Dataflow: The Road Less Complex

Steven Swanson
Andrew Schwerin
Ken Michelson
Mark Oskin
University of Washington

Sponsored by NSF and Intel

Things to keep you up at night (~2016)

- Opportunities
  - 8 billion transistors; 28Ghz
  - One DRAM chip will exhaust a 32bit address space
  - 120 P4s OR 200,000 RISC-1 will fit on a die.
- Challenges
  - It will take 36 cycles to cross the die.
  - For reasonable yields, only 1 transistor in 24 billion may be broken, if one flaw breaks a chip.
  - 7yrs and 10000 people

Chips are networks

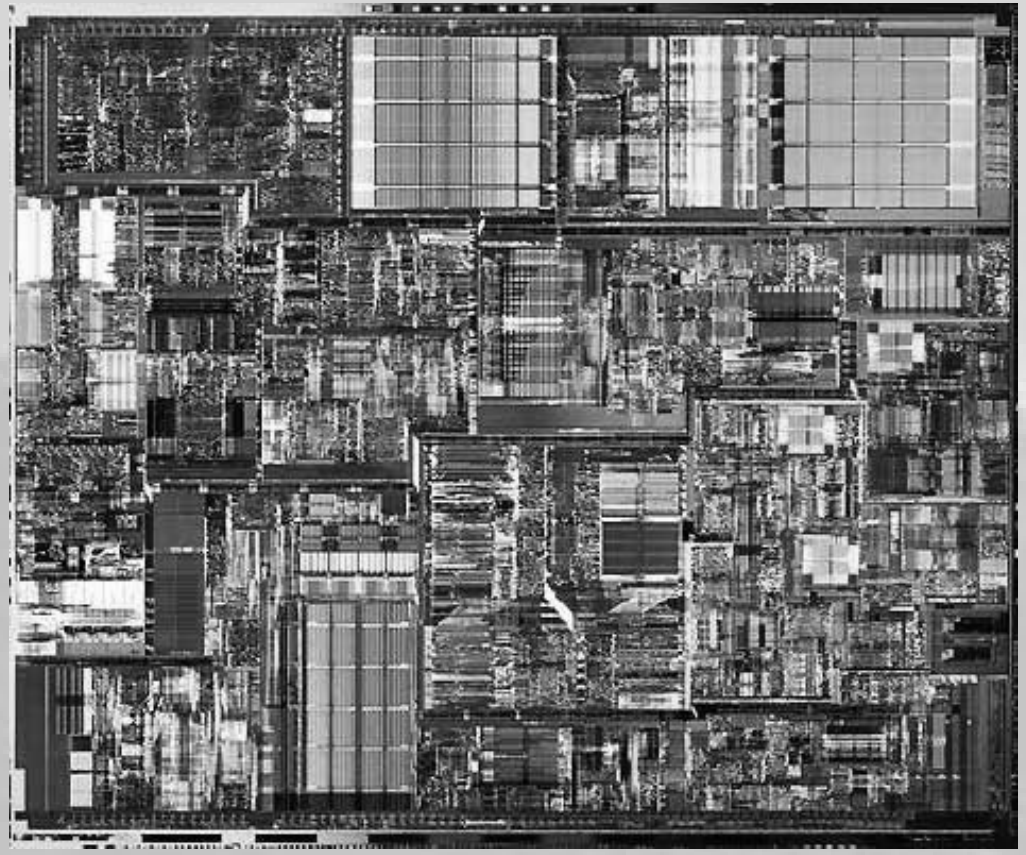
Fault tolerance is required

Simpler designs
Better tools

Outline

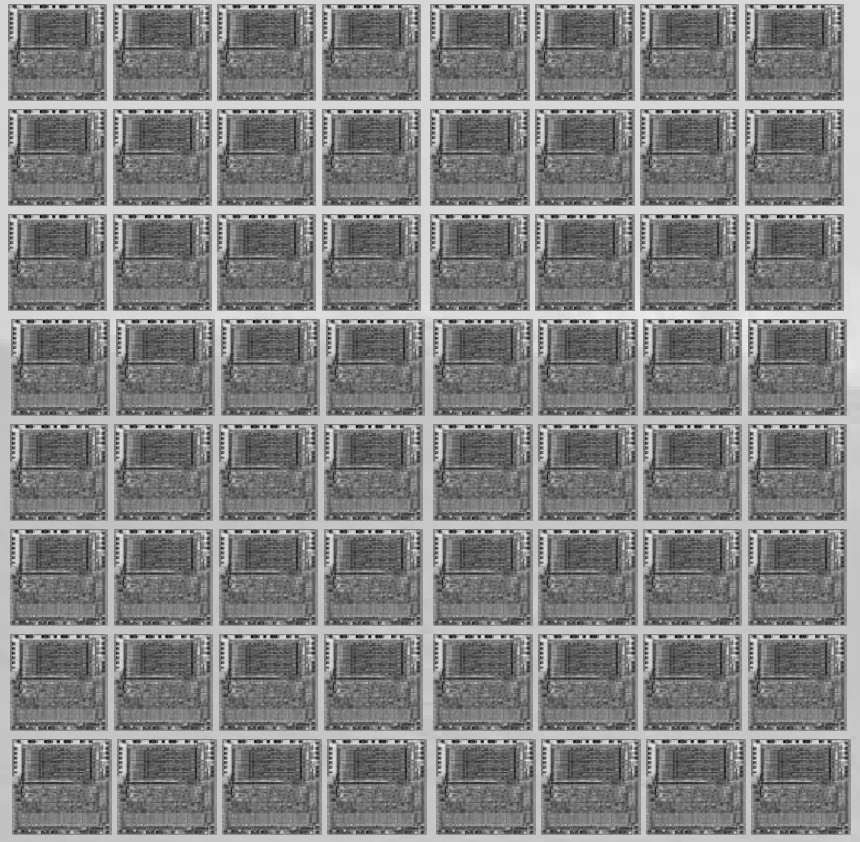
- Monolithic von Neumann processing
- WaveScalar
- Results
- Future work and Conclusions

Monolithic Processing



- Von Neumann is simple.
- We know how to build them.
- 2016?
  - ☹ Communication
  - ☹ Fault tolerance
  - ☹ Complexity
  - ☹ Performance

Decentralized Processing



- ☺ Communication
- ☺ Fault tolerance
- ☺ Complexity
- ☺ Performance

The Problem with Von Neumann

- Fundamentally centralized.
- Fetch is the key.
  - There is only one program counter.
  - There is no parallelism in the model.
- The alternative is dataflow

Dataflow has been done before..

- Dataflow is not new
- Operations fire when data is available
- No program counter
- No false control dependencies
- Exposes massive parallelism
- But…

...it had issues

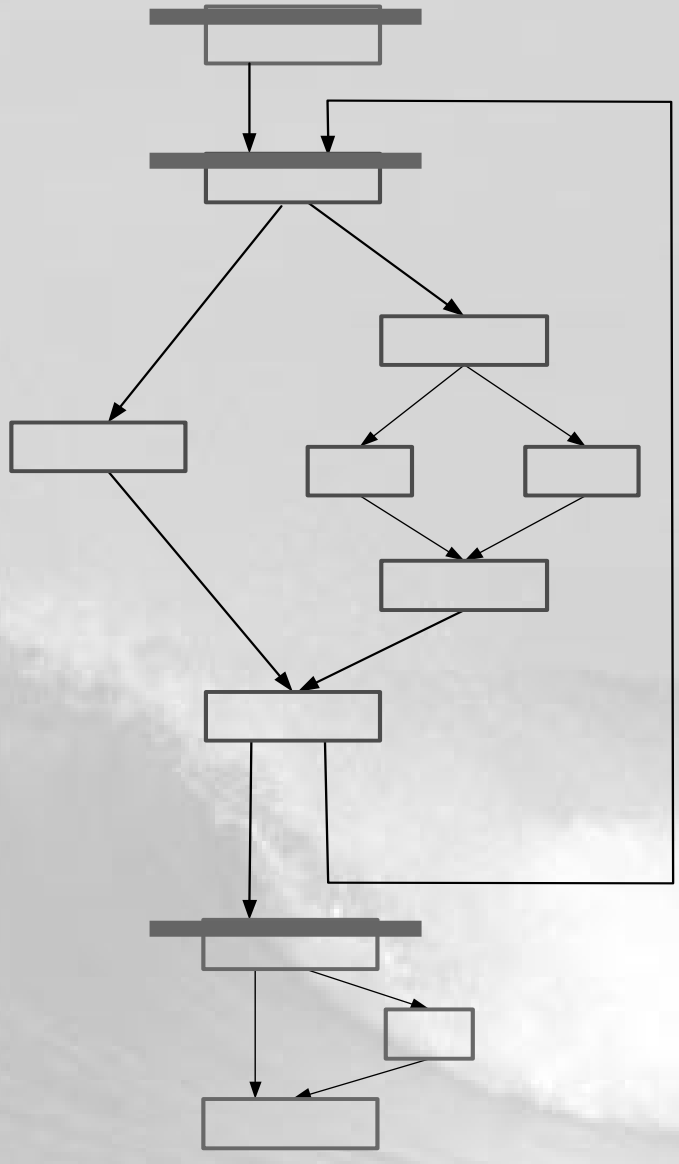
- It never executed mainstream code
  - Special languages
    - No mutable data structures
    - No aliasing
    - Functional
    - Strange memory semantics
- There are scalability concerns
  - Large slow token stores

The WaveScalar Model

- WaveScalar is *memory-centric Dataflow*
- Compared to Von Neumann
  - There is no fetch.
- Compared to traditional dataflow.
  - Memory ordering is a first-class citizen.
  - Normal memory semantics.
    - No I-structures or special languages.
    - We run Spec.

What is a wave?

- Maximal loop-free sections of the dataflow graph.
  - May contain branches and joins.
  - They are bigger than hyperblocks.
- Each dynamic wave has a *Wave number.*
  - Every value has a wave number.



Maintaining Memory Order

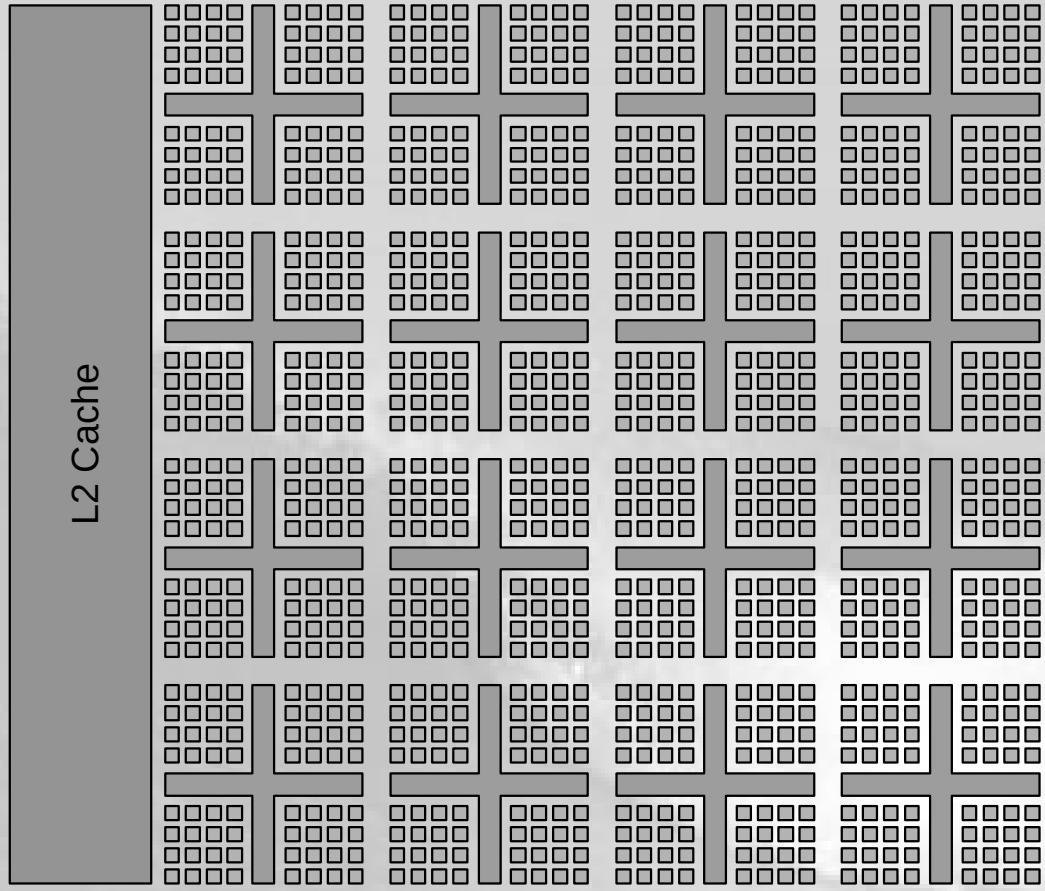
- Loads and stores can issue requests to memory in any order:
  - Wave number.
  - Operation sequence number.
  - Ordering information (predecessor and successor sequence numbers).
- The memory systems reconstructs the correct order.
  - Wave number+sequence numbers provide a total order
- Your favorite speculative memory system.
  - Or a store buffer.

WaveScalar benefits

- Expose everything about the program
  - Data dependencies
  - Memory order
    - Instructions manipulate wave numbers.
    - Multiple, parallel sequences of operations are possible.
  - Synchronization
  - Concurrency
  - Communication

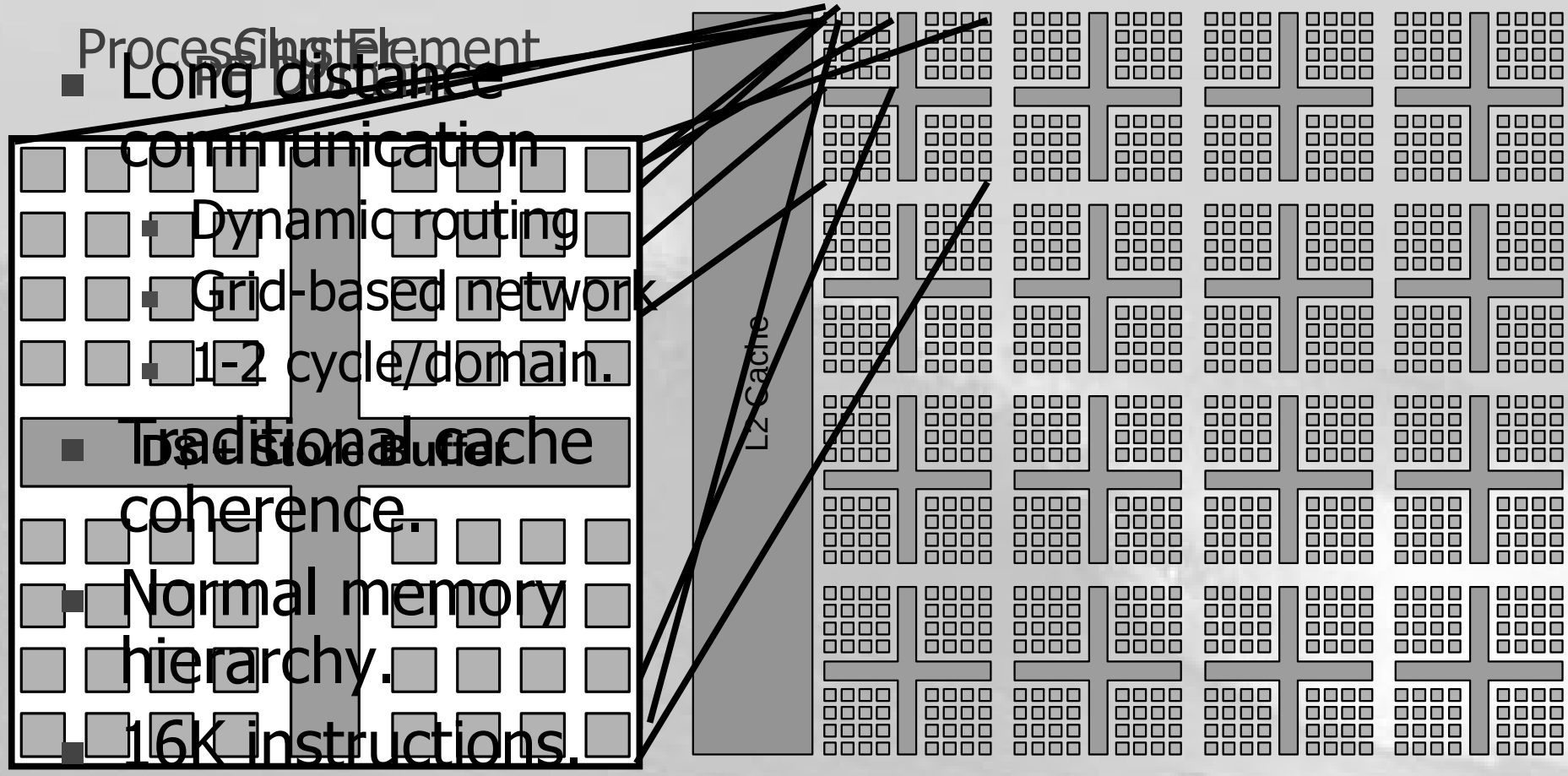
The WaveCache

The I-Cache *is* the processor.



WaveCache

- Long distance communication
  - Dynamic routing
  - Grid-based network
  - 1-2 cycle/domain.
- Traditional cache coherence.
- Normal memory hierarchy.
- 16K instructions.



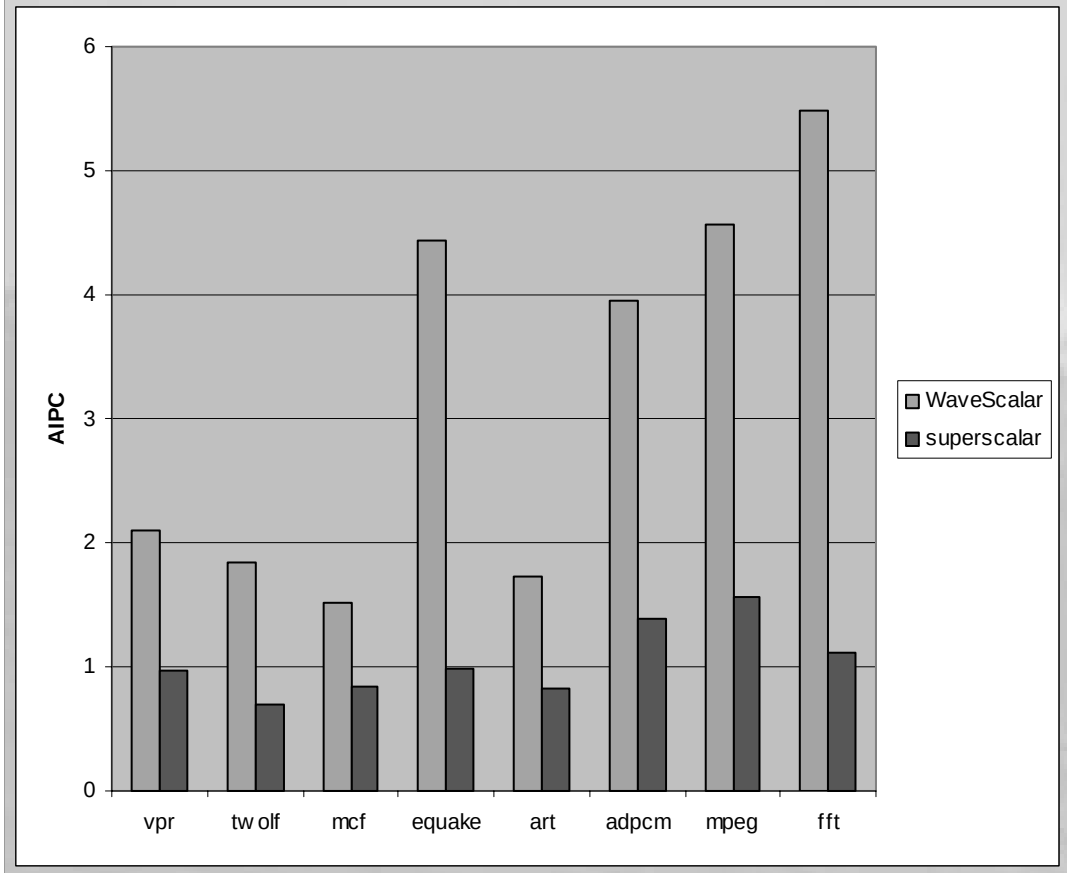
Current results

- Compiled SPEC/mediabench
  - DEC cc compiler (-O4 -unroll 16)
- Binary translator/compiler
  - From Alpha AXP → to WaveScalar
- Timing/execution-based simulation.
- Results in *alpha instructions per cycle* (AIPC)

Comparison architectures

- Superscalar
  - 16-wide, 16 ported cache, 1024 issue window, 1024 regs, gshare branch predictor
  - 15 stage pipeline.
  - Perfect cache.
- WaveCache
  - ~2000 processing elements
  - 16 elements/domain
  - Perfect cache.

WaveScalar vs Superscalar

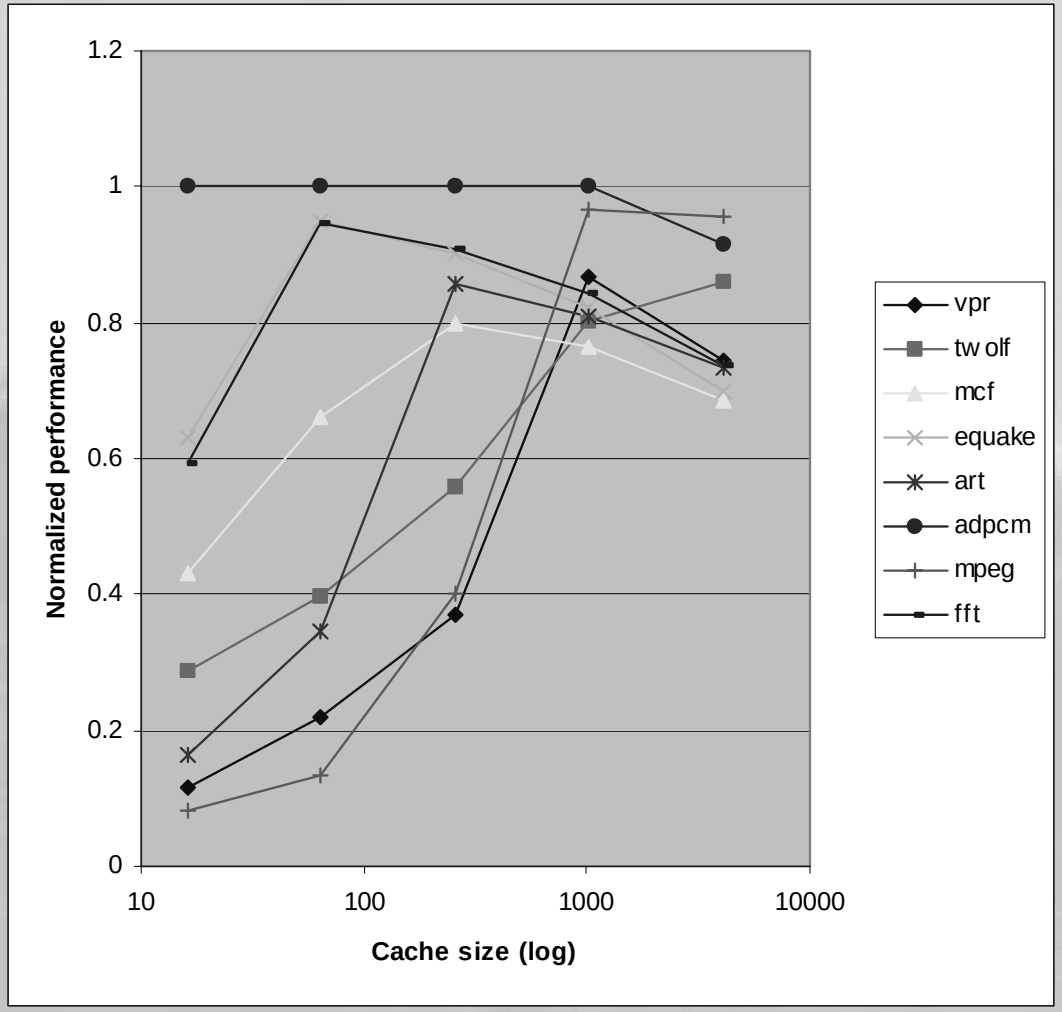


- 2.8x faster
- Not counting clock rate improvements.

Cache replacement

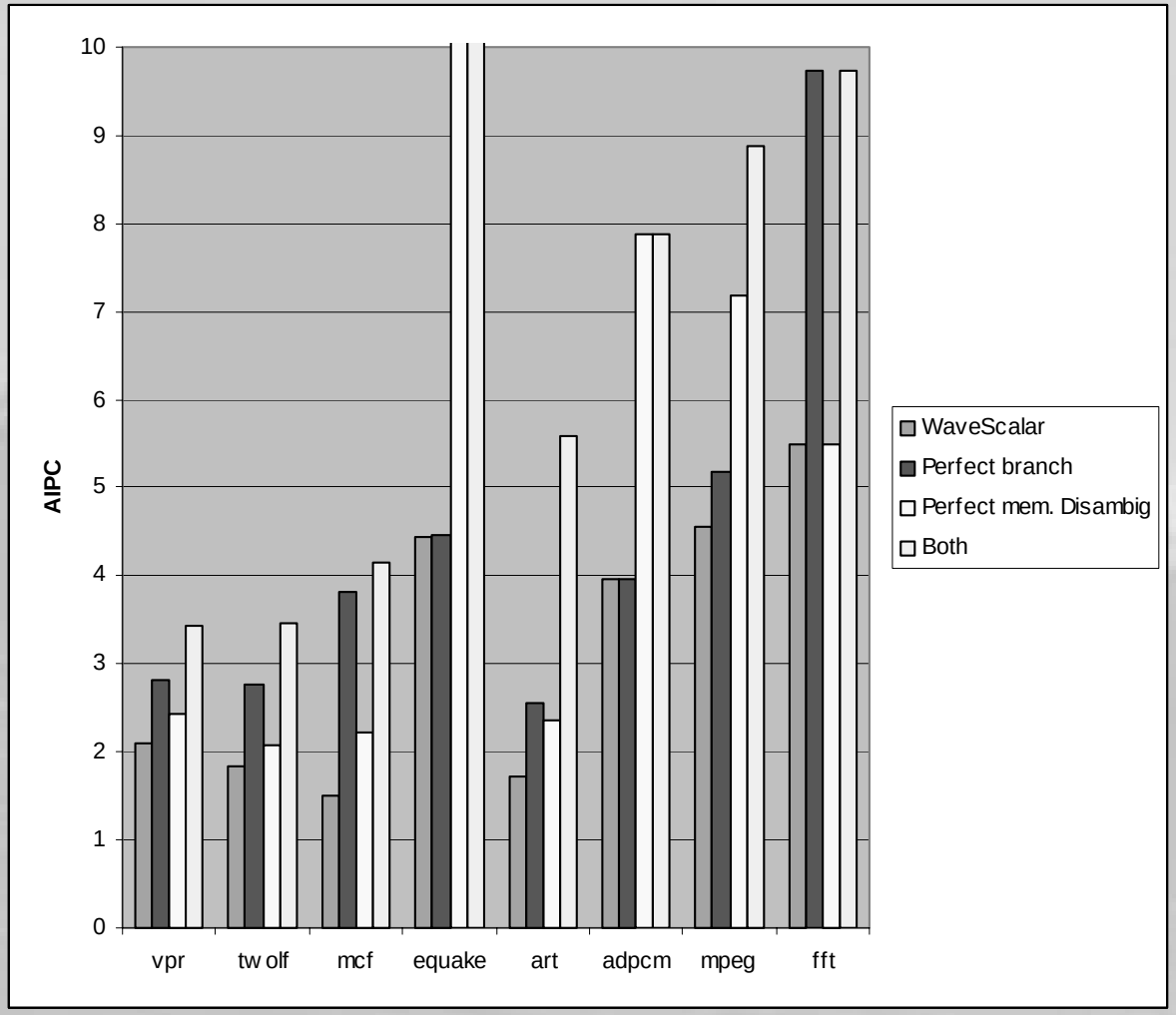
- Not all the instructions will fit.
- WaveCache miss
  - Destination instruction is not present
  - Evict/Load an instruction (flush/load queues)
- Instructions volunteer for removal
- Location is important
  - Normal hashing won’t work

Cache size



- Thrashing is dangerous
- Dynamic mapping is a big win.

Speculation



- Speculation helps
  - 2.4x on average for both
- This is gravy!!

Future work

- Hardware implementation
  - A la the Bathysphere
- Compiler issues
  - Memory parallelism
  - More than von Neumann emulation
    - Vector
    - Streaming
  - WaveScalar is an ISA for writing architectures.
- Operating system issues
  - What is a context switch?
  - What is a system call?

Future work

- Online placement optimization
  - Simulated annealing
- Defect tolerance
  - Hard and soft faults
- WaveCache as a computer system.
  - WaveScalar everything (graphics, IO, CPU, Keyboard, hard drive)
  - Uniform namespace for a computer.
  - Adaptation at load time.

Conclusion

- Decentralized computing will let you rest easy in 2016!
- WaveScalar and the WaveCache
  - Dataflow with normal memory!!
  - Outperforms an OOO superscalar by 2.8x
  - Feasible now *and* in 2016
- Enormous opportunities for future research