switch

RRM (decision)

10 Gbps

1 Gbps

1 Gbps

(-) All processing at client, might miss deadlines
(+) No BW or latency increase
network-level: first results

- distributed processing of video processing applications on server or client
- trade-off between processing at server, processing at client or transcoding to lower quality
- adaptive run-time resource management - using a mixture of the above - gives good results:

<table>
<thead>
<tr>
<th>Resolution/Quality</th>
<th>Processed at Server</th>
<th>Processed at Client</th>
<th>Transcode</th>
<th>Mixed Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>Medium-High</td>
</tr>
<tr>
<td>Missed Streams</td>
<td>6 (limited bw)</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BW (Gbps)</td>
<td>2.7</td>
<td>0.6</td>
<td>0.08</td>
<td>0.95</td>
</tr>
<tr>
<td>Latency (ms)</td>
<td>92</td>
<td>10</td>
<td>2.9</td>
<td>33</td>
</tr>
</tbody>
</table>
Networked Video Processing: Future Work

- Discover Distributed Processing Strategies
  - Trading of bandwidth, processing power and quality
- Implement
  - Currently extending the device-level manager
Conclusion

- Problem: how to develop software that runs on heterogeneous devices
  - At SoC level
  - At Device level
  - At Network Level

- Solution: runtime decision strategies decide what software component uses what resource

- Meta Remark: versatility of your studies make you valuable assets

- Meta Meta Remark: Choose according to Flexibility versus Pay