Performance Evaluation of Throughput Constrained Dataflow Programs Executed On Shared-Memory Multi-Core Architectures

Manuel SELVA

Supervised by:
Lionel MOREL, CITI
Kevin MARQUET, CITI
Stéphane FRÉNOT, CITI
Frédéric SOINNE, Bull
Stéphane ZENG, Bull

July 2, 2015
Moore’s Law

curve shows transistor count doubling every two years
What To Do With Transistors?

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)

Transistors

Frequency

Power

Instruction Level Parallelism

What To Do With Transistors?

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)

Multi-core

Transistors
Frequency
Power
Instruction Level Parallelism
Multi-core Architectures

Intel Nehalem - 4 cores - 2009

Samsung Exynos - 2 x 4 cores - 2012

Kalray MPPA - 256 cores - 2013
Multi-core Architectures

Intel Nehalem - 4 cores - 2009

Samsung Exynos - 2 x 4 cores - 2012

Kalray MPPA - 256 cores - 2013

One taxonomy

- Computing homogeneity
- Memory organization
Centralized Shared Memory
Centralized Shared Memory

Bandwidth bottleneck
Distributed Shared Memory - Aka NUMA
Distributed Private Memory

Core 1 -> Core 2 -> Core 3 -> Core 4

Core 5 -> Core 6 -> Core 7 -> Core 8

Core 9 -> Core 10 -> Core 11 -> Core 12

Core 13 -> Core 14 -> Core 15 -> Core 16

Core 17 -> Core 18 -> Core 19 -> Core 20

Core 21 -> Core 22 -> Core 23 -> Core 24

Core 25 -> Core 26 -> Core 27 -> Core 28

Core 29 -> Core 30 -> Core 31 -> Core 32

Mem

Network
Software Challenge: Several Applications
Software Challenge: Several Applications
Software Challenge: Single Application

Problem

- Identify several activities
- Handle communication and synchronization
Software Challenge: Single Application

Problem

- Identify several activities
- Handle communication and synchronization

Solutions

- (Semi) Automatically split existing apps
- (Re) Write apps using concurrent programming models
  - Threads, Data parallelism, **Dataflow**
Dataflow Applications Examples

Medical image processing [Albers2012]

Software Defined Radio [Dardaillon2014]

Video Decoding [Lucarz09]
Outline

Dataflow Programming & Problematic

Detection of SDF Bottleneck Actors

Profiling of Dataflow Programs

Conclusion & Perspectives
Dataflow Programming Model: Syntax

Dataflow Application Graph

- Actors with sequential atomic function
- Communication over FIFO channels only
Dataflow Programming Model: Semantic

Actors activation driven by tokens availability
Dataflow Programming Model: Semantic

Actors activation driven by tokens availability
Dataflow Programming Model: Semantic

Actors activation driven by tokens availability
Dataflow Programming Model: Semantic

Actors activation driven by tokens availability
Dataflow Programming Model: Semantic

Actors activation driven by tokens availability
A

B

C

D

Dataflow Programming Model: Semantic

Actors activation driven by tokens availability

Why is dataflow interesting?

- Actors can be executed in parallel
- Communication abstraction
Dataflow Execution Model: 4 Cores
Dataflow Execution Model: 4 Cores

![Dataflow Diagram]

- Core 1
- Core 2
- Core 3
- Core 4
- RAM
- Compiler

- A
- B
- C
- D

A; B; C; D;
Dataflow Execution Model: 4 Cores
Dataflow Execution Model: 2 Cores

A → B → D
A → C → D

Compiler
Mapper

Core 1: A; C; D;
Core 2: B;

RAM
Dataflow Execution Model: Single Core
Motivation

The graph shows the speedup vs single-core performance for different inputs in HEVC decoding. The x-axis represents the number of cores, ranging from 1 to 12, and the y-axis represents the speedup compared to a single-core performance. The graph includes lines for different inputs: 200 frames with 33 Actors.

Key Points:
- Different inputs (HEVC decoding).
- 200 frames.
- 33 Actors.
- Number of cores ranging from 1 to 12.
- Speedup vs single-core performance.

Graph Details:
- The graph includes multiple lines, each representing different configurations.
- The speedup reaches its peak at a certain number of cores and then stabilizes or decreases slightly for higher numbers of cores.

Overall, the graph highlights the performance benefits and potential bottlenecks when using multiple cores for HEVC decoding with different inputs.
Problem Statement

How to understand and identify performance bottlenecks in dataflow programs?

- **Contribution 1**: Automatic instrumentation to detect bottleneck actors in SDF graphs

- **Contribution 2**: CPU/memory profiling to analyse (and fix) bottlenecks on dataflow programs
Outline

Dataflow Programming & Problematic

Detection of SDF Bottleneck Actors

Profiling of Dataflow Programs

Conclusion & Perspectives
Synchronous Dataflow [Lee87] - SDF

Tokens consumption/production rates static

- Memory boundedness
- Static scheduling of actors
Applications Have Throughput Requirements

X Images/s

X Frames/s
Activation Frequency Analysis

Use throughput constraint *and* static rates
Activation Frequency Analysis

Use throughput constraint *and* static rates

- Required Activation Frequency (RAF)
Activation Frequency Analysis

Use throughput constraint *and* static rates

- Required Activation Frequency (RAF)
Activation Frequency Analysis

Use throughput constraint \textit{and} static rates

- Required Activation Frequency (RAF)

\begin{itemize}
  \item \textbf{Required Activation Frequency (RAF)}
\end{itemize}
Activation Frequency Analysis

Use throughput constraint \textit{and} static rates

- Required Activation Frequency (RAF)
- Time information in addition to data exchanges
Instrumentation of SDF Actors

A \rightarrow B: 2 \rightarrow 1
B \rightarrow C: 3 \rightarrow 1
B \rightarrow D: 3 \rightarrow 2
C \rightarrow A: 6 \rightarrow 1
D \rightarrow C: 2 \rightarrow 3

1000 \text{tokens/s}
Instrumentation of SDF Actors

Extended Compiler

250Hz  500Hz  250Hz  500Hz
A;  B;  C;  D;

1000 \text{tokens/s}
Instrumentation of SDF Actors

- 1000 tokens/s
- Extended Compiler
- 250Hz 500Hz
- Mapper
- Core 1: A; C; D;
- Core 2: B;
- RAM
**Instrumentation of SDF Actors**

![Diagram of SDF Actors Instrumentation]

- **A**; **B**; **C**; **D**; 250Hz 500Hz 250Hz 500Hz
- **Core 1**: A; B; C; 150Hz
- **Core 2**: B; 1000 tokens/s
- **Extended Compiler**
- **Mapper**
- **RAM**
Conclusion

- Identification of SDF bottleneck actors
- Validation on Streamit benchmarks [Thies10]

Limitations

- Real-life applications don’t fit all into SDF
- Identify where the problem is but not its origin
Outline

Dataflow Programming & Problematic

Detection of SDF Bottleneck Actors

Profiling of Dataflow Programs

Conclusion & Perspectives
Motivation

Different inputs
HEVC decoding
200 frames
33 Actors

Number of cores
Speedup vs single-core
Dynamic Dataflow [Lee95] - DDF

Token consumption/production rates dynamic

- No static analyses
- Runtime mechanisms required for scheduling
- Runtime mechanisms to identify bottlenecks
How To Understand and Identify Performance Bottlenecks in Dataflow Programs?

Correlate hw profiling to the DF graph
How To Understand and Identify Performance Bottlenecks in Dataflow Programs?

Correlate hw profiling to the DF graph
How To Understand and Identify Performance Bottlenecks in Dataflow Programs?

Core domain

Uncore domain

Correlate hw profiling to the DF graph
Dataflow Profiler

CPU Profiling

- Measure activation time
- At core level
- At actors level

Memory Profiling

- Based on hardware mechanisms
- Latency of accesses to FIFO
- Latency of accesses to the actors’ internal state
- Identification of hardware bottlenecks
Cores Balance

**Work distribution by core (%)**

**HEVC**
Input: Kimono
200 frames

**Number of cores**

1. Core 1: 100
2. Core 2: 54
3. Core 3: 43
4. Core 4: 36
5. Core 5: 34
6. Core 6: 31
7. Core 7: 29
8. Core 8: 29
9. Core 9: 27
10. Core 10: 27
11. Core 11: 28
12. Core 12: 28
Cores Balance

HEVC
Input: Kimono
200 frames

Single actor:
Inter pred.
Cores Balance

HEVC
Input: Kimono
200 frames

Single actor:
Inter pred.
Split it [Jerbi14]
Total Work Time is Increasing

Input: Kimono
200 frames

Number of cores

Total Work Time (cycles)

+49%

HEVC
Communication Overhead On NUMA

![Diagram showing communication overhead on NUMA architecture]

- Xeon X5650 processors
- Core domain: Core 1, L1, L2, ..., Core 6, L1, L2
- Uncore domain: L3, Mem. Ctrl, QPI, Memory Bank 1
- Xeon X5650 processors
- Core domain: Core 7, L1, L2, ..., Core 12, L1, L2
- Uncore domain: L3, Mem. Ctrl, QPI, Memory Bank 2
Communication Overhead On NUMA

Remote vs local latency
+30%
[Molka2009, David2013]
Communication Overhead On NUMA

Cache coherency protocol
QPI overhead
[Molka2009]
Communication Overhead On NUMA

Memory controllers and QPI links contention  
[Dashti2013]
NUMA - Performance Monitoring Unit

Hardware profiling mechanisms

- Hard to program
A library for NUMA Profiling

- Architecture abstraction
  - Memory bandwidth profiling
  - Memory access sampling
Using `numap` for Dataflow Memory Profiling
Using `numap` for Dataflow Memory Profiling

**Core 1**

1. L1
2. L2

**Core 6**

1. L1
2. L2

**Core 7**

1. L1
2. L2

**Core 12**

1. L1
2. L2

**Mem. Ctrl**

**QPI**

**Memory Bank 1**

**Memory Bank 2**

DF applications saturate memory bandwidth?
Using `numap` for Dataflow Memory Profiling

Associate remote accesses with actors and FIFOs

@=0x7123CFF
Memory Bandwidth Usage

Input: Kimono
200 frames

Number of cores

Average Bandwidth (GB/s)

Read max bandwidth
Write max bandwidth

Read
Write

HEVC

Table:

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Average Bandwidth (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>36 / 42</td>
</tr>
</tbody>
</table>
Communication Cost

Average Memory Latency (cycles)

Input: Kimono
200 frames

Number of cores

- L1
- LFB
- L2
- L3
- RemoteCache
- LocalRAM
- RemoteRAM

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>L1</th>
<th>LFB</th>
<th>L2</th>
<th>L3</th>
<th>RemoteCache</th>
<th>LocalRAM</th>
<th>RemoteRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.9</td>
<td>8.2</td>
<td>9.5</td>
<td>10.0</td>
<td>11.4</td>
<td>14.5</td>
<td>16.0</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>89</td>
<td>79</td>
<td>75</td>
<td>64</td>
<td>57</td>
<td>52</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>17</td>
<td>26</td>
<td>32</td>
<td>16</td>
<td>16</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>16</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
</tbody>
</table>
Where to Optimize?

High latency

Link memory samples to FIFO channels
Outline

Dataflow Programming & Problematic

Detection of SDF Bottleneck Actors

Profiling of Dataflow Programs

Conclusion & Perspectives
Conclusion

**Contribution 1: Detect SDF bottleneck actors**

- Independent of the language
- Independent of the architecture
- Published [Selva15]

**Contribution 2: CPU/memory profiling for DF programs**

- Implementation in Orcc [Yviquel13]
- Memory profiling with the help of numap
- Conclusions about where to optimize
- Journal article to be submitted in July
Perspectives

Tooling

- Integration in the official Orcc release
- Open source `numap` (in PAPI?)

Research

- Detect bottleneck actors in less restrictive models
- Optimize dataflow runtime using memory sampling results
- Towards a dataflow aware operating system
- What about other concurrent programming models?
Thank you for your attention
Bibliography I


Bibliography III


▶ Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. Traffic management: A holistic approach to memory placement on numa systems.

➤ Tudor David, Rachid Guerraoui, and Vasileios Trigonakis. Everything you always wanted to know about synchronization but were afraid to ask. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP ’13*, pages 33–48, New York, NY, USA, 2013. ACM.
Bibliography V


Bibliography VI


Bibliography VII


Bibliography VIII

► Edward A. Lee and D.G. Messerschmitt.  
  Synchronous data flow.  

► Edward A. Lee and T.M. Parks.  
  Dataflow process networks.  

► Xu Liu and John Mellor-Crummey.  
  A tool to analyze the performance of multithreaded programs on numa architectures.  


Bibliography X


- Manuel Selva, Lionel Morel, Kevin Marquet, and Stéphane Frénot. 
  A monitoring system for runtime adaptations of streaming applications. 
  In *Parallel, Distributed and Network-Based Processing (PDP), 2015 23nd Euromicro International Conference on*, March 2015.

  Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. 
Bibliography XI


Proposal

Extend DF languages

Compile time

- Throughput expression and exploitation in SDF

![Diagram showing the throughput expression and exploitation in SDF]

Throughput: \(1000 \text{ tokens/s}\)
Proposal

- Throughput expression and exploitation in SDF
- Throughput violation and bottlenecks identification
Proposal

- Extend DF languages
- Profile app/resources
- Adapt execution choices

• Throughput expression and exploitation in SDF
• Throughput violation and bottlenecks identification
• Adaptation of actors mapping and FIFO location
1st Contrib - Related Work

Use information computed statically at runtime

Statically compute SDF maximal throughput
[Ghamarian06, Ghamarian08, Stuijk07, Bonfietti10, Bartenstein14]
Actors execution time statically known

Identify bottlenecks using FIFOs filling
[Collins09, Choi12]
Complex runtime mechanisms
2nd Contrib - Related Work

Correlate low level profiling with the DF programming model

Non NUMA specific profiling abstraction in existing APIs
[Dongarra01]

Non DF-aware profilers
[Oprofile, Perf, Lachaize2012, Liu2014]

Same ideas for a task parallel programing model
[Drebes14]
Cores Imbalance

A

B

C

D

A1

A2

A3

A4

A5

A6

A7

B1

B2

B3

B4

B5

B6

C1

C2

C3

D1

D2

D3

Context | DF & Problematic | Detection of SDF Bottleneck Actors | Profiling of DF Programs | Perspectives
---|---|---|---|---

**Diagram Description:**
- The diagram illustrates the concept of cores imbalance in a computational environment.
- Nodes A, B, C, and D represent different actors or processes.
- The arrows indicate directional relationships or interactions between these nodes.
- The cores are labeled with A1 to A7, B1 to B6, and C1 to C3, each possibly representing different levels of computation or processing stages.