Hardware–Software Co-Design: Not Just a Cliché

Adrian Sampson
James Bornholt
Luis Ceze
free lunch

exponential single-threaded performance scaling!
The Future of Computing Performance: Game Over or Next Level?

SUMMARY

FIGURE S.1

Processor performance from 1986 to 2008 as measured by the benchmark suite SPECint2000 and consensus targets from the International Technology Roadmap for Semiconductors for 2009 to 2020. The vertical scale is logarithmic. A break in the growth rate at around 2004 can be seen. Before 2004, processor performance was growing by a factor of about 100 per decade; since 2004, processor performance has been growing and is forecasted to grow by a factor of only about 2 per decade. An expectation gap is apparent. In 2010, this expectation gap for single-processor performance is about a factor of 10; by 2020, it will have grown to a factor of 1,000. Most sectors of the economy and society implicitly or explicitly expect computing to deliver steady, exponentially increasing performance, but as these graphs illustrate, traditional single-processor computing systems will not match expectations. Note that the SPEC benchmarks are a set of artificial workloads intended to measure a computer system's speed. A machine that achieves a SPEC benchmark score that is 30 percent faster than that of another machine should feel about 30 percent faster than the other machine on real workloads.
We’ll scale the number of cores instead.
The multicore transition was a stopgap, not a panacea.
free lunch | multicore era | who knows?
---|---|---
time | 2005 | 2015
immemorial | | ?

parallelism
data movement
guard bands
energy costs
lessons learned from Approximate Computing

New Opportunities for hardware–software co-design
lessons learned from
Approximate Computing

New Opportunities
for hardware–software co-design
new abstractions for incorrectness
new abstractions for incorrectness

type systems  debuggers  probabilistic guarantees  auto-tuning

flaky functional units  lossy cache compression  neural acceleration  drowsy SRAMs
The von Neumann curse

useful work

other crud
we don’t care about and can’t fix
Hardware design costs sanity & well-being

Thierry Moreau, FPGA design champion

[Moreau et al.; HPCA 2015]
Trust your compiler

approximate cache

[Esmaeilzadeh, Sampson, Ceze, Burger; ASPLOS 2012]
Trust your compiler

approximate cache

\texttt{st \ r1 \ x} \quad \rightarrow \quad \text{ld \ x \ r3}

\texttt{st.a \ r2 \ y} \quad \rightarrow \quad \text{ld.a \ y \ r4}

[Esmaeilzadeh, Sampson, Ceze, Burger; ASPLOS 2012]
Trust your compiler

approximate cache

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

line state bits?

[Esmaeilzadeh, Sampson, Ceze, Burger; ASPLOS 2012]
Trust your compiler

approximate cache

\texttt{st \ r1 \ x} \quad \rightarrow \quad \rightarrow \quad \texttt{ld \ x \ r3}

\texttt{st.a \ r2 \ y} \quad \rightarrow \quad \rightarrow \quad \texttt{ld.a \ y \ r4}

\text{line state bits?}

[Esmaeilzadeh, Sampson, Ceze, Burger; ASPLOS 2012]
lessons learned from
Approximate Computing

New Opportunities
for hardware–software co-design
More hardware flexibility that humans can actually program
More hardware flexibility that humans can actually program

FPGA
More hardware flexibility that humans can actually program

- explicit data movement
- explicit memory blocks
- explicit physical routing
- explicit clock frequency
- explicit ILP
- explicit numeric bit width
More hardware flexibility that humans can actually program

A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services

Andrew Putnam  Adrian M. Caulfield  Eric S. Chung  Derek Chiou
Kypros Constantinides  John Demme  Hadi Esmaeilzadeh  Jeremy Fowers
Gopi Prashanth Gopal  Jan Gray  Michael Haselman  Scott Hauck
Amir Hormati  Joo-Young Kim  Sitaram Lanka  James Larus
Eric Peterson  Simon Pope  Aaron Smith  Jason Thong  Phillip Yi Xiao  Doug Burger

Microsoft

Abstract

Datacenter workloads demand high computational capabilities, flexibility, power efficiency, and low cost. It is challenging to improve all of these factors simultaneously. To advance datacenter capabilities beyond what commodity server designs can provide, we have designed and built a composable, reconfigurable fabric to accelerate portions of large-scale software services. Each instantiation of the fabric consists of a 6x8 2-D torus of high-end Stratix V FPGAs embedded into a half-rack of 48 machines. One FPGA is placed into each server, accessible through PCIe, and wired directly to other FPGAs with pairs of 10 Gb SAS cables.

In this paper, we describe a medium-scale deployment of this fabric on a bed of 1,632 servers, and measure its efficacy in accelerating the Bing web search engine. We describe the requirements and architecture of the system, detail the desirable to reduce management issues and to provide a consistent platform that applications can rely on. Second, datacenter services evolve extremely rapidly, making non-programmable hardware features impractical. Thus, datacenter providers are faced with a conundrum: they need continued improvements in performance and efficiency, but cannot obtain those improvements from general-purpose systems.

Reconfigurable chips, such as Field Programmable Gate Arrays (FPGAs), offer the potential for flexible acceleration of many workloads. However, as of this writing, FPGAs have not been widely deployed as compute accelerators in either datacenter infrastructure or in client devices. One challenge traditionally associated with FPGAs is the need to fit the accelerated function into the available reconfigurable area. One could virtualize the FPGA by reconfiguring it at run-time to support more functions than could fit into a single device. However, current reconfiguration times for standard FPGAs
More hardware flexibility that humans can actually program
Trust, but formally verify

useful work
Trust, but formally verify

- checking that software doesn’t do anything crazy
- useful work
Trust, but formally verify

- Application
- Language
- Architecture
- Circuits

verified properties

e.g., [Hunt and Larus; OSR April 2007]
Hardware beyond core computation

- Power supply & battery
- New memory technologies
- Mobile display & backlight
- CPU
- GPU
- FPGA
- Software-defined networking
- Accelerators