NUMA Profiling for Dynamic Dataflow Applications

Manuel Selva – Lionel Morel – Kevin Marquet

CITI - INRIA SOCRATE
Université de Lyon

September 29th, 2015
In CPUs everywhere:

- Intel Nehalem - 4 cores - 2009
- Kalray MPPA - 256 cores - 2013
- Samsung Exynos - 2 x 4 cores - 2012

On the headlines:

- David P.: “The Trouble with Multicore”
- Herb S.: “Welcome to the Jungle”
- Ed L.: “The Problem with Threads”
- Timothy R.: “Mind the Gap…”
- David P.: “The Hail Mary of Programming”
But Programming them is Hard ...
• Actors exchanging data **only** through FIFO channels
• Different forms of parallelism
  • **Task**
  • **Pipeline**
  • **Data**
Dataflow Applications Examples

Medical image processing [Albers2012]

Software Defined Radio [Dardaillon2014]

Video Decoding [Lucarz09]
The setting
The question

Do DF applications scale? If not, why?
Does it scale?

- Different inputs
- HEVC decoding
- 200 frames
- 33 Actors

Number of cores vs Speedup vs single-core
What are the reasons for that?

• Are the applications well written?
  • ⇒ blame the app designer.

• Are the runtimes well implemented?
  • ⇒ blame the runtime designer.

• Is the model of computation really the good one?
  • Programmer tricked into some idiosyncracies?
  • ⇒ blame the language designer.
How to identify and understand performance bottlenecks in dataflow programs?

**Contribution:** CPU/memory profiling to analyse (and fix) bottlenecks on dataflow programs
RVC-Cal - [Yviquel13]

- Dynamic Dataflow
- Dedicated to video codec applications
- Many applications available (hevc, h264, gzip, zigbee)
- Active community
Preliminary: Dataflow Execution Model
Preliminary: Dataflow Execution Model
Preliminary: Dataflow Execution Model

1 thread per core - actors scheduled within thread
Preliminary: Which Architecture?

- Commodity HW
- NUMA
- PMU
- linux-supported
Goal: Identify and Understand Performance Bottlenecks in Dataflow Programs

Correlate hw profiling to the DF graph
CPU Profiling
Goal: Identify and Understand Cores Imbalance
Cores Balance

Input: Kimono
200 frames
Introduction

Motivation

Profiling DF Programs

Cpu Profiling

Memory Profiling

Cores Balance

HEVC
Input: Kimono
200 frames

Single actor:
Inter pred.
The application is not parallel enough:
- Split the Interframe Prediction actor! [Jerbi14]
- Split other actors as well ... 
- Parallelize the sequential code inside actors?
Total Work Time is Increasing!

Total Work Time = Sum of cpu time for all cores used

Question: where does this overhead come from?
Memory Profiling
Goal: Identify and Understand Memory Usage

Mem. traffic → FIFOs
NUMA - Performance Monitoring Unit

Introduction
Motivation
Profiling DF Programs
Cpu Profiling
Memory Profiling

Core domain
Uncore domain

Xeon X5650
PMU
Core 1
L1
L2
PMU
Core 6
L1
L2
PMU
Core 7
L1
L2
PMU
Core 12
L1
L2

L3
Mem. Ctrl
QPI
Mem. Ctrl
QPI

Memory Bank 1
Memory Bank 2
NUMA - Performance Monitoring Unit

Hardware profiling mechanisms

- Hard to program
A library for NUMA Profiling

- Memory bandwidth profiling
- Memory access sampling
Using numap for memory bandwidth usage
Using numap for memory bandwidth usage

DF applications saturate memory bandwidth?
Main Memory Bandwidth Usage

DF applications saturate memory bandwidth? **NO!**

**HEVC**
Input: Kimono
200 frames
Do you pay for too many distant accesses?
Do you pay for too many distant accesses?

Associate mem accesses to actors and FIFOs
Communication Cost

Intel X5650
Westemere

HEVC
Input: Kimono
200 frames
A small part of the accesses are responsible for a large share of the latency.
Where to Optimize?

The profiler gives us:

- High latency data exchanges at the dataflow level

We plan on using this for:

- Feeding this information to the mapping heuristics
Conclusion

Proposition

- **Main goal**: Improve scalability of DF programs
- **How**: Understand performance bottlenecks in DF programs
- **Approach**: connect HW-level performance monitoring to DF runtime

Contributions

- **numap**: memory profiling for NUMA architectures
- Connection to the **RVC-Cal** runtime
- **Memory profiling** of video decoders
Perspectives

Short-term

- Continue analysis of memory sampling results
- Build more intelligent (re-)mapping decisions

Mid- and Long-term

- Compare resource usage of DF-written decoders with traditional thread-based implementations (eg ffmpeg)
- Integrate DF notions (ie data-dependencies) into OS kernel
- Adapt runtime strategies to many-core architectures
- Run and adapt multiple DF applications simultaneously


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.
Khaled Jerbi, Daniele Renzi, Damien de Saint-Jorre, Hervé Yviquel, Mickaël Raulet, Claudio Alberti, and Marco Mattavelli.
Development and optimization of high level dataflow programs: the HEVC decoder design case.

Reconfigurable video coding on multicore.
Daniel Molka, Daniel Hackenberg, Robert Schone, and Matthias S. Muller.  
Memory performance and cache coherency effects on an intel nehalem multiprocessor system.  

Herve Yviquel, Antoine Lorence, Khaled Jerbi, Gildas Cochereel, Alexandre Sanchez, and Mickael Raulet.  
Orcc: Multimedia development made easy.  
Communication Overhead On NUMA
Communication Overhead On NUMA

Remote vs local latency +30%

[Molka2009, David2013]
Communication Overhead On NUMA

Cache coherency protocol
QPI overhead
lat. * 4

[Molka2009]
Communication Overhead On NUMA

Memory controllers and QPI links contention lat. * 5

[Dashti2013]
Why build a “dataflow” profiler?

Why not use a “regular” profiler alone?

Because they are generally too low-level:
- Distance to programmer’s thinking is too long
- May know about threads, but not actors
- Will not be aware of data dependencies between actors
Preliminary: Dataflow Actors Internals

Application graph

A → B → C

int[512] fifo_AB;
int[512] fifo_BC;
void action1() {
    int in = pop(fifo_AB);
    int out = in * 42 + 7;
    push(fifo_BC, tmp);
}

void action2() {
    ...
}

C code generated for actor B

Work time(B) = \sum_{a \in actions} \text{cpu time} (a)
Sample correlation

Application graph

A → B → C

int[512] fifo_AB;
int[512] fifo_BC;
void action1() {
    int in = pop(fifo_AB);
    int out = in * 42 + 7;
    push(fifo_BC, tmp);
}

C code generated for actor B
C code generated for actor B

```c
int[512] fifo_AB;
int[512] fifo_BC;
void action1() {
    int in = pop(fifo_AB);
    int out = in * 42 + 7;
    push(fifo_BC, tmp);
}
```

PMU Sample

PC = 0x7867BC
@ = 0xEFC234A
latency = 50 cycles
C code generated for actor B

```c
int[512] fifo_AB;
int[512] fifo_BC;
void action1() {
    int in = pop(fifo_AB);
    int out = in * 42 + 7;
    push(fifo_BC, tmp);
}
```

PMU Sample

PC = 0x7867BC
@ = 0xEFC234A
latency = 50 cycles

Dataflow Sample

B:action1
fifo_AB
latency = 50 cycles