From Program to Mixed SW/HW Implementation: How to Get It Right

Carl Seger

Feb. 17, 2017
Outline

- Motivation
- Problem statement
- Previous work
- Suggested solution
- Research Questions
Moore’s Law

~40%/year (double every 2 years)

Source: University of Wisconsin-Madison
Figure 1. In CPU architecture today, heat is becoming an unmanageable problem. (Courtesy of Pat Gelsinger, Intel Developer Forum, Spring 2004)
As a Result

Stuttering

- Transistors per chip, ’000
- Clock speed (max), MHz
- Thermal design power*, w

Sources: Intel; press reports; Bob Colwell; Linley Group; IB Consulting; The Economist

*Maximum safe power consumption
In Practice

Mac Pro CPU/GPU Performance Evolution

- Relative Performance to Base Mac Pro 2006 CPU/GPU

- Slowest GPU Option
- Fastest GPU Option
- Single Threaded CPU Perf (Fastest CPU)
- Multithreaded CPU Perf (Fastest CPU)
Single-Treaded Performance

**Single-Threaded Integer Performance**
Based on adjusted SPECint® results

- +52% per year

**Single-Threaded Floating-Point Performance**
Based on adjusted SPECfp® results

- +64% per year

- Intel Xeon
- Intel Core
- Intel Pentium
- Intel Itanium
- Intel Celeron
- AMD FX
- AMD Opteron
- AMD Phenom
- AMD Athlon
- IBM POWER
- PowerPC
- Fujitsu SPARC
- Sun SPARC
- DEC Alpha
- MIPS
- HP PA-RISC
Who Cares: Just Parallelize…

- Theoretical problem:
  - If NC ! = P, then there are problems that cannot be parallelized efficiently!

- Practical problem:
  - Only known algorithms are inherently sequential
    - The choice of computation depends critically (and immediately) on the result just computed.

- Even more practical problem:
  - A lot of useful and practical algorithms are highly sequential but need to be sped up!

NC = Nick’s class; problems solvable on O(n^k) machines in O((log(n))^c) time.
How to Increase ST-Performance

- Higher clock frequency
  - None: power wall has stopped this…
- Higher instructions-per-cycle (IPC)
  - Marginal: architects have pretty much wrung out most of it
- Better branch-prediction
  - Marginal: modern branch-predictors are about as good as it gets
- More advanced compilers
  - Marginal: “It’s the actual data, stupid”
- Off-load work onto (specialized) hardware
  - Potentially huge impact (2-3 orders of magnitude speed up)
Hardware Acceleration

• Alternatives:
  - New instructions in CPU
    - E.g., The AES-class instructions in x86 architecture
  - Specialized HW support in CPU (“custom CPU”)
    - E.g., Intel builds special CPUs for Facebook/Microsoft,…
  - Custom chip
    - GPUs, video codecs in chipsets
  - Field Programmable Gate Array (FPGA)
    - Traditionally on the PCIe bus, but now integrated with CPU
HW Acceleration cont.

• By 2020, it is projected that:
  ➢ Every person will create ~1.5Gbyte of data per day
  ➢ An autonomous car will create ~40Gbyte of data per hour
  ➢ 3-D sports casts will create 2000Gbyte of data per minute

• It is clear that HW acceleration is critical!

• At the same time, the algorithms used are changing and improving rapidly.
  ➢ Fixed HW is unlikely to keep up
Today:

- Microsoft Azure (cloud services) combines 1 server CPU with 1 FPGA and all communication from the CPU to the network goes via the FPGA.
- Many algorithms have been sped up by factors between 10 and 1000 times!
Today:

• Microsoft Azure (cloud services) combines 1 server CPU with 1 FPGA and all communication from the CPU to the network goes via the FPGA.

• Many algorithms have been sped up by factors between 10 and 1000 times!

• However:

At the same time, this design introduces new risks, since a bug or fault impacts the whole system. That, said [Microsoft Distinguished Engineer] Burger, has been the key challenge. "You are putting an alien technology into a very mature system. All of the network traffic runs through this thing. You screw it up, you can do some real damage."
Recall:

Ideas

Architect
- Architecture Analysis
- Development of micro-architecture

Micro-Architect
- MAS

Design Engineer
- RTL
- Schematics
- Mapping of RTL to transistors

Mask Designer
- Development of mask that yield transistors and wires

Test Engineer
- Making Silicon + Stepping(s)

Validation

Original Product Target

Chip

MAS: Micro-Architecture Specification
RTL: Register-Transfer Language

This is the theory...
Or More Realistically...

30-50% of effort

Validation

2-3 years!

Original Product Target

Target Repainted to fit Reality
Challenge

- Create a good algorithm
- Partition it into SW and HW parts
- Implement SW part
  - Remember the critical communication link with the HW accelerator!
- Implement HW version
  - Re-design several times to achieve needed performance & size
- Debug HW
- Debug SW/HW system
- Profile resulting system
- Improve HW, improve SW, re-think partition, re-think algorithm
- Repeat…Repeat…Repeat…
Bad News

- To verify HW designs is:
  - Hard
  - Time consuming
- To debug a HW design:
  - Is even worse!
- To debug combined SW/HW:
  - Is cause of short life span…
  - ..and lots of grey hair!
Good News!

- It could be worse...
What Can be Done?

- Separate “what” from “how”
  - In practice, capture the algorithm at a high level of abstraction
- Use property driven verification/testing to ensure high-level model is “correct”.
- Rely on “correct-by-construction” for common tasks
  - Introducing the interface code between SW and HW is (almost) always the same. Automate its generation!
- Incorporate verification as part of the design process
  - No “design first, verify later” (if at all)!
Questions to be Answered

• How to capture desired functionality?
  ➢ Language / level of abstraction

• How to ensure correct capture?
  ➢ Property verification / validation

• How to refine the spec. to an imp?
  ➢ Transformations / manual re-write

• How to ensure valid refinement?
  ➢ FEV / correct by design
Integrate Design and Verification

- All validation work is reactive; the design gets created somehow and now we need to figure out if it is correct.
- Rather than trying to do post-design verification, verify each step along the way.
  - Can mix “correct-by-construction” and “trust-but-verify” parts.
  - Can use different verification engines at different levels of abstraction.
  - Imposes a relatively modest overhead on the design process for a big payoff.
  - A system can be built to track the “quality” of a design from correctness point of view.

IDV prototype system for abstract RTL to layout with complete verification
Logical Design Transformations

- Add correct-by-construction implementation details
  - Examples:
    - Bypass
    - Re-timing
    - Duplication/merging of logic
    - Changing state encoding
    - Don’t care usage
    - Introducing clock gating
    - ...

- Allow arbitrary design changes when coupled with machine-checked justification
Example 1 From First IDV System

Graphics execution unit
High Level Model to Layout
HLM: 2k lines of code + 20 pages tables

Front:
1: Control decoding and data alignment
2: Partial products and CSA tree
3: CPA adder and (re-)

Back:
4: FP-adder part 1
5: FP-adder part 2
6: Dot product
7: Rounder part 1
8: Rounder part 2
9: Rounder part 3 + re-

Outside FPU:
≤ 0:   Read from register file and send data
≥ 10: Send data back to register file and write

gclk
clk
dt_latchopen
dt_latchclosed
Read
Write
FPU pipeline
FPU pipeline
Accumulator

Final placed result
~120,000 gates
Converged to strict timing

Design and verification in IDV

Bottom line: It actually works!
Example 2 From First IDV System

Top-level RTL Entry

4k to 12k lines of aRTL during 13 months

Final Design Sent to Router

>250,000 trans. Converged to 250ps

Logic And Physical View

Bottom line: During 13 months of design effort, no aRTL changes were needed because of implementation considerations.

3 designers (instead of 8)
25 FUBs
5 RF, 3 CAM EBBs
In production flow for more than 1 year
Example 3 From First System

- Integer multiplication unit
  - RTL (“How”) >3,000 lines
  - HLM (“What”) <300 lines
- Two implementations derived inside IDV
  1. To the existing RTL implementation
  2. New version using a different algorithm and partitioning
     - New version was 20% smaller than original version
     - Both provably equal to HLM and thus HLM validation was shared.

Bottom line: Rapid design exploration is made possible without extra verification cost.
Lessons Learned

• Integrating Design and Verification:
  ➢ Is technically entirely feasible
    - Requires fairly significant system to be built for approach to be practical.
    - Rapidly changing specifications are challenging, but doable.
  ➢ Allowed far more design exploration
    - First implementation took “normal” time
    - Second, third, … versions took only a fraction of initial design time.
  ➢ Requires a completely different mentality
    - Combines two roles (design engineer and verification engineer)
    - Requires a new approach to teaching design & verification

• IDV idea failed to be widely deployed inside Intel
  ➢ Project eventually cancelled.
  ➢ Likely ahead of its time…
Why Do it Again?

“Insanity is doing the same thing, over and over again, but expecting different results.”

- Narcotics Anonymous

- The short design cycle ideal for IDV
  - Trying multiple alternatives not only useful, but necessary

- The user community is entirely different
  - Training in HW design is required from day one
  - No legacy “style” in place to tear down.

- FPGA based design require much less physical design work
  - A major part of the original IDV system devoted to physical design
  - 2/3 of transformations were related to physical design aspects

- Great need for efficient techniques for developing these types of accelerated applications!
Some Further Research Questions

- What transformations do we need in the SW domain?
- What decision procedures are needed for SW refinements?
- How does an efficient “split into SW+HW” transformation look like?
  - Must it be “trusted” or can it be verified (added flexibility)
- How do we train “vertical developers” that can move seamlessly between SW and HW?
Integrate Design & Verification:

==

Catch the bugs as soon as they are created!
Thank you!

Questions?