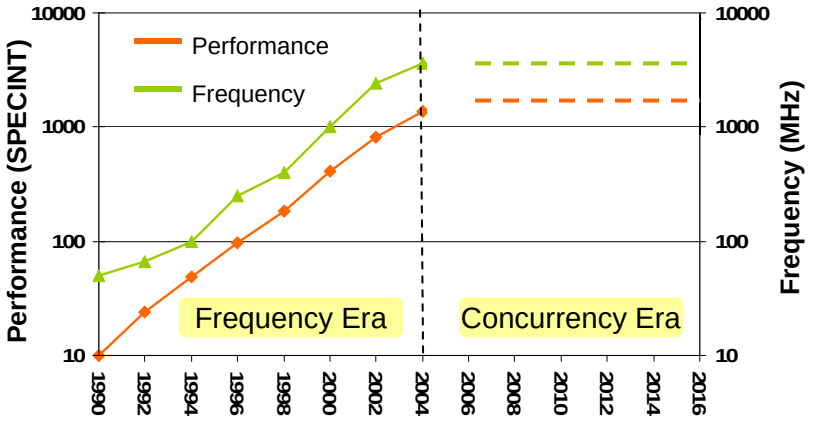
Increasing Concurrency using EDGE Architectures

Karu Sankaralingam

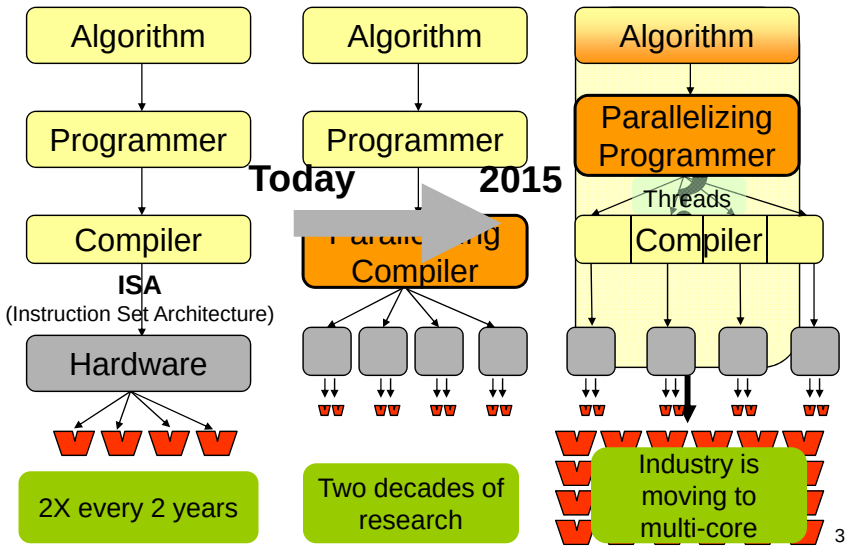
Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin
http://www.cs.utexas.edu/~karu

A Frequency to Concurrency Paradigm Shift



Performance growth must come from concurrency

Exploiting Concurrency - 10,000 ft View



3

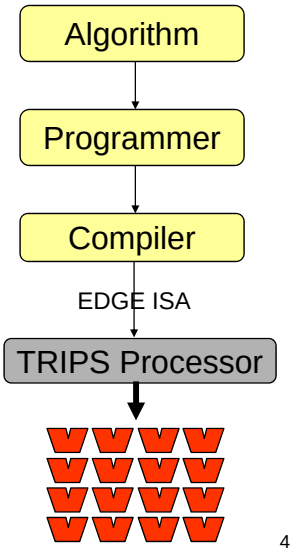
Concurrency in TRIPS - 10,000 ft View

Maintain conventional software design methodology

Renegotiate Compiler-H/W interface
Design ISA to match technology

Explicit Data Graph Execution
1. *Program is a sequence of blocks*
2. *Each block is a dataflow graph*

Concurrent hardware that can scale with technology



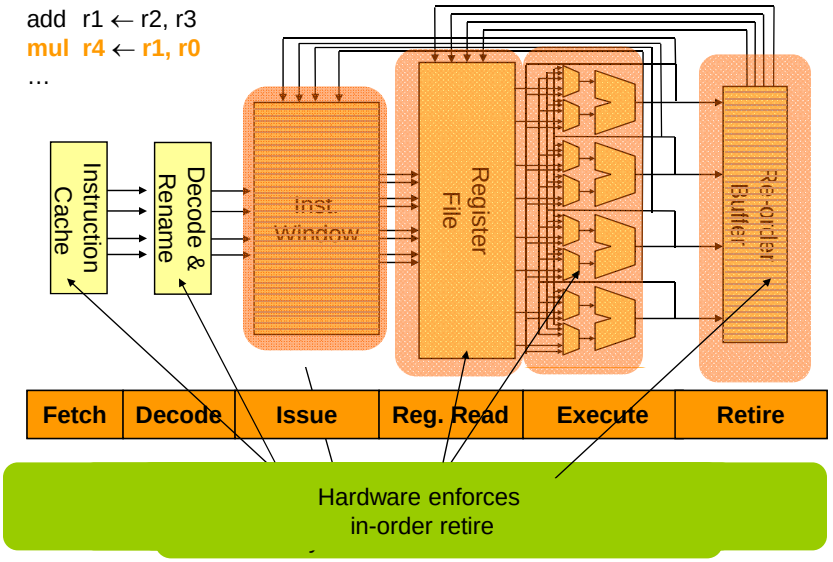
4

Outline

- Introduction
  - Scaling conventional microarchitectures
  - Summary
- Irregular concurrency
  - EDGE: A class of ISAs for concurrency
  - TRIPS: Concurrency in the microarchitecture
  - Performance results
  - TRIPS Prototype chip
- Extensions for regular concurrency
- Future work
- Conclusions

5

Why is Scaling Uniprocessors Hard?



6

Problem Statement

- How to design a concurrent architecture for sequential programming?

- Principles
  1. Rebalance compiler and hardware effort
  2. Amortize instruction-level overheads
  3. Eliminate centralized resources

7

Summary

- Principled approach
  - Programs encoded as blocks
    - *Amortize instruction-level overheads*
  - Each block is a dataflow graph
    - *Rebalance compiler and hardware effort*
  - Distributed microarchitecture built with mesh networks
    - *Eliminate centralized resources*

- Results:
  - Up to 5X better performance than best superscalar processors
  - Prototype system expected in summer

8

Outline

- Introduction
  - Scaling conventional microarchitectures
  - Summary of contributions
- Irregular Concurrency
  - EDGE: A class of ISAs for concurrency
  - TRIPS: Concurrency in the microarchitecture
  - Performance results
  - TRIPS Prototype chip
- Extensions for regular concurrency
- Future work
- Conclusions

9

Principle 1: Rebalance compiler and hardware effort
Principle 2: Amortize instruction-level overheads
Principle 3: Eliminate centralized resources

10

EDGE: A class of ISAs for concurrency

- **Explicit Data Graph Execution**
  - Defined by two key features

1. Block-atomic execution
   - Program graph is broken into sequences of *blocks*
   - Basic blocks, hyperblocks, or something else

2. Blocks encoded as dataflow graphs: Direct instruction communication
   - The block's dataflow graph is explicit in the architecture
   - Within a block, ISA support for direct producer-to-consumer communication
   - Across blocks, ISA support for named registers
   - Caveat: memory is still a shared namespace

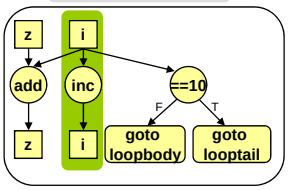
11

An Example

C-code

```
int main(void) {
  int z, i;
  z = 0;
  for (i = 1; i <= 10; i++) {
    z += i;
  }
  printf("Sum %d\n", z);
}
```

Dataflow graph



RISC ISA

```
$g1 ← 0
$g2 ← 1
loopbody:
add $g1 ← $g1, $g2
cmp $g0 ← $g2, 10
bz looptail
inc $g2 ← $g2
br loopbody
looptail:
…
```

Dataflow gr

EDGE ISA

```
.blockbegin init
    block's instructions
.blockend

.blockbegin loopbody
    block's instructions
.blockend

.blockbegin looptail
    block's instructions
.blockend
```

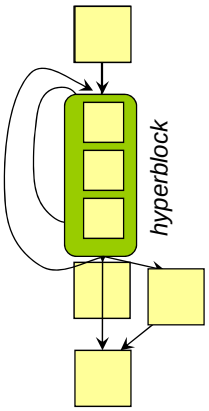
12

Role of the Compiler: Form Large Blocks

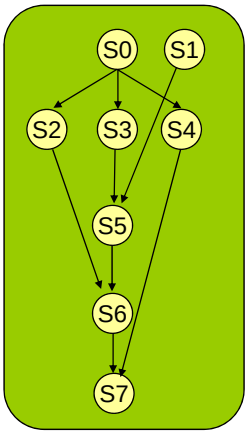
C Code

```
int main(void) {
  int z, i;
  z = 0;
  for (i = 1;
       i <= 10; i++) {
    z += i;
  }
  printf("%d\n", z);
}
```

Control flow Graph



Dataflow Graph

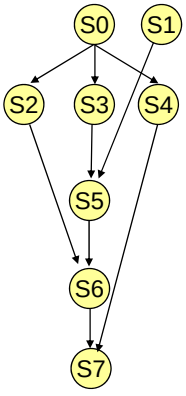


Control flow heuristics – loop unrolling, inlining, if-conversion…

13

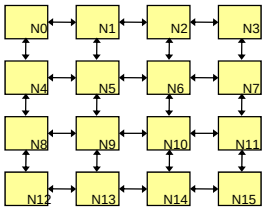
Role of the Compiler: Schedule Blocks

Dataflow graph

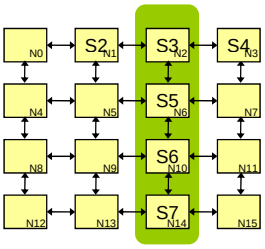


Scheduler

Processor topology model



Scheduled Dataflow graph



Map dependence graph paths to physical paths on execution substrate

14

Principle 1: Rebalance compiler and hardware effort

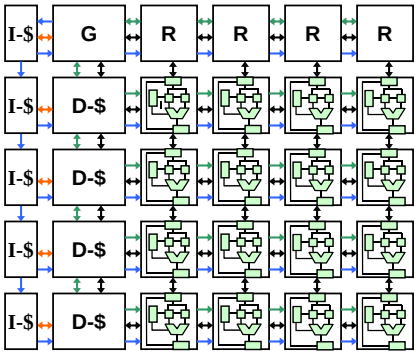
Principle 2: Amortize instruction-level overheads

Principle 3: Eliminate centralized resources

15

TRIPS Microarchitecture Principles

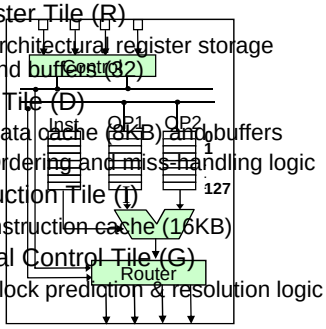
- Limit wire lengths
  - Architecture is partitioned and distributed
  - No centralized resources
  - Local wires are short
  - Networks connect only nearest neighbors
- Design for scalability
  - Design productivity by replicating tiles
  - Communication through well-defined control and data networks

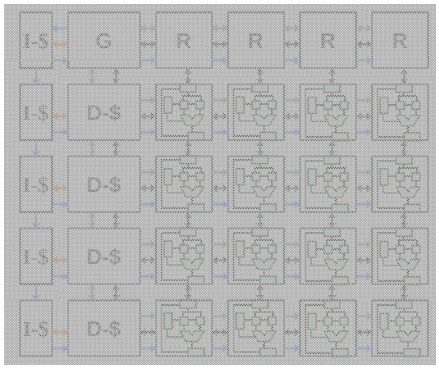


16

TRIPS Processor Organization

- Partition all major structures into banks, distribute, and interconnect
- Execution Tile (E)
  - Instruction and operand storage
- Register Tile (R)
  - Architectural register storage and buffers (32)
- Data Tile (D)
  - Data cache (8KB) and buffers
  - Ordering and miss handling logic
- Instruction Tile (I)
  - Instruction cache (16KB)
- Global Control Tile (G)
  - Block prediction & resolution logic

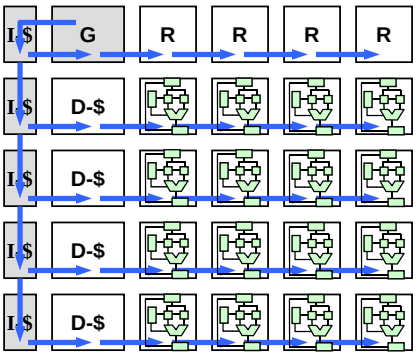




Communication Networks

17

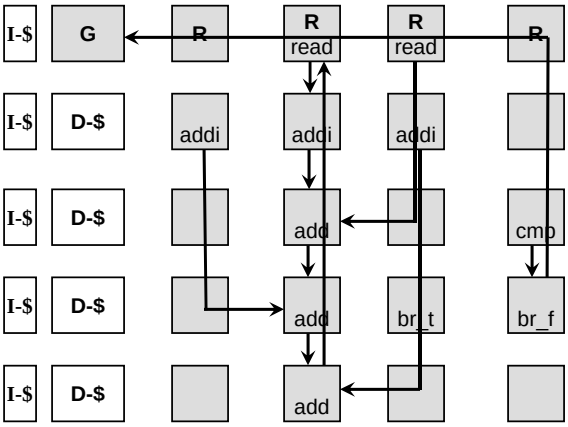
Execution Model (Fetch)



Instructions mapped to processor

18

Execution Model (Execution)

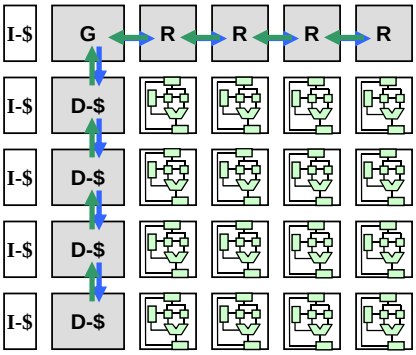


Instructions execute when operands arrive
Ordering determined dynamically
Operands routed through dynamic network

19

Execution Model (Commit)

Completion detected by data tile, register tile, and global tile



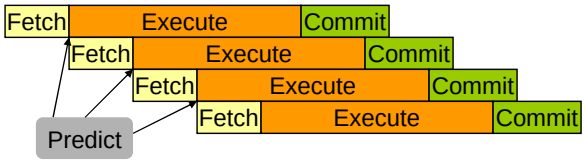
Resources can be deallocated

20

Inter-Block Concurrency

- Logically blocks execute sequentially

- Reservation stations and control speculation enables deep instruction window



21

Performance Results

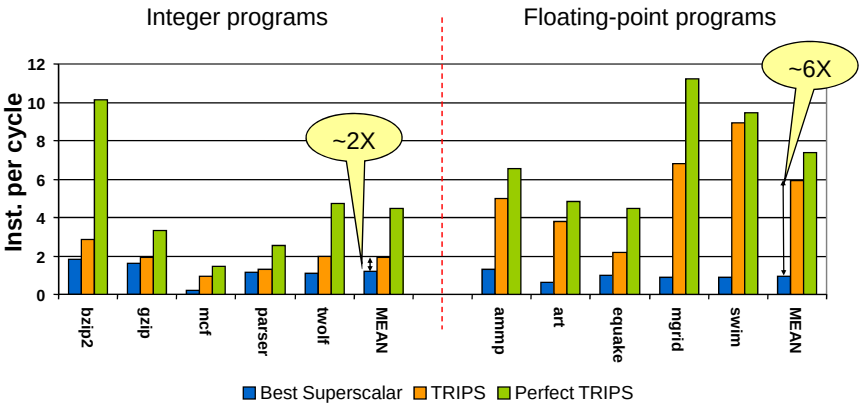
22

Performance Comparison

- What is a meaningful processor to compare against TRIPS?
  - Alpha 21264 is a balanced 4-wide machine
  - Goal of TRIPS is a balanced 16-wide machine
    - 1024 deep window, 4-port memory, 4-port register file
    - ISA exposes concurrency, exploits locality
- Experimental evaluation methodology
  - Sensitivity to design parameters
    - High level analysis: ALU-ALU delay, network topology, routing policies, *speculation depth*
  - Complexity implication of design decisions
    - H/W prototype answers some questions: OPN area, OPN routing latency, block predictor complexity, LSQ
  - Breakdown of overheads and sources of performance
- High quality compiled code essential
  - Compiler analysis and tuning ongoing
  - More progress needed to draw final conclusions

23

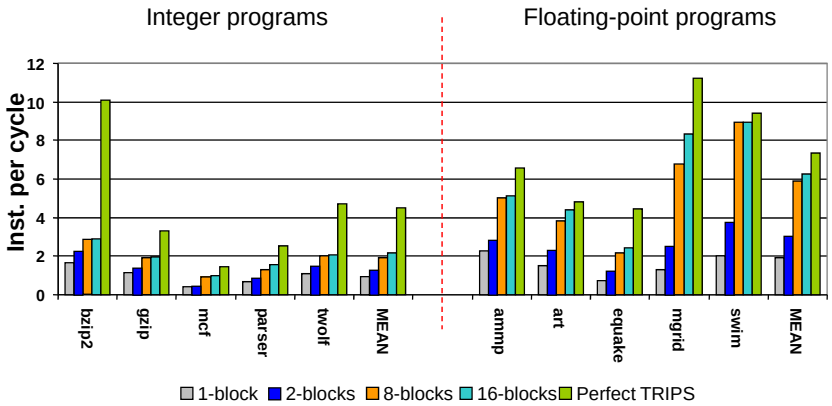
Comparison to Superscalar Processors



Both processors have similar L2, main-memory system

24

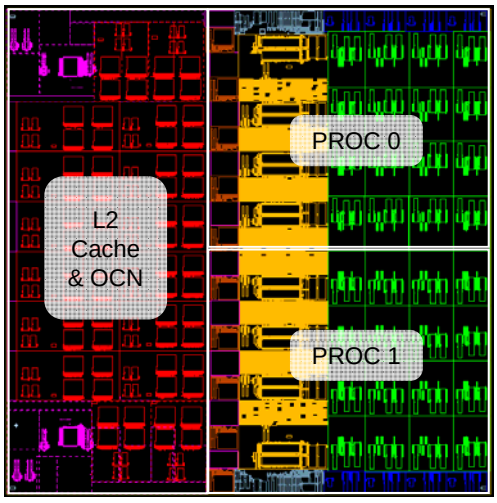
Sensitivity to Blocks in Flight



TRIPS allows 8 blocks in flight
High quality code and block predictor tuning will affect sensitivity

25

TRIPS Chip



130 nm 7LM IBM ASIC process
335 mm² die
~170 million transistors

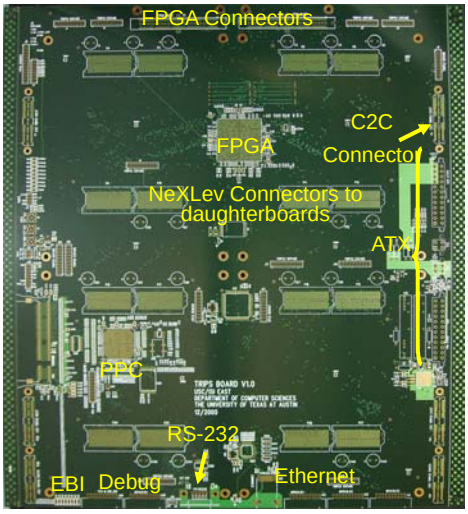
Overall Chip Area:
29% - Processor 0
29% - Processor 1
21% - Level 2 Cache
14% - On-Chip Network
7% - Other

Processor Area:
30% - Functional Units
4% - Register Files & Queues
10% - Level 1 Caches
13% - Instruction Queues
13% - Load & Store Queues
12% - Operand Network
2% - Branch Predictor
16% - Other

26

Motherboard Blank PCB

- Size 14” x 17”
- 18 layers
- Host
  - PowerPC 440GP (400 MHz, 3-way superscalar)
- Debug
  - FPGA XC2VP40 (1148 pins)
  - FPGA connectors for external I/O
- Four daughtercards each with 1 TRIPS chip



27

Prototype Design

- Design
  - Modularity reduced complexity: Specification → Physical design
  - SoC-like *but* tiles form one large uniprocessor
- Verification
  - Hierarchical verification (265 bugs total)
    - Tile-level, processor-level, chip-level
  - Performance verification (16 bugs total)
- Lessons
  - \+ Clean predicate model and simple block exit path
  - \+ Register renaming design revised, full search done once
  - \+ H/W prototype design helped push s/w toolchain flow
    - \+ Compiler heuristics, register allocator, scheduler
  - Block predictor design complexity ⇒ 3-cycles to predict
  - Significant router area (12%), routing logic on critical path
  - LSQ replication consumed significant area
    - Ongoing work addresses this challenge

28

My Contributions

- Processor architecture (MICRO 2001, ICCD 2003)
  - Dataflow execution
  - Explicit Data Graph Encoding ISA
  - Distributed microarchitecture
  - *NOT my work: block prediction, memory disambiguation, and TRIPS compiler*
- Prototype chip development
  - Processor ISA and microarchitecture specification
  - Many simulators, lines of verilog, and several tools
  - Processor verification lead and chip physical design lead
  - Spice → Netlist → Verilog → Microarchitecture
- Processor flexibility (ISCA 2003, MICRO 2003)
  - Exploiting different types and granularities of parallelism
  - Systematic identification of attributes to describe DLP programs
  - Mechanisms to match these attributes

29

Outline

- Introduction
  - Scaling conventional microarchitectures
  - Summary of contributions
- Irregular Concurrency
  - EDGE: A class of ISAs for concurrency
  - TRIPS: Concurrency in the microarchitecture
  - Performance results
  - TRIPS Prototype chip
- Extensions for regular concurrency
- Future work
- Conclusions

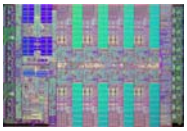
30

Irregular vs. Regular Concurrency

Hardware mines concurrency
Few ALUs (4-6)

Pentium 4

Programmer can find concurrency
Regular processor organization
Many ALUs (>32)



IBM Cell

NVIDIA G40
(graphics chip)

What are the underlying attributes of applications?
How to efficiently support regular concurrency in a dataflow substrate?

31

Regular Concurrency (Data Level Parallelism)

- A systematic analysis of DLP
  - Identify and characterize fundamental program behavior
  - Spanning microarchitecture mechanisms
  - **This talk: Constant reuse**

Scientific
Vector-add
FFT
LU-Decomposition

Network Proc.
MD5
AES encryption
Blowfish encryption

Multimedia
RGB-YIQ conversion
2D DCT
2D High-pass filter

Graphics (vertex)
Simple lighting
Reflection surface
Skinning

Graphics (pixel)
Simple lighting
Reflection surface
Anisotropic texture filter

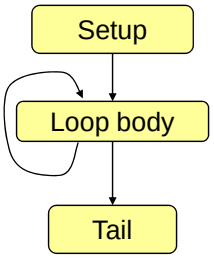
32

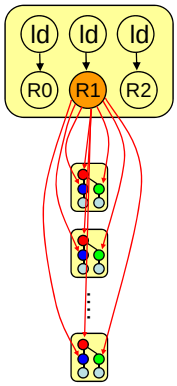
Constant Reuse

Pseudocode | Control flow graph | Dataflow graph

Setup phase:
Load constants from memory into registers
Loop body:
Read data to operate on
Perform computation
- High constant re-use

Write data to memory
Tail:
…

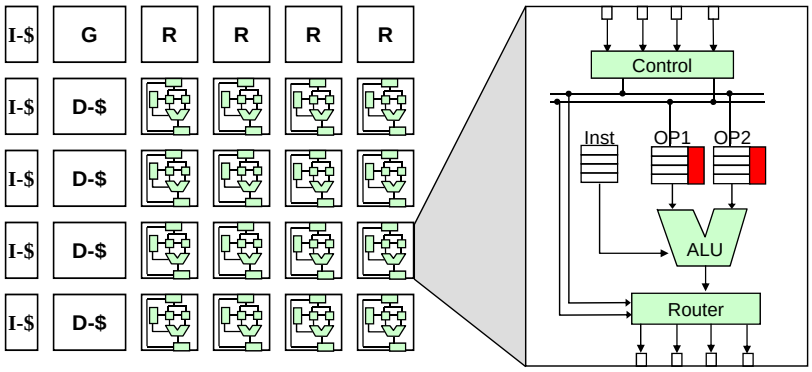




- High constant re-use results in high operand fanout
- Persistent local data storage eliminates operand fanout

33

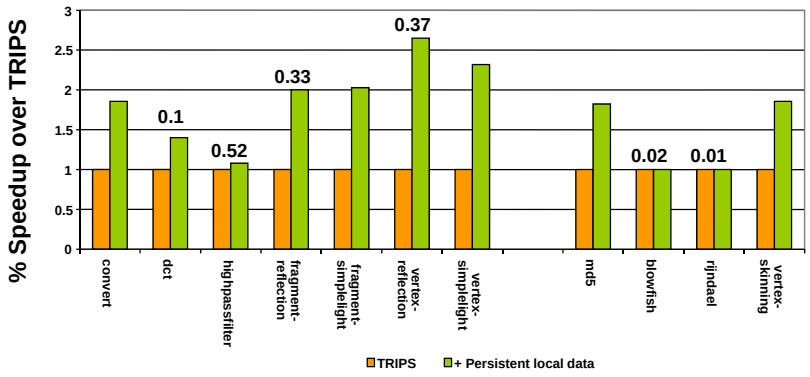
Persistent Local Data for Constant Reuse



Track reuse using status bits in reservation stations

34

DLP Results



~2.5X Speedup over TRIPS
Speedup correlated with constant fanout

35

Summary of Attributes and Mechanisms

- Memory
  - High constant reuse
    - *Persistent local data*
  - Regular and numerous memory accesses
    - *High bandwidth channels and multi-word load*
  - Frequently accessed small lookup tables
    - *Local per-tile scratchpad memories*
- Control
  - Regular control flow and many loop iterations
    - *Instruction revitalization*
  - Inner-loops with data-dependent branching
    - *Local per-tile control*
- Flexibility provides 55% better performance than a fixed configuration

36

Related Work

- Dataflow
  - Dataflow machines (Dennis 1974, U-Interpreter, Manchester 1985, Sigma-1 1987, TTDA/Monsoon 1990)
  - WaveScalar (2003), ASH (2004)
- Tiled Architectures
  - RAW(1997), Smart Memories(2000)
  - Cell (2001-2005), Niagara (2001-2005)
- Concurrency
  - Multiscalar (1995), ILDP (2002), Continual Flow pipelines (2004)
  - Transactional memory (1993)
  - Thread level speculation (1995)

37

Summary

- Three design principles
  - Rebalance compiler and hardware effort
  - Amortize instruction level overheads
  - Eliminate centralized resources
- EDGE: A new class of ISAs to match technology
  - Block atomic execution
  - Instruction-to-instruction communication, dependences explicit
- TRIPS microarchitecture
  - Modular and tiled design with mesh network
- Up to 5X better performance than superscalar processors

38

Future Work: Systems 10-15 years from now

- Energy efficiency and programmer productivity at high performance
- Technology
  - Much more unreliable
  - High process variability: each transistor is *unique*
- Software
  - Applications will change: mobility, modeling, ubiquitous connectivity
  - Concurrency will be expressed at language level also
    - Transactions, active objects, futures, parallel STL algorithms
  - Abundant latent concurrency in the application – 1000 “threads”
- Architecture:
  - Exploiting concurrency in “threads”: Where to run, when to run, how to run (voltage/frequency)?
  - What abstraction should compiler and language see?
  - Mechanisms and microarchitecture optimized for common case
  - *Agile at many levels*

39

Acknowledgements

Advisors

Stephen W. Keckler Doug Burger

Architecture Contributors

Raj Desikan
Saurabh Drolia
Sibi Govindan
Paul Gratz
Divya Gulati
Heather Hanson
Changkyu Kim
Haiming Liu
Robert McDonald
Ramdas Nagarajan
Simha Sethumadhavan
Premkishore Shivakumar
Nitya Ranganathan
Bill Yoder

40

41

"A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. Specialization is for insects."

- Lazarus Long

42

Backup Slides

43

Experimental Design Methodology

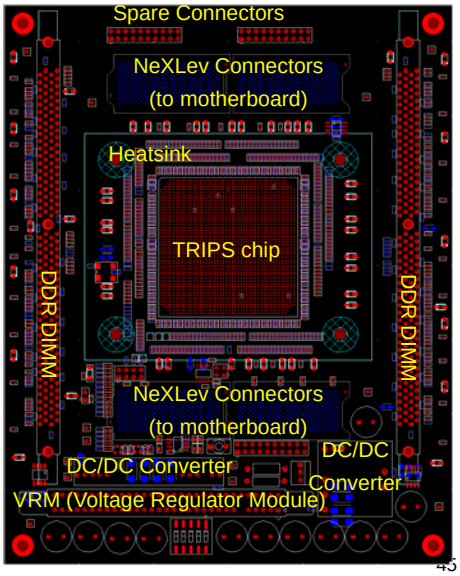
- Baseline balanced machine
  - Alpha 21264, 80 instructions in flight, 4 issue
- Target: Balanced 16 issue machine
- Challenges for conventional superscalar processor
  - $O(N^2)$ area growth in many structures
  - Scalable fetch bandwidth
  - Scalable memory bandwidth
- TRIPS: 16 issue, 1024 deep window, distributed organization
  - ILP through deep speculation and wide issue

- Sensitivity analysis (2001-2003)
  - Wire delay, ALU-ALU delay, network topology, *speculation depth*
- Microarchitecture tuning (2004)
  - Operand network routing algorithm, fairness policy, separated global networks, execution pipeline issue policies
- Processor microarchitecture critical path tool (2005-2006)
  - Quantify effects of “every” microarchitecture event

44

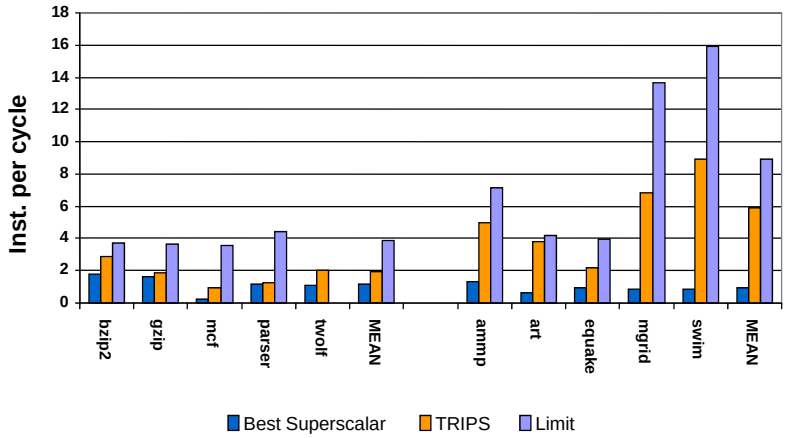
Daughterboard Components and Placement

- Size 6” x 7.5”
- 18 layers
- DRAM: up to 2 GB (2 x 1GB)
- Power: derived from 12V supply
- JTAG for TRIPS debugging



45

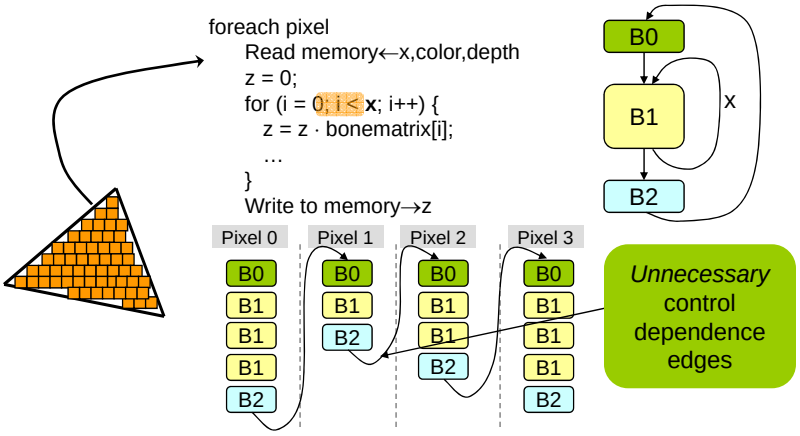
Results – Comparison to Limit IPC



Limit = 16-issue, 2048 entry instruction window, perfect prediction, 1-cycle memory

46

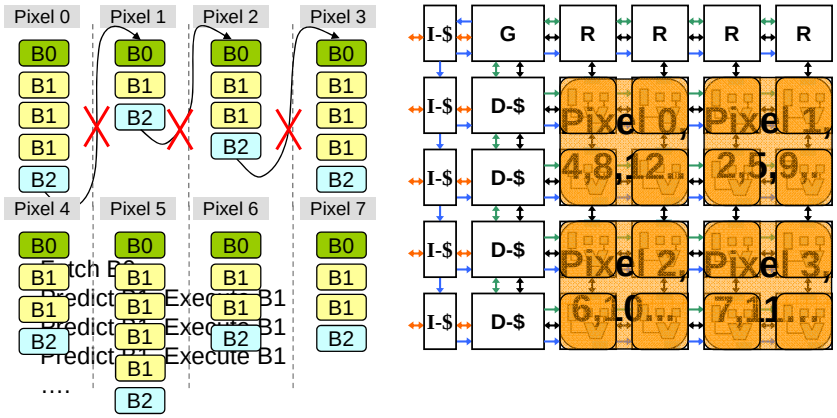
Data-dependent Branching



1. Concurrency within each pixel is small
2. Data-dependent branching causes mispredictions
3. Local control can expose intra-pixel concurrency

47

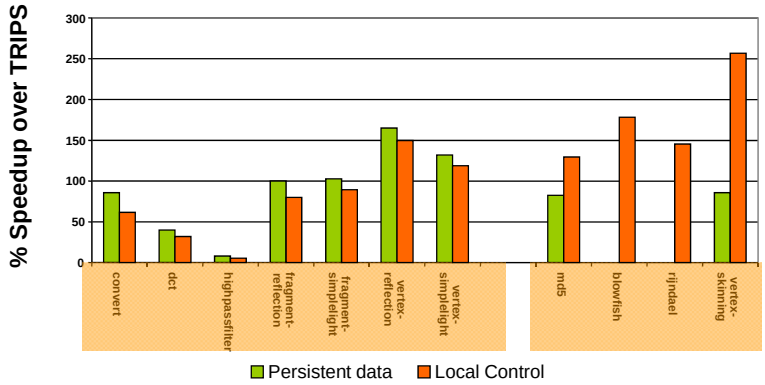
Local Control for Data-dependent Branching



Local control allows MIMD-like execution

48

DLP Results



Up to 250% improvement
Run-time processor flexibility

49

Dataflow History

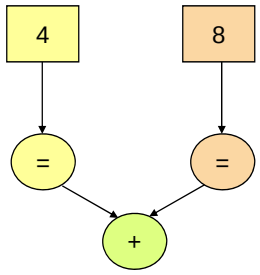
1. Petri nets (1962)
2. Estrin and Turn (1963)
   Karp and Miller (1966)
3. Tomasulo's Algorithm (1964)
4. Chamberlain (1971) – SSA dataflow language
5. Kahn (1974) – parallel language
   - Embedded systems: SDF(1986), Multi-dimensional dataflow, boolean dataflow, cyclostatic dataflow, Timed SDF, Heterochronous dataflow
6. Dennis (1974)
7. Arvind – TTDA

50

Dataflow Graph Definition

- Directed acyclic graph
- Each node is an “instruction” or operator in the program
- Control-flow (branch) represented by “merge” nodes
- Edges represent inputs for operators

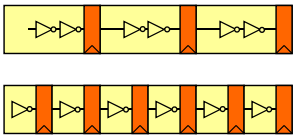
```
foo() {
  x = 4;
  y = 8;
  z = x + y;
}
```



51

A Frequency to Concurrency Paradigm Shift

- Deeper pipelines



Latch overhead is fixed ⇒ At best 6X more than Pentium4

- Faster transistors
  - Moore’s law ⇒ smaller transistors
  - Smaller translates to faster

Vd, and Vth does not scale ⇒ Only smaller, **not** faster

52

Overhead of Distributed Protocols

- Performance overhead (Nagarajan et al. ISPASS '06)
  - Fetch, Complete, Commit, Commit-ack: ~1% to 10% slowdown
  - Operand network: ~30% slowdown relative to oracle n/w
- Implementation overhead: Area, power, and design complexity
  - Fetch
    - Wires: ~140 bits between tiles
    - ~3500 latches
  - Complete, Commit, Commit-ack
    - Wires: ~30 bits between tiles
    - ~750 latches
  - Operand network
    - Wires: ~140 bits between tiles
    - **12% of processor area**
    - **Non-recurring one-time design complexity**
    - **High frequency design can be a challenge**

Dependent instructions always execute "back-to-back"

53

Processor Memory Gap

- Reduce latency
  - Prefetch: H/W or S/W hides latency as best as possible
  - Streaming model: Programmer fetches from memory before use
- Tolerate latency
  - Switch to another thread
  - Speculate on value and continue
  - Deep control speculation generates useful work early

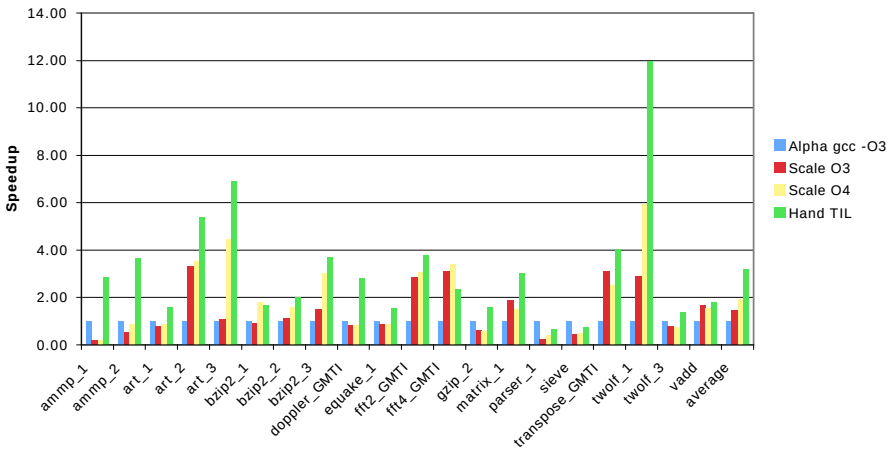
54

TRIPS Compiler – Block Sizes

	Compiler -O4		
Benchmarks	Min	Max	Mean
SPECINT2000	8.8	29.2	18.4
SPECFP2000	13.7	75.9	29.5
EEMBC	8.2	49.9	22.7
Microbenchmarks	16.5	101.3	48.6
	Hand optimized		
Microbenchmarks	17	112.5	73.5

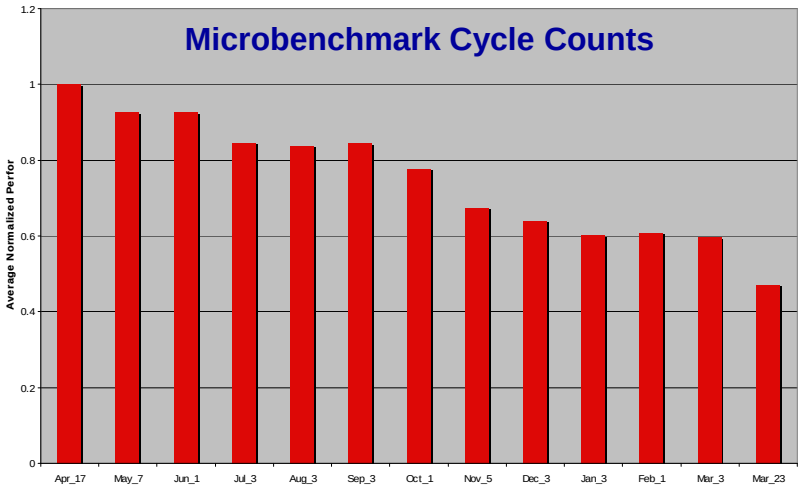
55

TRIPS Compiler - Performance



56

Compiler Progress April 2005 - March 2006



57

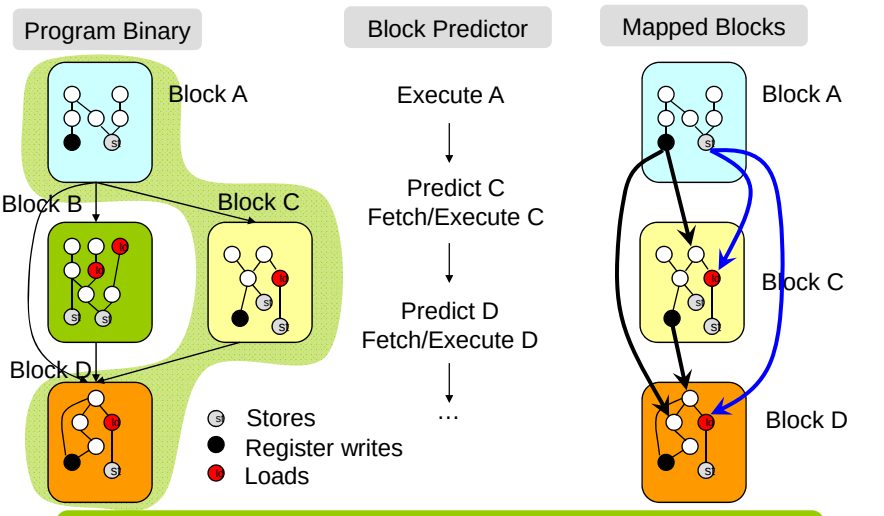
Processor Performance

Name	TRIPS Speedup	Alpha IPC	TRIPS IPC	TRIPS Inst/Block	Description
a2time	5.05	0.81	4.05	77	Control, integer math
bezier	3.30	1.05	3.20	76	Bezier curve, fixed-point math
dct8x8	2.66	1.70	4.70	90	2D discrete cosine transform
matrix	3.30	1.68	4.05	72	Matrix multiply
sha	0.92	2.28	2.10	80	Secure hash (mostly sequential algorithm)
vadd	1.92	3.04	6.51	74	Vector add (limited by load/store bandwidth)

Simulated on TRIPS and Alpha 21264 cycle simulators
Alpha compilation with GEM compiler and maximum opts (O4 and tuned for 21264)
TRIPS compilation with in-development compiler plus some hand-tuning
Speedup measured by comparing Alpha cycles to TRIPS cycles

58

Hardware Role in ILP

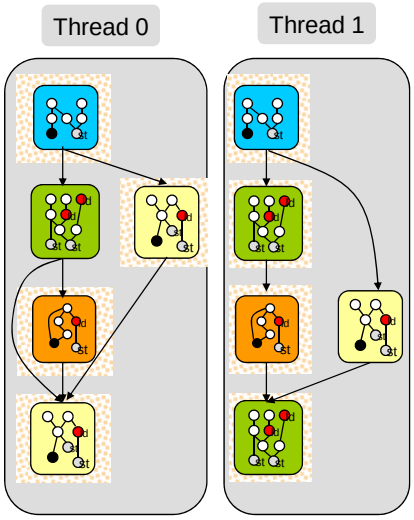


Hardware builds dynamic full program dataflow graph
by **register-renaming** and **enforcing load store dependences**

59

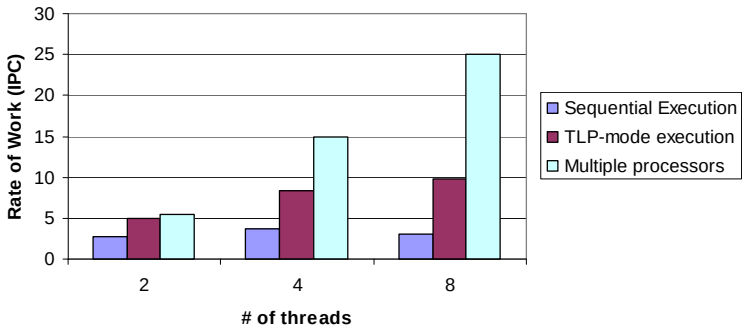
Thread-level Parallelism

- Partition instruction window, register files, L/S queues between threads
- Speculate within each thread
- Round-robin fetch and issue between threads



60

TLP Results



- Speedup: 1.8x to 2.9x
- Reasons for performance losses
  - Contention for resources (principally in instruction and data supply)
  - Reduced instruction window size

61

P4 Microarchitecture

Basic Pentium III Processor Misprediction Pipeline

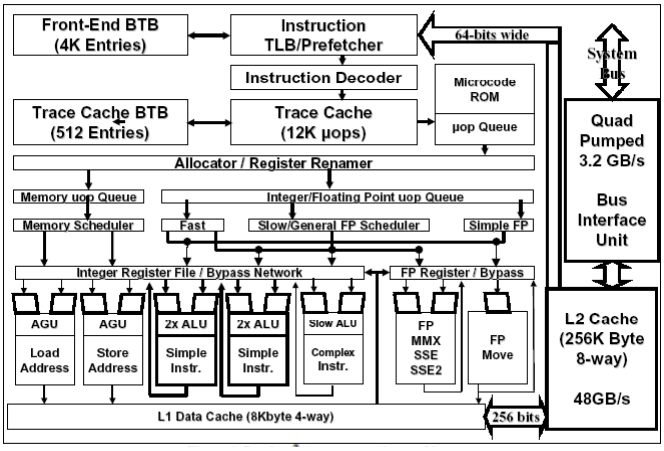
1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium 4 Processor Misprediction Pipeline

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP		TC Fetch		Drive	Alloc	Rename		Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive

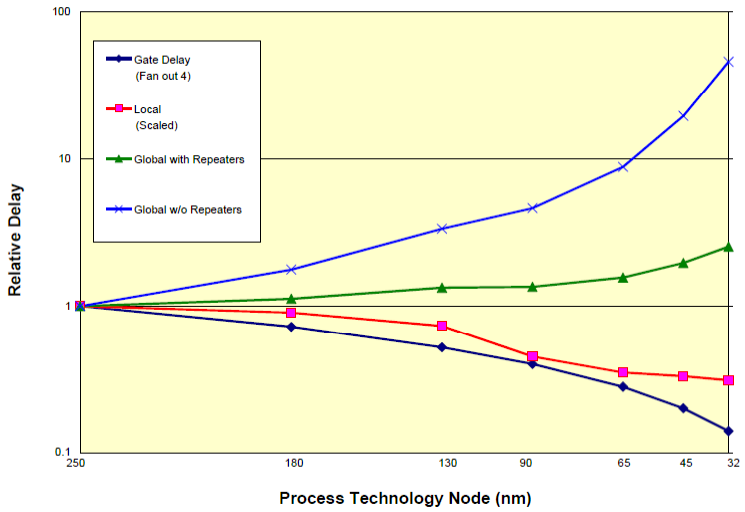
62

P4 Microarchitecture



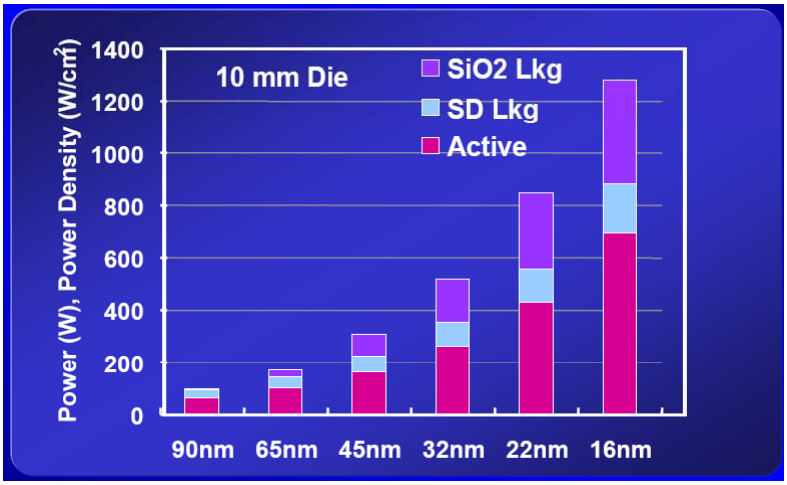
63

Wire Delays



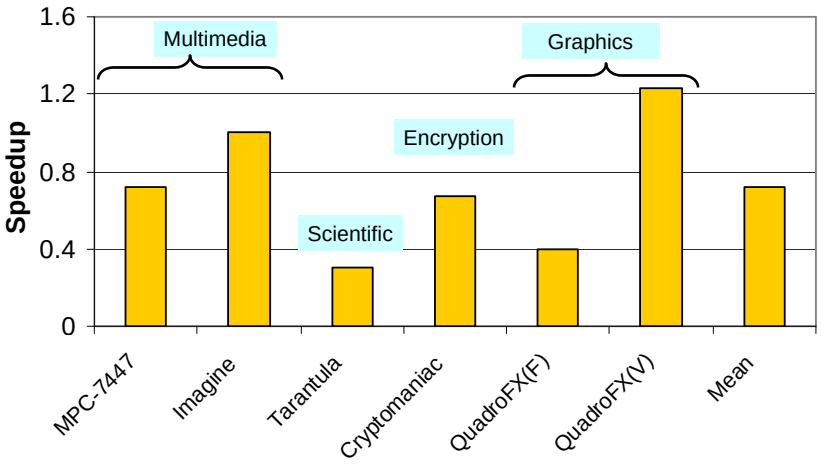
64

Power Scaling



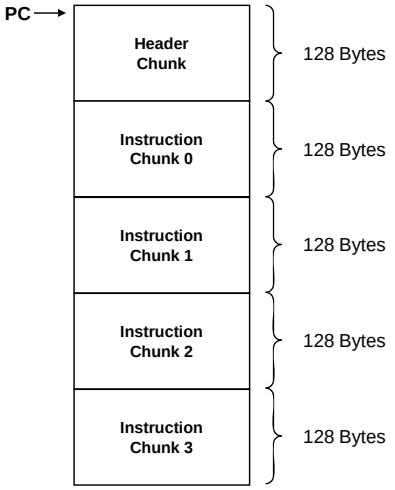
65

Comparison to Specialized Hardware



66

TRIPS Block Format



- Each block is formed five 128-byte program “chunks”
- The header chunk includes a block header (execution flags plus a store mask) and register read/write instructions
- Each instruction chunk holds 32 4-byte instructions (including NOPs)
- A maximally sized block contains 128 regular instructions, 32 read instructions, and 32 write instructions

67

Block Contents

```
R[0] read G[0] N[0] N[1]
R[1] read G[1] N[2]
…
R[31] read G[3] N[4]
```

- 32 Register reads

```
N[0]  add N[1]
N[1] mul w[0]
…
N[127] br
```

- 128 Instructions
- 32 Loads+Stores total
- 8 Branches

```
W[0] write G[64]
W[1] write G[65]
…
W[31] write G[127]
```

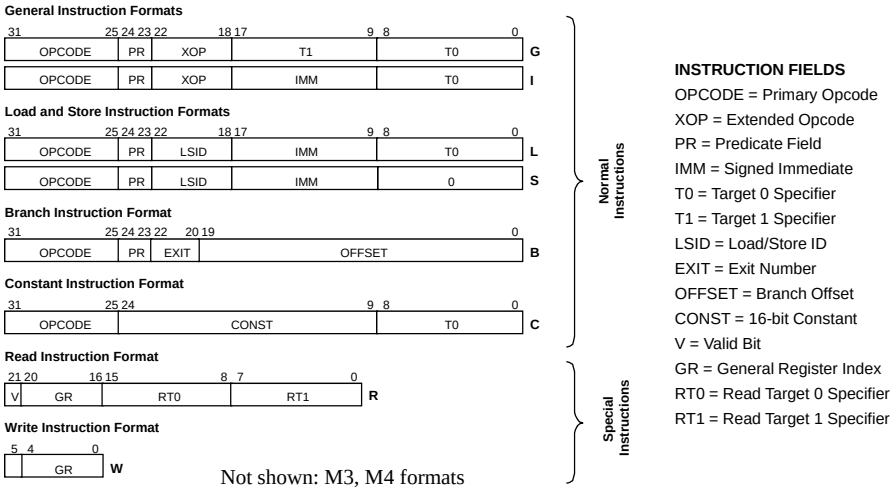
- 32 Writes

68

Key Features of TRIPS ISA

- Fixed instruction lengths - all instructions 32 bits
- Read and write instructions
  - Contained in block header for managing DFG/register file communication
- Target format `(T0, T1)`
  - Output destinations specified with 9-bit targets
- Predication `(PR)`
  - Nearly every instruction has a predicate field, encoded for efficient dataflow predication
- Load/store IDs `(LSID)`
  - Used to maintain sequential memory semantics despite EDGE ISA
- Exit bits `(EXIT)`
  - Used to improve block exit control prediction

TRIPS Instruction Formats

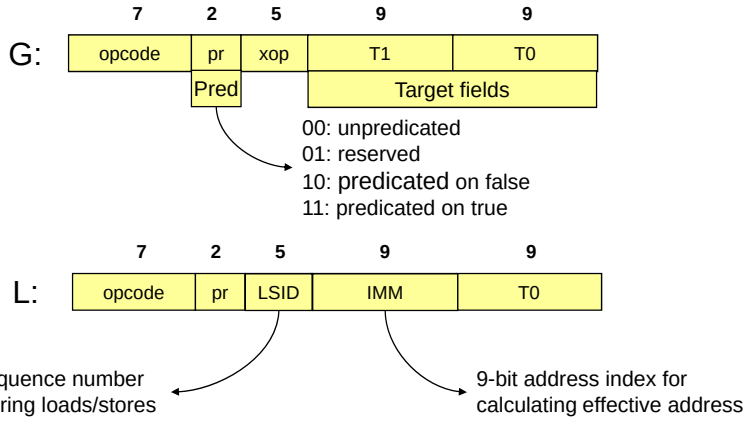


Not shown: M3, M4 formats

INSTRUCTION FIELDS

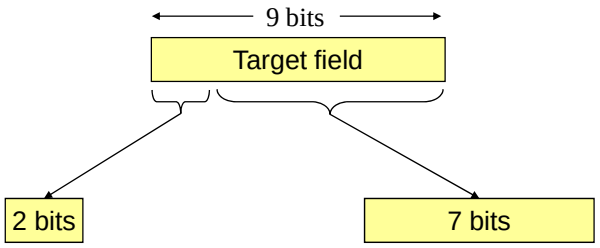
OPCODE = Primary Opcode
XOP = Extended Opcode
PR = Predicate Field
IMM = Signed Immediate
T0 = Target 0 Specifier
T1 = Target 1 Specifier
LSID = Load/Store ID
EXIT = Exit Number
OFFSET = Branch Offset
CONST = 16-bit Constant
V = Valid Bit
GR = General Register Index
RT0 = Read Target 0 Specifier
RT1 = Read Target 1 Specifier

TRIPS G and L formats



71

Target (Operand) Formats



- 00: No target (or write inst.)
- 01: Targets predicate field
- 10: Targets left operand
- 11: Targets right operand

- Names one of 128 target instructions
- Location of instruction is microarchitecture dependent

72

Object Code Example

TASL (target format)

```
[R1] $g1   [2]
[R2] $g2   [1] [4]
[1]  ld    L[1] 4 [2]
[2]  add   [3] [4]
[3]  mov   [5] [6]
[4]  st    S[2] 4
[5]  addi  2 [W1]
[6]  teqz  [7] [8]
[7]  b_t   block3
[8]  b_f   block2
[W1] $g5
```

Object Code

v	reg	T1	T0	
v	reg	T1	T0	
opcode	pr	LSID	imm	T0
opcode	pr	xop	T1	T0
opcode	pr	xop	T1	T0
opcode	pr	LSID	imm	0
opcode	pr	xop	imm	T0
opcode	pr	xop	T1	T0
opcode	pr	exit	offset	
opcode	pr	exit	offset	
v	reg			

73

Object Code Example

TASL (target format)

```
[R1] $g1   [2]
[R2] $g2   [1] [4]
[1]  ld    L[1] 4 [2]
[2]  add   [3] [4]
[3]  mov   [5] [6]
[4]  st    S[2] 4
[5]  addi  2 [W1]
[6]  teqz  [7] [8]
[7]  b_t   block3
[8]  b_f   block2
[W1] $g5
```

Object Code

1	00001	[2]	----	
1	00010	[1]	[4]	
ld	00	00001	4	[2]
add	00	---	[3]	[4]
mov	00	---	[5]	[6]
st	00	00010	4	---
addi	00	---	2	[W1]
teqz	00	---	[7]	[8]
b	11 (t)	E0	block3 - PC	
b	10 (f)	E1	block2 - PC	
1	00101			

Store mask: 00000000000000000000000000000100

74

Publications

- "Universal Mechanisms for Data-Parallel Architectures," *Micro 2003*
- “Routed Inter-ALU Networks for ILP Scalability and Performance," *ICCD 2003*
- "Distributed Pagerank for P2P Systems," *HPDC 2003*

75

End of Backup Slides

76