.

Amdahl's Law: Not Just an Equation

June 2, 1997

James E. Smith

Dept. of Elect. and Comp. Engr.
1415 Johnson Drive
Univ. of Wisconsin
Madison, WI 53706

jes@ece.wisc.edu
http://www.engr.wisc.edu/ece/faculty/smith_james.html

Quiz

In his famous 1967 talk, Gene Amdahl said:

a) "$$Speedup = \frac{1}{\frac{Fraction_{enhanced}}{Speedup_{ehanced}} + (1 - Fraction_{enhanced})}$$"

b) "in vector processors you must have good scalar performance"

c) "make the common case go fast"

d) "demonstration is made of the continued validity of the single processor approach and of the weakness of the multiple processor approach in terms of application to real problems and their attendant irregularities"

The correct answer:

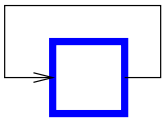
d -- *Build powerful uniprocessors!*

Outline

- The Next Big Thing
  - Future generation processor organization
- Examples
  - Evolutionary and revolutionary
    - *and* an historical precursor
- Future Microarchitecture Technologies
  - Where future gains can be made
  - Exploiting instruction level parallelism
  - Enhancing ILP

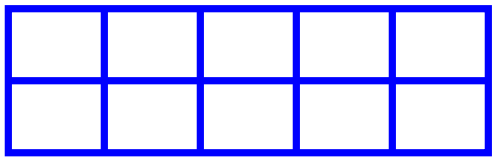
The Next Big Thing: 4th Generation Microarchitecture

- Generation 1: Serial Execution

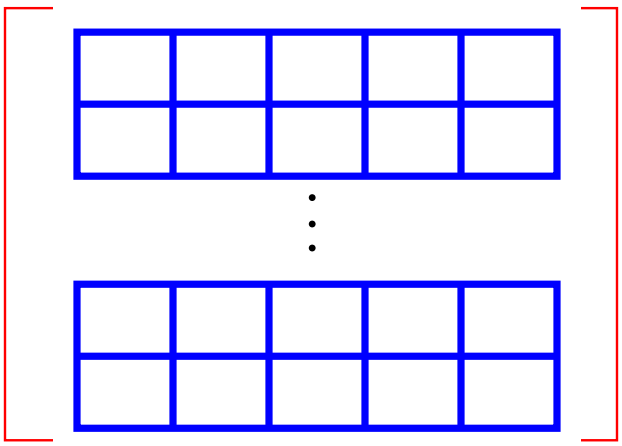


- Generation 2: Pipelining

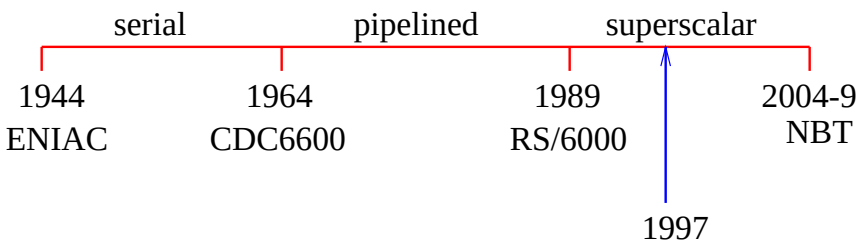
- Generation 3: Superscalar



- Generation 4: Distributed Superscalar/Multiscalar



Historical Perspective



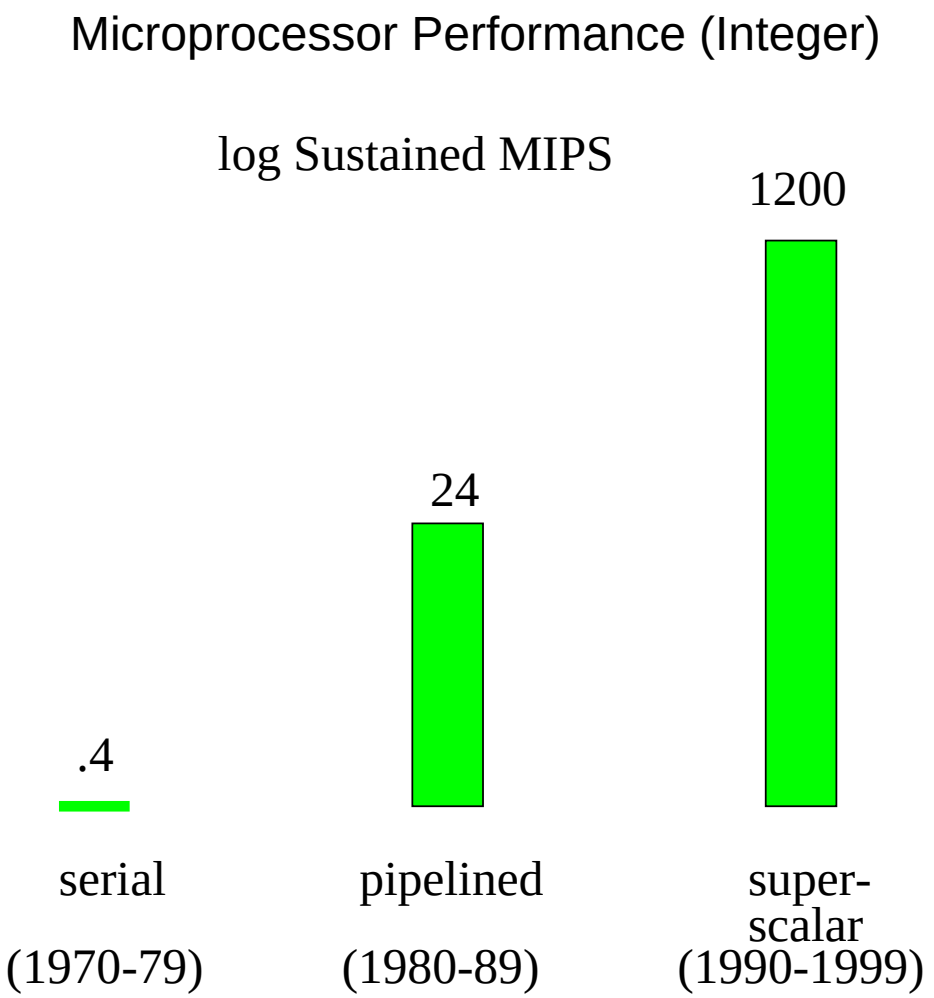
- 20 years separate major microarch. generations

  Although there are "experimental" precursors

- Research for the next generation only begins about half way through the current one

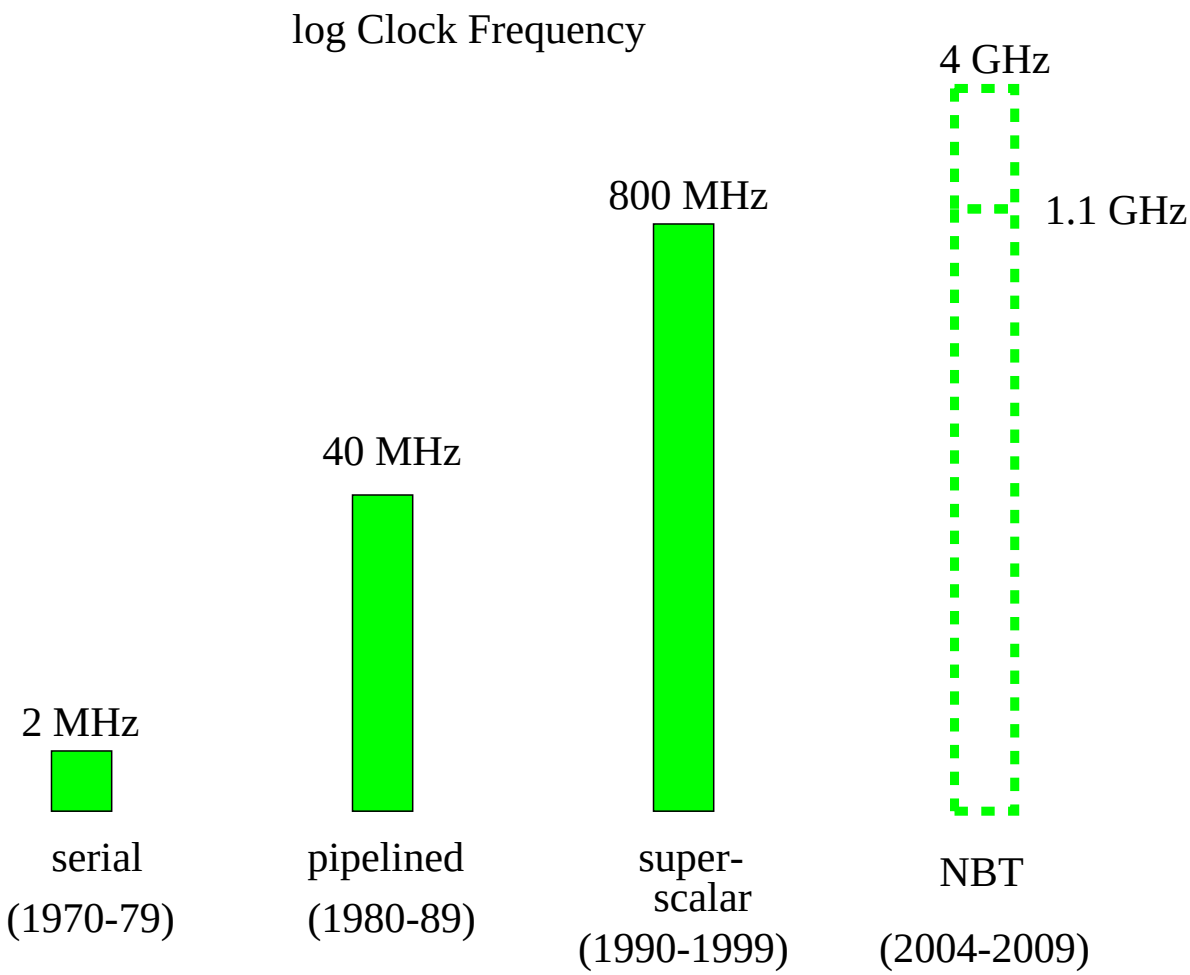
  => now is prime time for research

Performance



- Overall performance has increased exponentially
- BUT, consider microarchitecture (IPC) contribution and technology (clock freq.) contributions

Performance, Clock Frequency

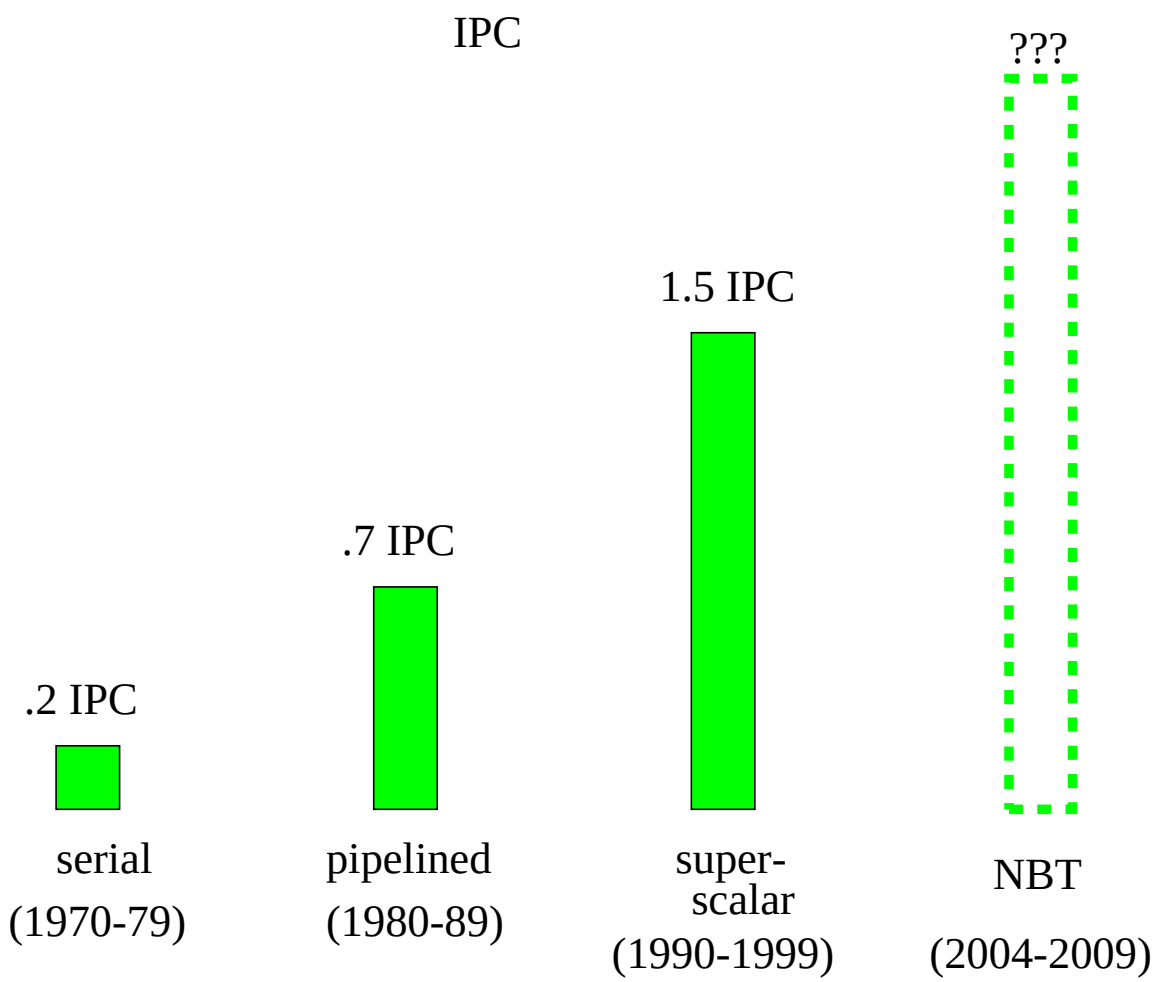


- Improvement has been exponential,

  BUT, is falling off, even with more optimistic projections

  SIA Roadmap projects 1.1 GHz in 2010.

Performance, IPC



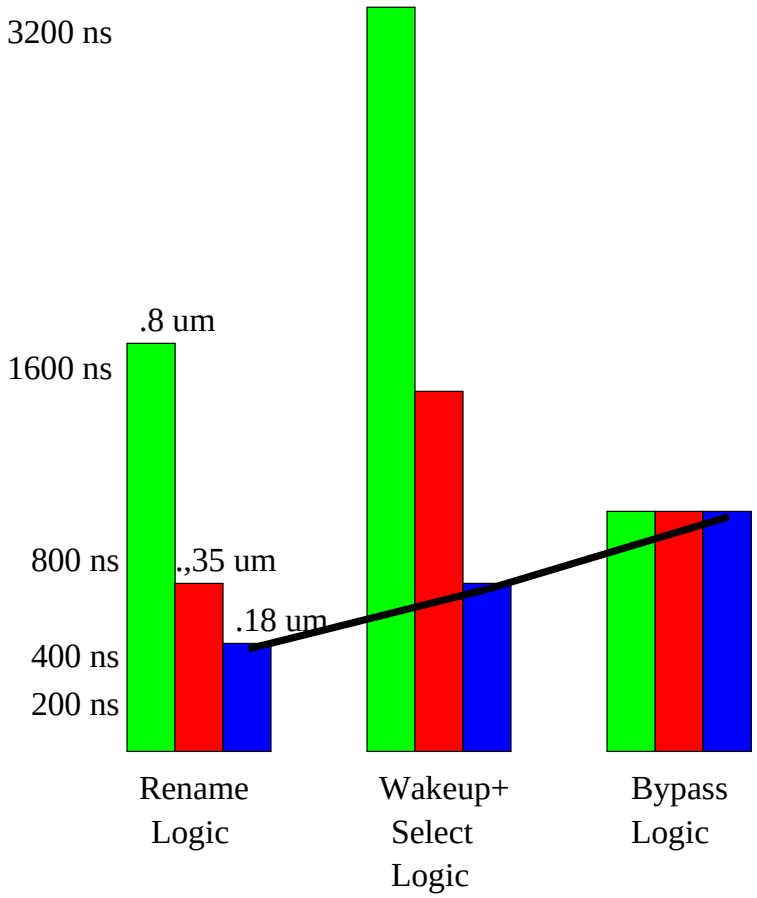
- Improvement slightly less than exponential,

  BUT, 40 years of mainframe advances compressed into 20 years

- Microarchitecture is now under pressure to "produce"

Technology Trends and Impact

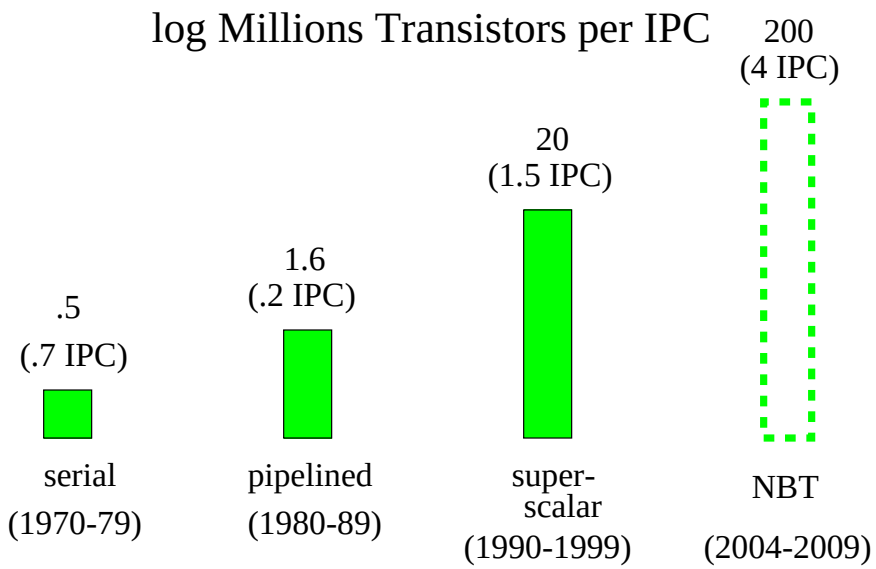
8-way issue; 64 window



- Wire delays don't change with smaller feature sizes
- Bypass delays become dominant in the future
- Wakeup + select important *after* bypasses

Billions and Billions of Transistors

- How much does IPC cost?



- Exponential cost per IPC
  (Although this includes on-chip caches)

- And don't forget design costs...

  From the SIA Technology Roadmap:

  "there is a general belief that a crisis is approaching when product complexity or time-to-market will depend more on the limits of design tools than on the underlying wafer fabrication technology"

Future Microarchitectures: Underlying Structure

- Communication Locality
  - minimize communication delay
  - reduce bypasses
- Hierarchy
  - distributed control
- Replication
  - reduce design/test time
- RAM structures for performance
  - space/time tradeoff
  - reduce design/test time

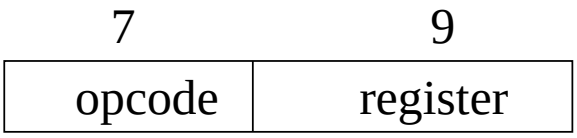
Future Microarchitectures: More ILP

Lots of transistors and good structure are not enough

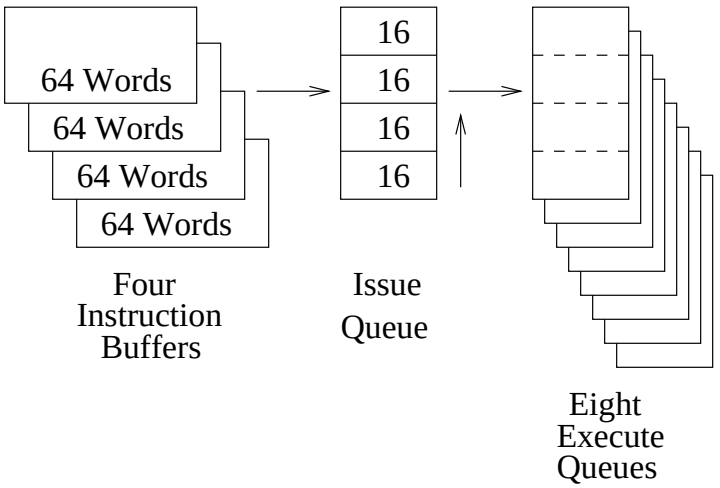
- Very wide issue window
- Control dependences
  - good prediction => accurate window
  - hierarchical control => hierarchical prediction
- Data dependences
  - prediction
  - latency reduction
  - caches
- Ambiguous memory references

Historical: Early Cray-2

- A circa-1979 Cray-2 design that was abandoned
  Vestiges appeared in the Cray IOP
- Two level register hierarchy
  512 General registers
  Single accumulator
- Single accumulator instruction style



Early Cray-2, cont



- Implementation has 8 physical accumulators

  Execution queue for each physical accumulator

- Dependent instructions are placed in same queue

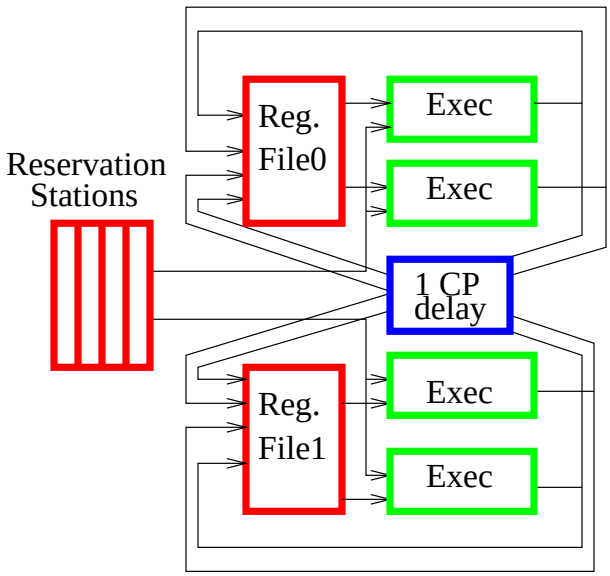
  All use same physical accumulator

  Parallelism is across queues

  Compiler groups dependent instructions *together*

- Essentially a 4-issue o-o-o machine (in 1979)

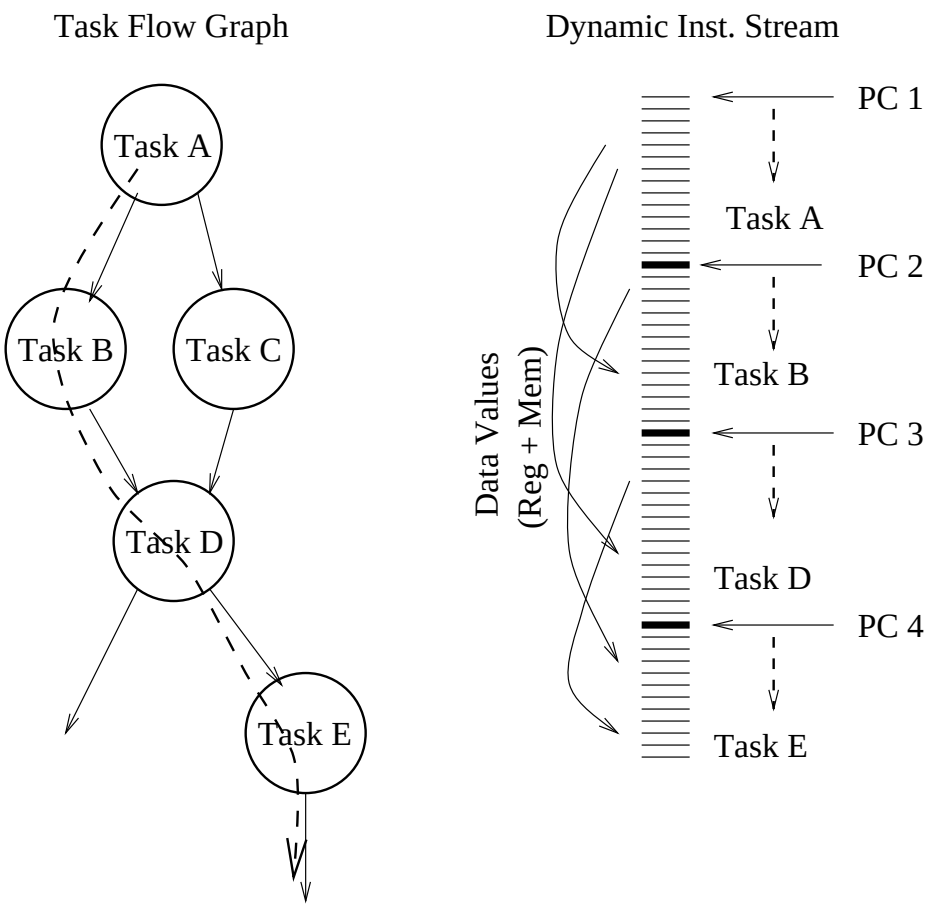
Evolutionary: Alpha 21264



- Leading edge designs hit limits first
- Two "Clusters" of Registers/Units
- Inter-cluster bypass => 1 cycle
- Instructions are enabled to go to either queue

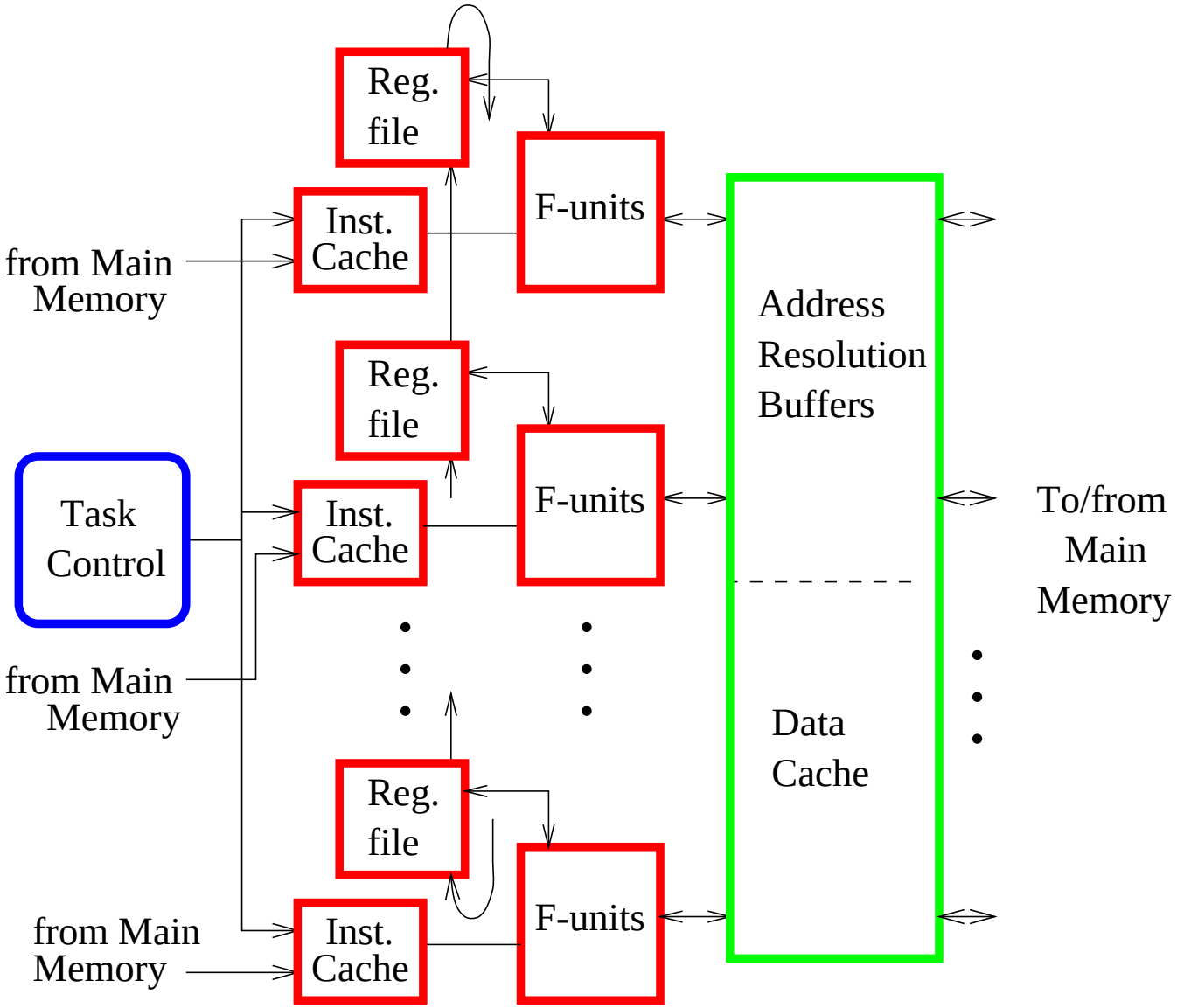
  Dependences will tend to draw dependent instructions to same cluster

Precursor: Multiscalar



- Partition program into *sequential* tasks
- Initiate tasks according to *predicted* order
  => multiple PCs at intervals down the sequential program
- Communicate data values mostly via hardware
  (with some compiler assist)

Multiscalar Implementation



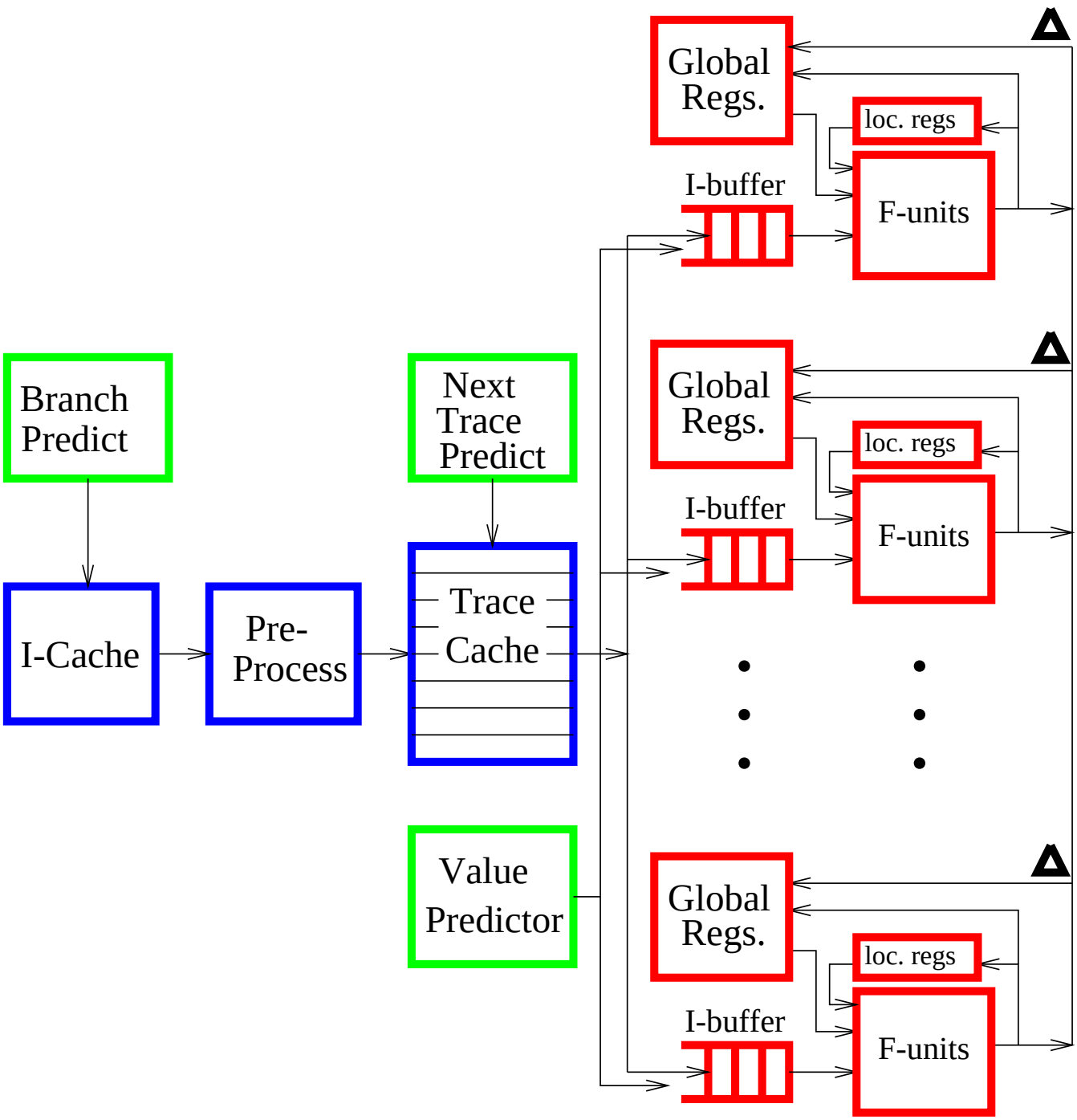
Multiscalar Features: ILP

- Wide Instruction Window
  - Processing units fetch/issue in parallel
  - Task prediction opens window quickly
- Control dependences
  - Task speculation over multiple branches
  - many branch predictions internal to a task
- Memory disambiguation
  - Address resolution buffer
  - speculates address independence

Multiscalar Features: Structure

- Uses hierarchy
  - Task dispatch unit drives processing elements
- Communication locality
  - Register values held and used locally
  - otherwise passed around ring
- Distributed control
  - Parallel processoring units dispatch/issue multiple instruction streams
- RAM structures for performance
  - Task prediction tables
  - Task header cache
  - ARB
  - I and D Caches

Next Big Thing: Trace-driven MS



Trace-driven Paradigm

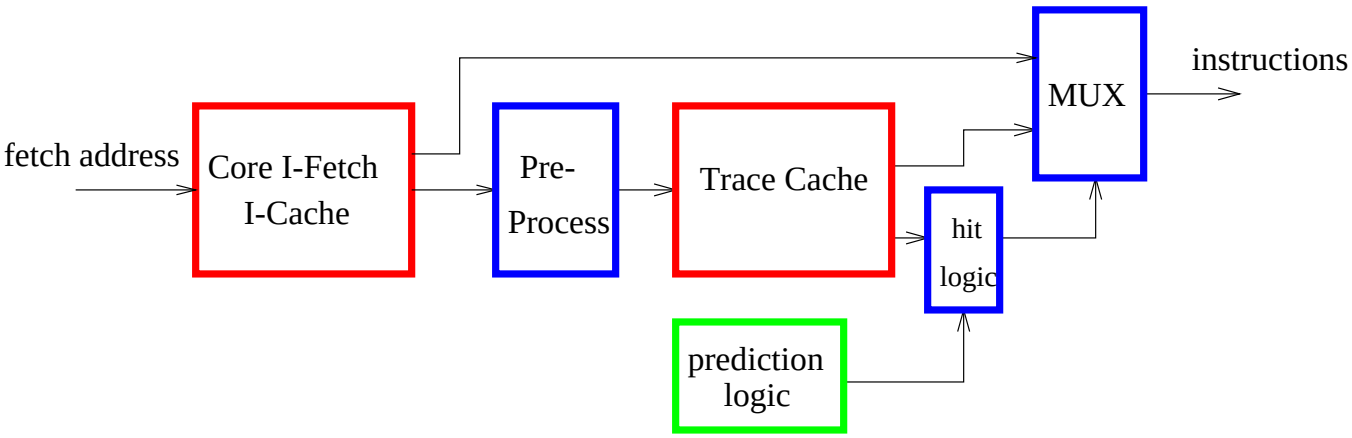
- Provides an alternative research vehicle
- Like multiscalar with tasks, "pre-unwound"
- Hardware forms traces (tasks) from ordinary binary instruction stream

  capable of strict binary compatibility
- Pre-processing phase during unwinding
- Structured data prediction
- Physical register file is replicated and hierarchical
- Memory system like multiscalar

Enabling ILP Technologies

- Wide Issue Window
  - Distributed across multiple traces
  - Trace-caches
- Control dependence
  - Next trace prediction
  - Dual path execution
- Data dependence
  - Instruction collapsing
  - Data value/address speculation via pre-processing
  - Advanced data caching
  - Advanced memory disambiguation
  - Prediction confidence methods
- Ambiguous memory references
  - ARB-like devices

Trace Caches



- Conventional, core I-fetch unit

  Used for building dynamic traces

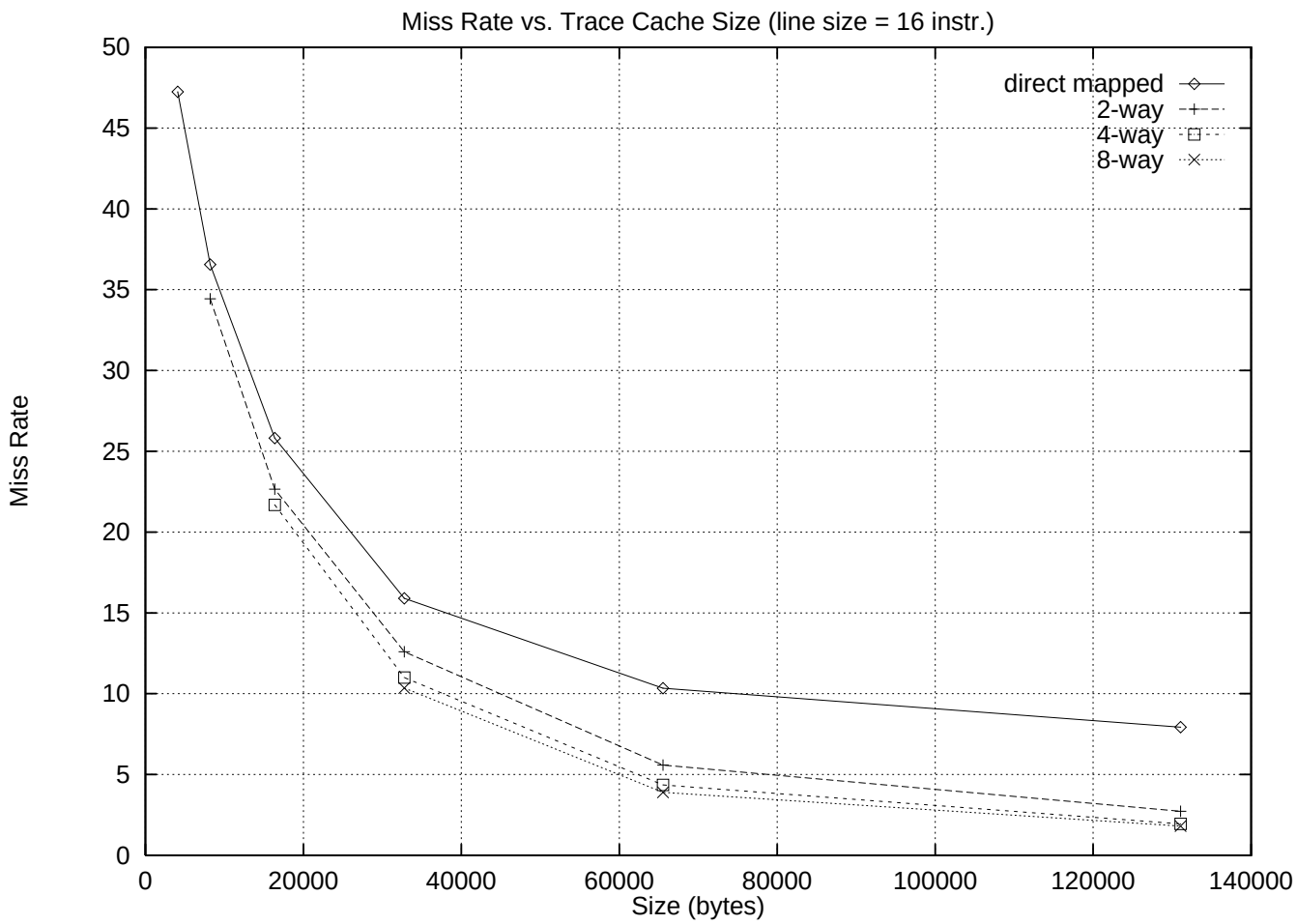
- Trace cache captures dynamic traces including internal predicted branches

- Predict and execute traces based on history

- Pre-processing

  Always?
  Optionally?
  Let processing => pre-processing?

Trace Cache Performance

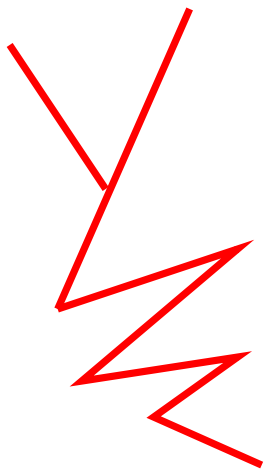


- Associativity helps
- 64K bytes => miss rates < 5%

A Lesson from Dr. Seuss

- Beyond Z:

  You'll be sort of surprised what there is to be found
  Once you go beyond Z and start poking around!

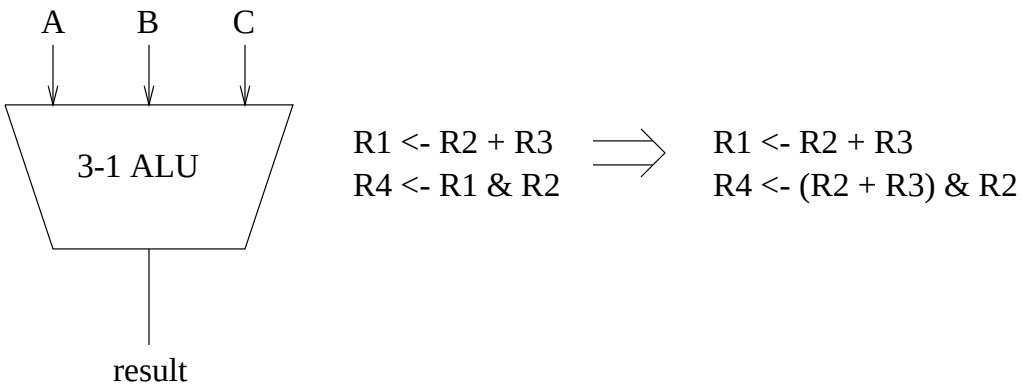


- There are pipe stages before I-Fetch!

Instruction Pre-Processing

- Make use of space/time tradeoff
- Some instruction re-ordering can be done
  - Reorder buffer contents can be pre-calculated
  - Some load/store buffer information can be pre-calculated
- Dependent instructions can be collapsed
- Renaming for local registers can be done

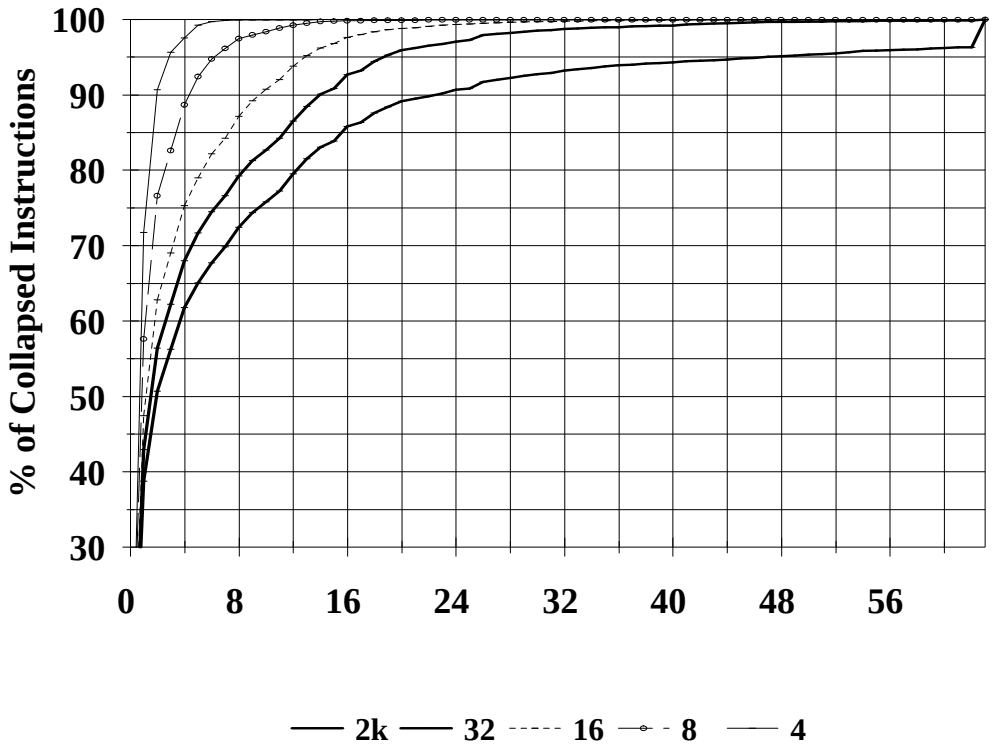
Pre-Processing: Instruction Collapsing



- Non-consecutive instructions can be collapsed (via pre-processing)
- Collapsing can take place across basic block boundaries

Instruction Collapsing

- Distance Between Collapsed Instructions



window sizes/issue widths vary

Next Task/Trace Prediction

- Higher level prediction than single branches
  - A task or trace may contain internal branches
- Multiple outcomes (not just taken/not taken)
- Collect path information
  - Shift multiple PC bits into shift register
  - More recent PCs get more bits
  - Fold register and XOR before accessing table

Preliminary Results (gcc)

- Trace Predictor:

  predicts inter-trace transfers

  avg. 1.67 cond. branches per trace

  GCC conventional branch prediction: 7-8%

predictor	miss %	equiv. branch %
Path based	14.9	8.9
Advanced path based	9.1	5.7
Add some tricks	8.0	4.8

- Predict second trace

  Avoids slow trace construction process

  Aggregate miss rate: 3.4%

Data value/Address speculation

- Concept:

  Attempt to "beat" the critical dependence path

- Use confidence measures to decide whether to speculate.

  Because certain address and values will be *very* difficult to predict

- Sequence Models

Constant:	5 5 5 5 5 5
Stride:	1 2 3 4 5 6 7 8
Non-stride:	1 4 3 9 2 7
Repeated stride:	1 2 3 1 2 3 1 2
Repeated non-stride:	1 4 3 1 4 3 1 4

Predictor Types

- Functional (constant, stride)

  prediction is a computed from previous value

  can predict a value -- never having seen it before

- Patterns (correlation)

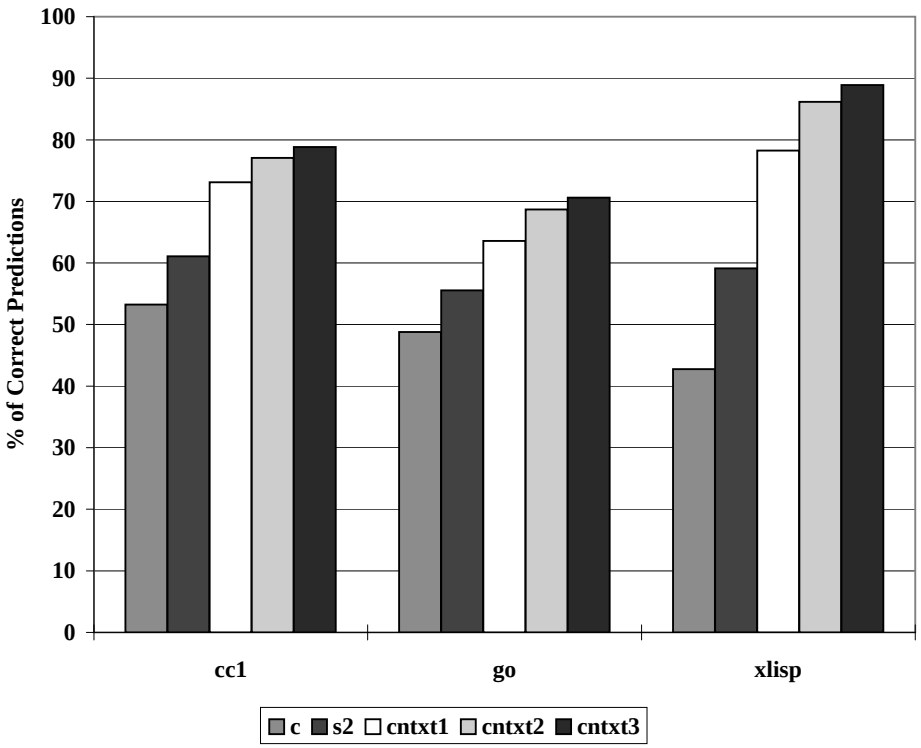
  capture repeating behavior in tables
  (a value must be seen before it can be predicted)

  Patterns of different orders can be matched:

  order 1: "Y" often follows "X"
  order 2: "Z" often follows "XY"

- One tradeoff: accuracy versus learning time

Performance



- Ideal assumptions:
  - unbounded table sizes
  - immediate update
  - (significant engineering remains)
- Strides prediction outperforms constant
- The pattern models work best
  - higher order models work better

Other Types of Prediction

- Ambiguous dependences

  Allow reads to pass writes before addresses are computed

  If true addresses reveal conflict, backup and restart

  Used in multiscalar memory system <Moshovos, Sohi 97 ISCA>

- Memory consistency

  Allow memory references out of order assuming no sequential consistency problems

  If ordering problems are detected (via cache coherence events), then backup and restart

  <MIPS R10000>>

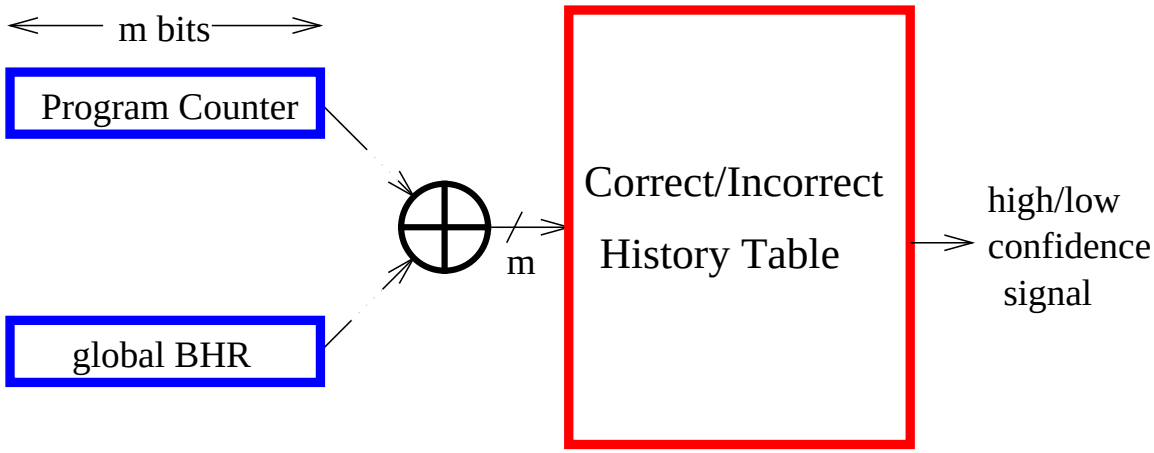
Confidence in Predictions

- Speculation typically consumes resources

  wrong prediction => wasted resources
  (and reduced performance)

- Attach a confidence to a prediction
  and speculate only when highly confident

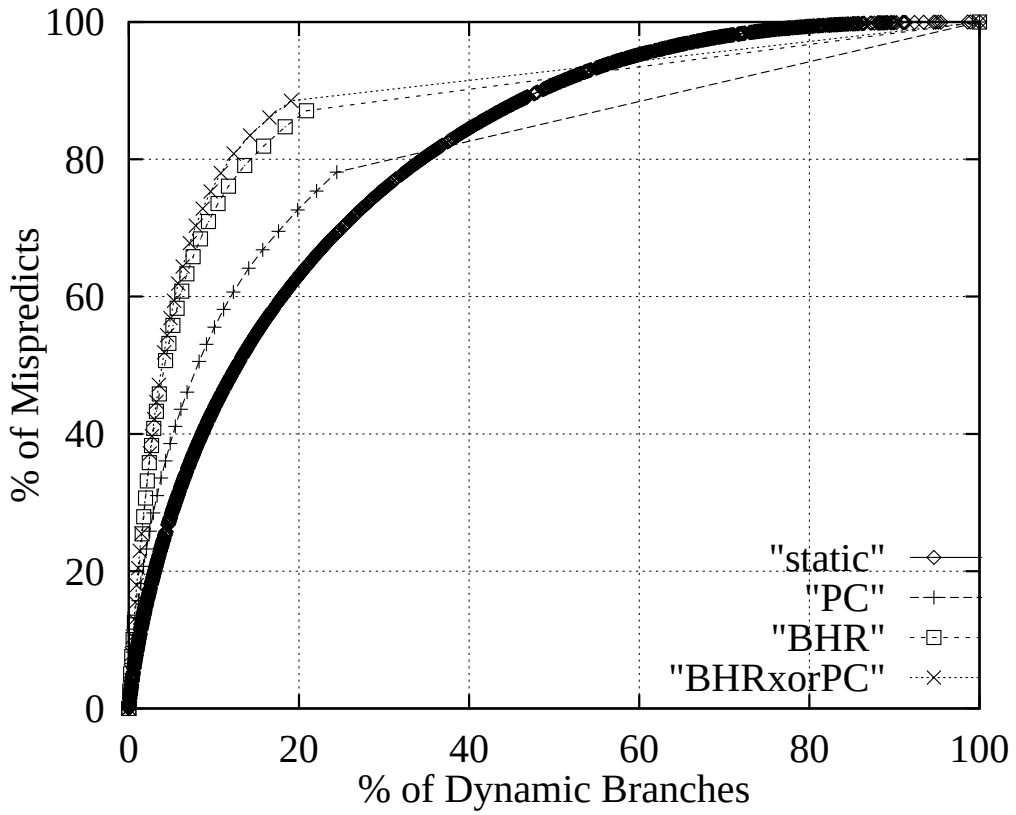
- Branch confidence predictor



- Table of resetting counters works well

Performance of Branch Confidence Mech.

- Concentrate miss-predictions into a small set



- Dynamic confidence is important
- Confidence can also be added to other predictions
  e.g. addresses or values

Dual Path Execution

- 100% Branch prediction will not be achieved
- Use confidence measure to determine branch predictions likely to be wrong
- Execute *both* paths and discard one
- Potential problems:
  - mispredictions tend to cluster
  - instruction fetch bandwidth goes up

Memory System Issues

- Lots of room for improvement...
- Consider Burger et. al. ISCA 96

  Minimal Traffic Cache

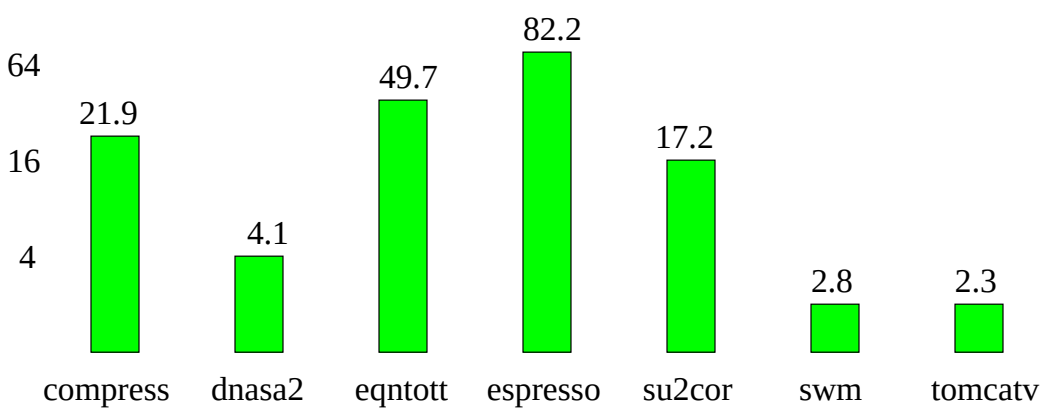
  full assoc.

  transfer size = request size

  optimal replacement policy

  low-priority loads bypass cache

- "Excess" traffic 2 to 82 times for 16KB cache



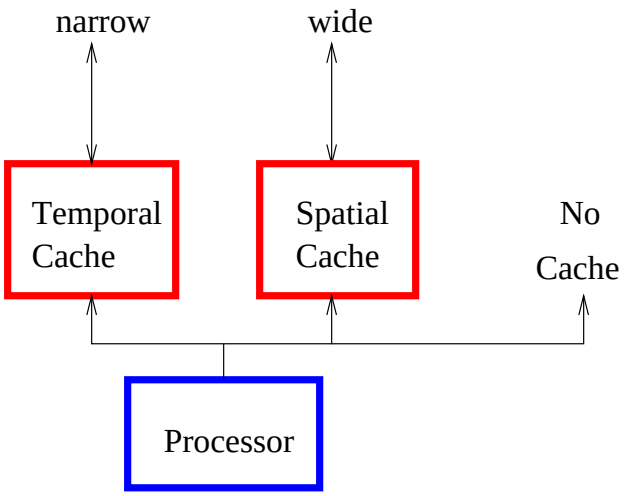
Memory System Approaches

- Smart prefetching

  related to address prediction

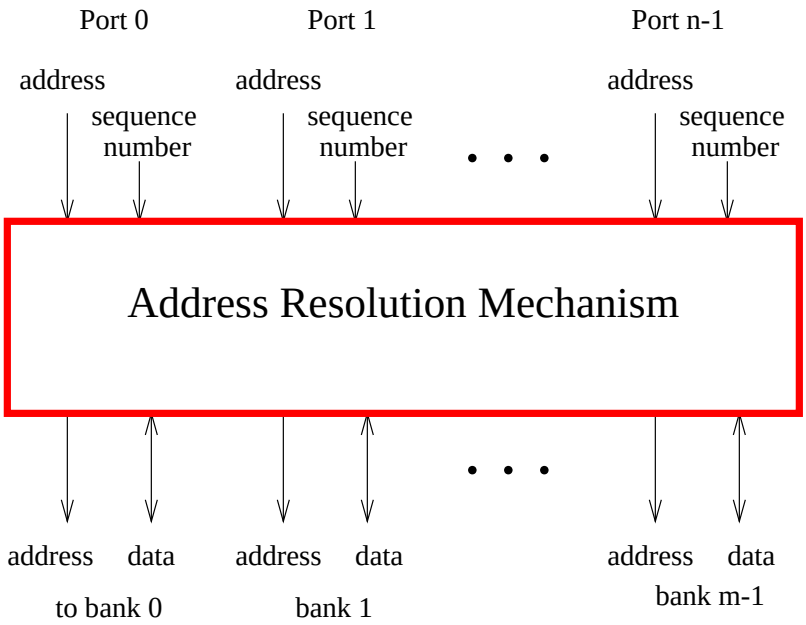
- Specialized Caches

  <Gonzalez et. al., ICS '95>



Advanced Memory Disambiguation

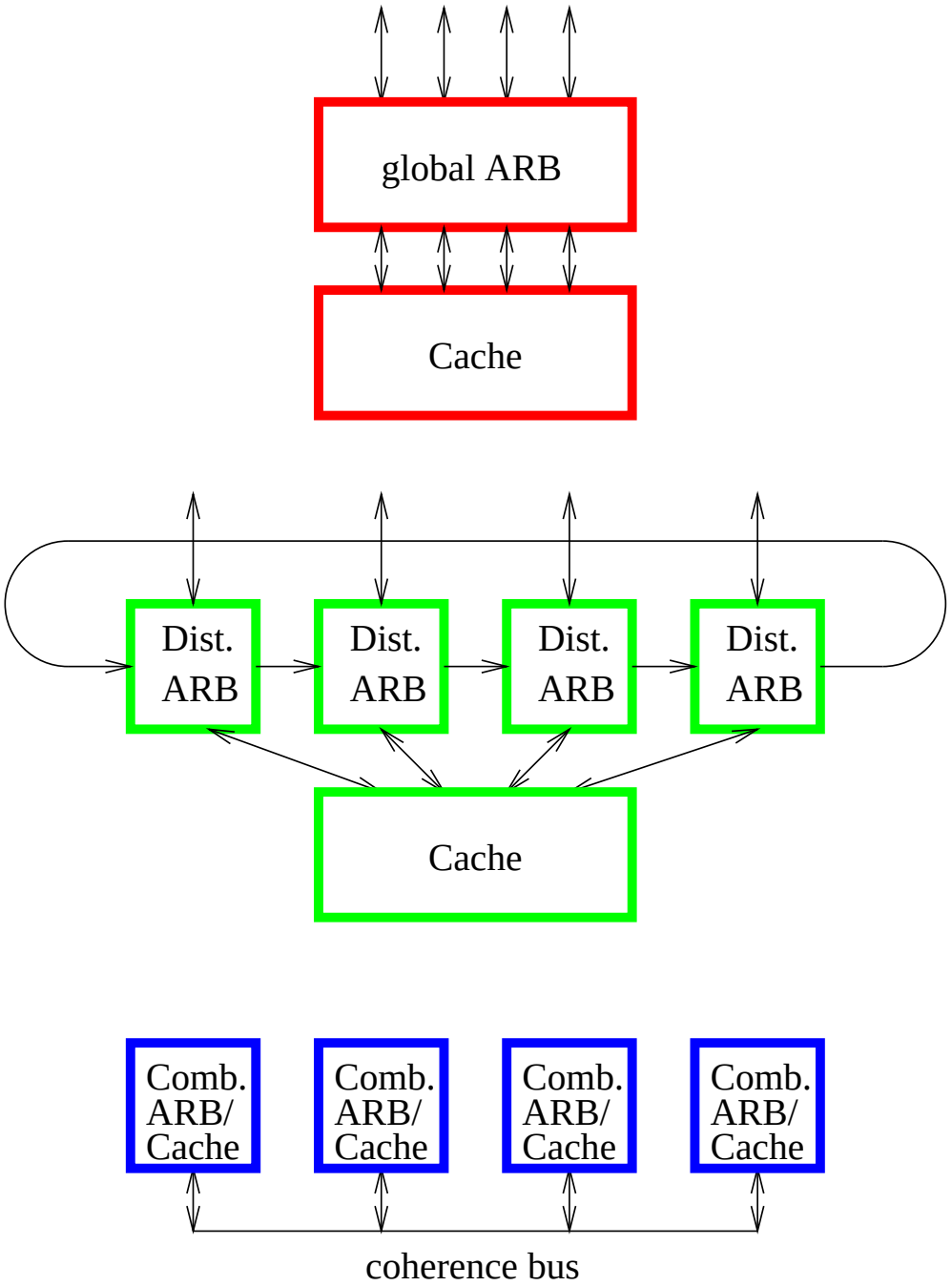
- An address resolution mechanism compares and buffers addresses from different processing units.



<Franklin and Sohi "Address Resolution Buffer">

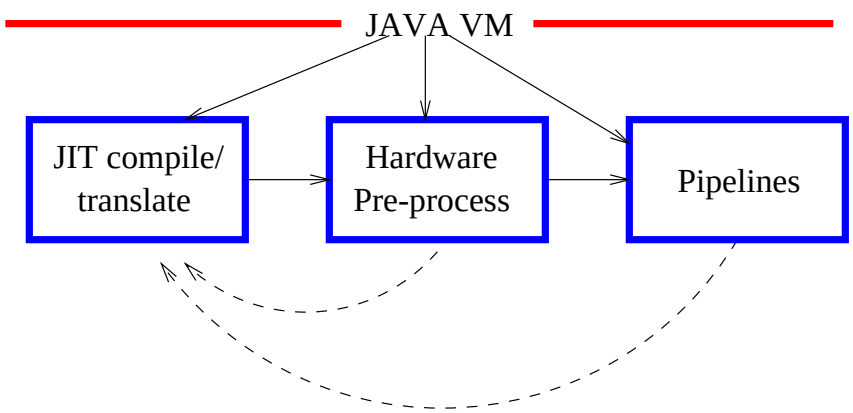
- Loads and stores execute speculatively
- Buffer speculative store data
- Recovery and state update must be efficient

ARB Alternatives



Instruction Sets

- The Java thing
  (or substitute x86... or IA-64)



- JIT Compile/Translation and/or Pre-processing will likely make traditional ISAs irrelevant

  (just as assembly language is largely irrelevant to programmers today)

- HOWEVER, there is now an internal instruction set, where designers have complete freedom (like microcode)

Compiler Implications

- Second and Third Generations:
  - Increase independence
  - Avoid register hazards
- Fourth Generation:
  - Improve communication
  - Increase register locality

  => put dependent instructions close together

  => register hazards increase register locality

  Stack instruction sets ??!!
- Compiler can focus on higher level parallelism
- Should take speculation into account

Conclusions

- Need structures that conform to technology
  - hierarchy
  - replication (incl. RAM-structures)
  - local communication
  - distributed control
- Must enhance ILP
  - wide instruction window
  - control dependences
  - data dependences
  - ambiguous memory references
- Research opportunities abound
  - focus on new generation microarchitecture
  - rather than enhancing superscalar

the last word (Amdahl, 1967)

"At any point in time it is difficult to foresee how the previous bottlenecks in a sequential computer will be effectively overcome.

If it were easy they would not have been left as bottlenecks.

It is true by historical example that the successive obstacles have been hurdled, so it is appropriate to quote the Rev. Adam Clayton Powell -- "Keep the faith, baby!"

Acknowledgements

- The people:

Guri Sohi

Selim Bilgin	Scott Breach
Andy Glew	Sridhar Gopal
Tim Heil	Quinn Jacobson
Anand Kamannavar	Matt Kupperman
Matt Mergener	Andreas Moshovos
Subbarao Palacharla	Eric Rotenberg
Amir Roth	Subrumanya Sastry
Yanos Sazeides	Avinash Sodani
Paul Thayer	T. N. Vijaykumar
Jeremy Williamson	Craig Zilles

- The funds:

DARPA

NSF

IBM

Intel