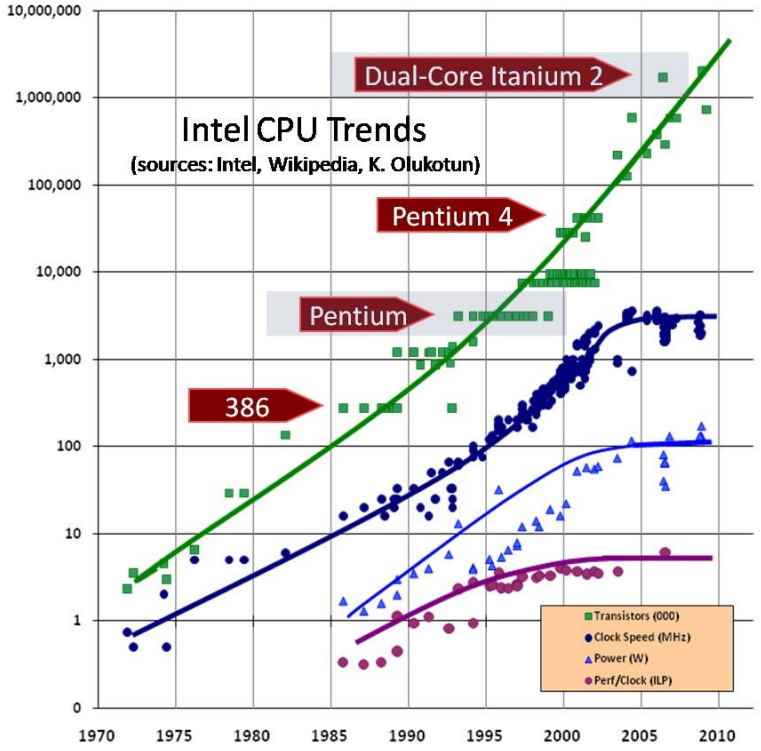
Enabling Higher Processor Performance via Architectural Simplification

Pete Foley
Ascenium

ChipEx2019 • May 13, 2019

The CPU Performance Problem

- We have all seen this chart . . .
- The combination of slowdown in Moore's law and Dennard scaling sent single threaded processor performance "off the rails" circa 2003
- Your laptop should be > 100 Ghz



"This is a golden age of computer architecture"[1]

[1] David Patterson, IEEE Spectrum, Sept 2018

Its Just Always Been Done This Way . . .

- Then along comes a huge “embarrassingly parallel” workload to distract us all – aka Deep Learning
  - Where we can go massively parallel
- But there remains a large unmet need for faster general purpose single threaded CPUs . . . .
- How did we get here?
  - The industry is locked into an old sticky contract

=> ***the ISA contract***

Old Contracts Die Hard

- This ISA (Instruction Set Architecture) contract erects a barrier between the compiler and the hardware
  - But eases development of both
  - Think of it as the "API" between the two
- This contract was first made in late '70s for x86 and early '80s for pipelined RISC architectures
  - Formed a good balance between what the compilers of the day could do, and what hardware could be built
- But, over time - this has led to a "Tyranny of the ISA"
- Yet the balance has changed over the last 40 yrs Compilers can do much more now
  - They can place, they can route, they can schedule . . .

Heroic Efforts to Increase IPC

- With heroic expenditures of $ and resources, and huge increases in complexity, the avg IPC (Instructions per Clock) has only increased from ~1 => ~1.9 (SkyLake)
- What forms has this heroic complexity increase taken?

Deep pipeline with hazard detection	Register Alias Table (RAT)
Instruction decoder	Reservations Station (RS)
Micro-op queuing, fusion, and caches	Renaming and move elimination HW
Instruction/micro-op fusion HW	Scheduler
Forwarding hardware	Different types of EUs and AGUs
Branch Prediction Unit (BPU)	Integer Register File
Branch Target Buffer (BTB)	Vector Register File
Reorder Buffer (ROB)	etc
Branch Order Buffer (BOB)	

But is this just throwing good transistors after bad?

How do we break the Tyranny of the ISA?

We Eliminate It!

A Better Approach to Improving Performance

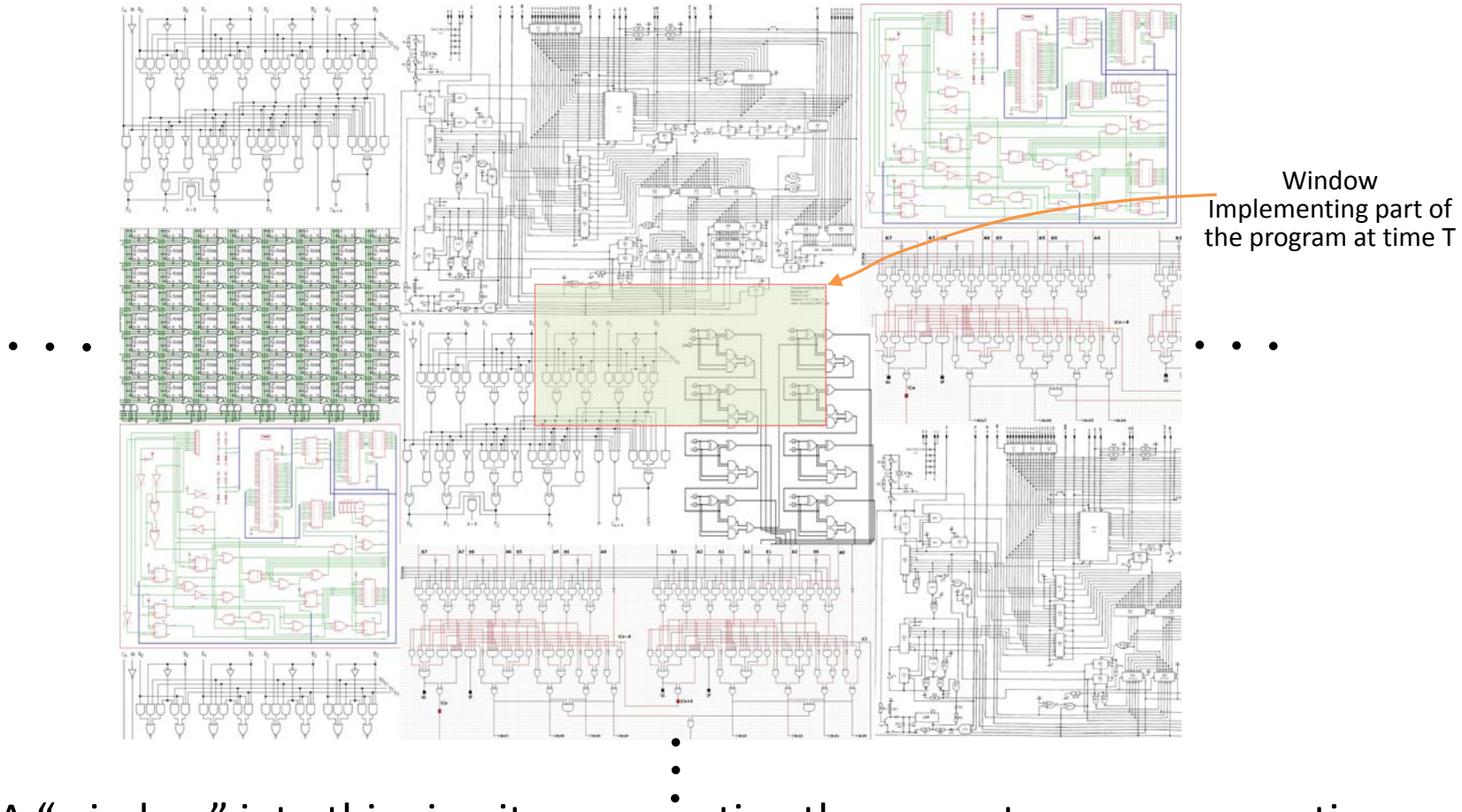
- Eliminate the ISA, the deep pipeline, and the superscalar hardware!
  - $\Rightarrow$ Let the compiler do more of the work!
- Unlocking performance via architectural simplification
  - Enabling an FPGA based proof point
- ***Aptos***, the world's first ***Software Defined Processor***
  - Aptos is a ***General Purpose*** processor ***without an instruction set***!

Opportunities Once Freed From The ISA

- Exploit far more spatial parallelism in the hardware
  - Better leverage huge # transistors available in modern processes
- Eliminate memory operations
  - Provides speedup, power savings, and enables ability to support even more parallelism on same external memory bandwidth
- Keep memory closer to computation
  - Minimize power and memory bandwidth requirements
- Build an architecture that can seamlessly and dynamically shift between SIMD (vector) and VLIW constructs
  - To dynamically & optimally adapt to complex acceleration tasks
- Enables the compiler to take a more global view
  - Resulting in better optimization

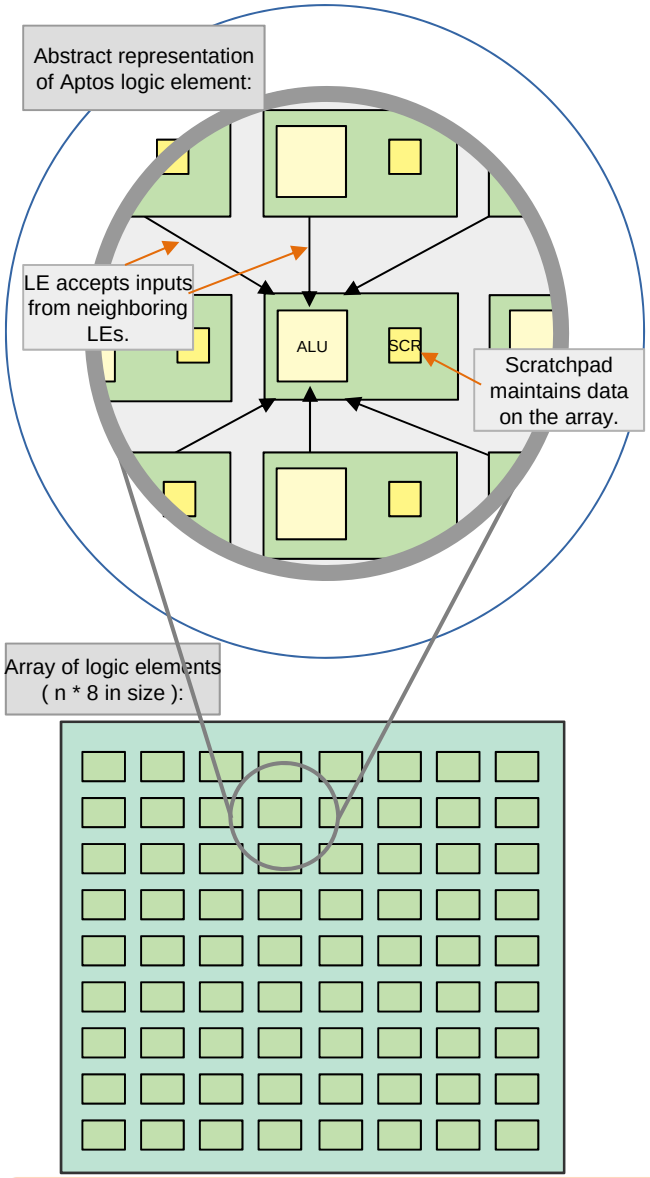
Aptos: A Disruptive Approach

- Convert software into custom hardware! Take a “global view” and convert the user’s program into an ***optimum*** circuit of potentially infinite extent



- A “window” into this circuit, representing the current program execution point, is emulated in real time in the Aptos configurable fabric

The Aptos Computation Array



- Homogenous array of 256 simple 32-bit computational elements
- Single cycle evaluation
- Tightly coupled
- Nearest neighbor connected + column & row ringbusses
- Multiplier (multicycle) resources at bottom and top of the array
- Configuration Words broadside loaded from the sides on cycle-by-cycle basis
  - Configuration size for each LE is about same as avg x86 instr: ~42bits
- D$ subsystem at the bottom

Program Test Suite

- Ascenium has assembled a suite of ~50 small to medium programs to exercise its compiler and hardware, including:

482.sphinx3	SPEC2006	18K LOC	CMU speech recognition system
401.bzip2	SPEC2006	7.3K LOC	lossless compression
998.specrand	SPEC2006	0.3K LOC	pseudo random number generation
coremark		2.2K LOC	EEMBC embedded CPU perf benchmark
mibench-sha		1.3K LOC	cryptographic hashing (bitcoin etc)
458.sjeng	SPEC2006	13.3K LOC	chess playing program
462.libquantum	SPEC2006	3.4K LOC	quantum computing, Shor's algorithm
429.mcf	SPEC2006	8.1K LOC	single depot vehicle scheduling
433.milc	SPEC2006	12.8K LOC	lattice gauge theory computation
456.hmmr	SPEC2006	33K LOC	protein sequence analysis (gene search)
470.lbm	SPEC2006	3.5K LOC	lattice boltzman method (3D fluid flow)
word2vec		1.5K LOC	2-layer neural network (word embedding)

LOC is without header files or runtime library

All compile and execute on Ascenium SW simulator and on FPGA

- Also included are 12 BDTI benchmarks, and 32 kernels from the Polybench & Polybench-NN (Neural Network) suite

- ***These programs compile and execute correctly on an FPGA implementation of our baseline hardware architecture!***

Some Performance Gain Factors

- Most inner loops are entirely captured inside the array
  - Minimizing off-array memory traffic
- Significant loop unrolling and leveraging of spatial parallelism
- Elimination of memops removes serialization points

  => Further improving performance
- Bunching memops into groups of parallel memory ops
  - 16 parallel simultaneous 64-bit load and store ops supported
- Speculative execution bandwidth cost is eliminated
- Computational wavefront propagation is omnidirectional through the array

But What About External Memory BW?

- Isn't the bandwidth to the external DRAM the great equalizer?
  - You only have 2-4 DDR4/5 channels . . .

- So improve re-use and keep data close
  - The key is to minimize accesses to the L1 D$ by keeping intermediate data in the array

  => The global view of the compiler afforded by Aptos enables quite good data re-use

Performance Examples

Examples below are hand placed and routed in Aptos

Benchmark Name	Taken From	Performance Improvement[1]	Aptos D-Cache Access Reduction	# x86 Instructions	# Aptos CWs[3]
milc (excerpt)	SPEC2006	**20x**	**3.8x fewer** (Rd+Wr)[2]	52	2
Complex FIR Filter	TI Databook	**12x**	**30x fewer** (Rd+Wr) **32x fewer** 32b Reads	73	3
libquantum (excerpt)	2017 Release	**22x**	**9x fewer** (Rd+Wr)[2]	279	1

[1] *For the code excerpt*

[2] *Estimate for the entire benchmark with optimizations folded in*

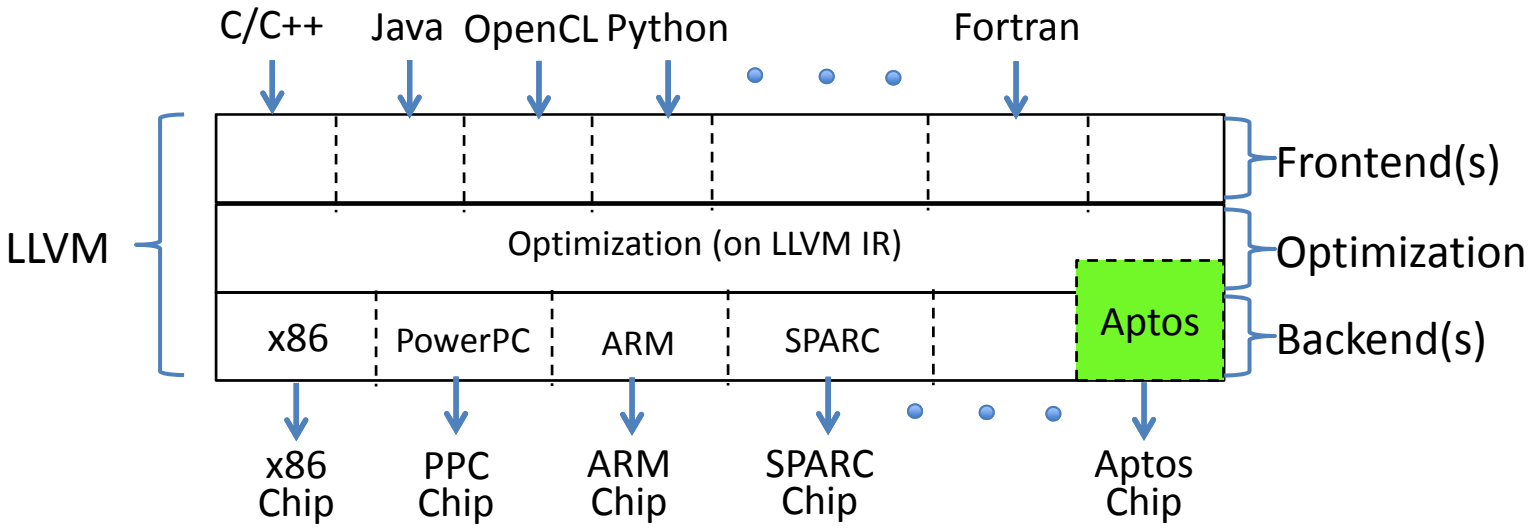
[3] *Aptos CW (Configuration Word) executes in same time as x86 instruction*

x86 code compiled using [LLVM Clang -O3 -march=Skylake] (enabling AVX2)
cycles measured on 3.8Ghz Skylake

milc performs matrix calculations on a large number of small matrices

Leveraging Standard Tools and Flows

- The Aptos compiler plugs into the industry standard LLVM compiler framework
  - Digests standard libraries (newlib, std lib, printf . . .)



- The compiler leverages information from LLVM at a higher, more abstract level than typical CPU backends to facilitate global optimizations

Technical Challenges

- Array is tightly coupled
  - For the compiler to be all seeing and all knowing, entire array must be halted on an exception
  - Micro-architecture must maintain the illusion of constant memory latency to the compiler
- Interface between compute array and L1 D$ array is the heart of the machine
  - D$ hierarchy for each column of the array
  - Can't duplicate cache lines without implementing coherency protocols between cache silo's (yuk . .)

    => Crossbar + compiler hints + split transactions
- Not a trivial compilation process
  - Compilation + 2D Placement + Scheduling + Routing
  - But the simplicity and homogeneity of the array opens the door to Constraint Programming (CSP) approaches such as SAT solvers

Compiler is All Seeing – All Knowing

- Compiler can do a surprisingly good job of resolving addresses at compile time
- Compiler can supply precedence information for D$ accesses simultaneously emerging from the array

  => More efficiently manage and/or avoid hazard issues and minimize required hazard detection and management
- Compiler can very effectively pre-fetch into read buffers, and by doing so minimize bank conflict at the D$ silos
  - Split transaction loads

What is Hard About the Compilation Process

- Extensive branch de-overlap to have a “flow forward” only implementation of C that is amenable to a virtual hardware implementation
  - Systematic extensive branch de-overalap would appear to be an NP hard problem – but Ascenium found an O(n) soln
- Dealing with pointers (of course . . . .)
- Software is not hardware
  - In a circuit everything is “always on/executing” and functionally sequential
- How to efficiently manage a finite sized window into limitless virtual 2D hardware implementation
  - Much more complex than a register window or stack spill management task

Ascenium Summary

- A truly disruptive approach to general purpose computing
  - **Unlocking performance via architectural simplification**
- Aptos is a general purpose processor without an instruction set
  - With 10x the performance of legacy deeply pipelined processors - on the same external memory bandwidth at lower power and lower cost
- Using a compiler centric approach with simpler regular hardware
  - Compiling unmodified standard HLL programs
- Key among many advantages are
  - High spatial parallelism
  - Significant reduction in memory operations
  - The ability to implement many compiler optimizations enabled by a much larger program view/window
  - Significant hardware simplification, which also enables FPGA proof point
  - Much improved security
- Initial go-to-market target is datacenter compute acceleration
  - As existing alternatives (i.e. GPU, FPGA, ASIC) are more difficult to program
  - To reduce datacenter energy usage and improve performance

THANK YOU!

Q&A