Specialization Is for Insects

Polymorphous Architectures: A Unified Approach for Extracting Concurrency of Different Granularities

Karu Sankaralingam

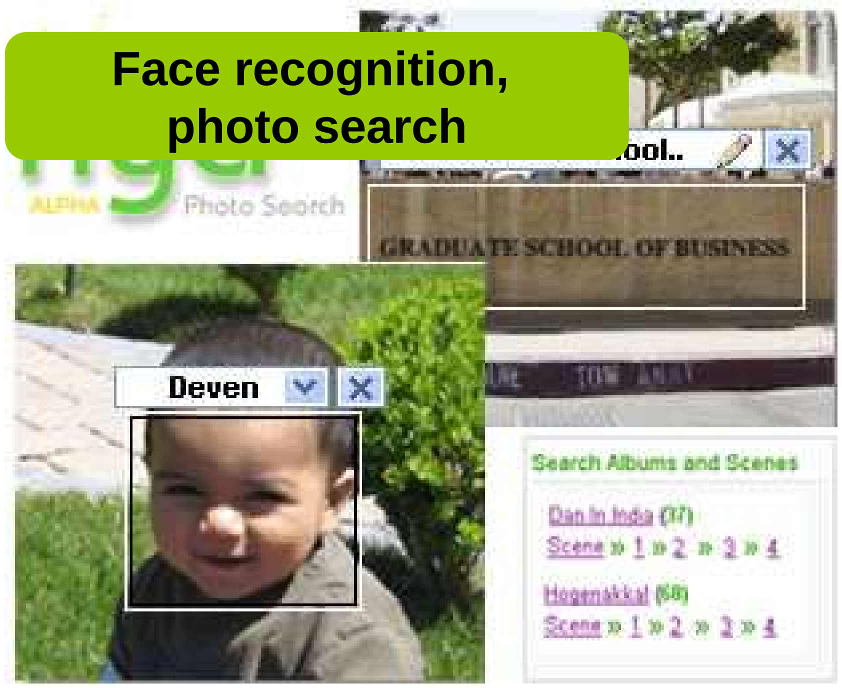
Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin
http://www.cs.utexas.edu/~karu

Technology Trends

- Wire delays
  - Less than 1% of chip reachable in a cycle
  - Architectures must be partitioned
- Power
  - Limits on pipelining reached
  - 12 to 22 FO4 seems optimal
- Processor complexity

Performance must come from concurrency

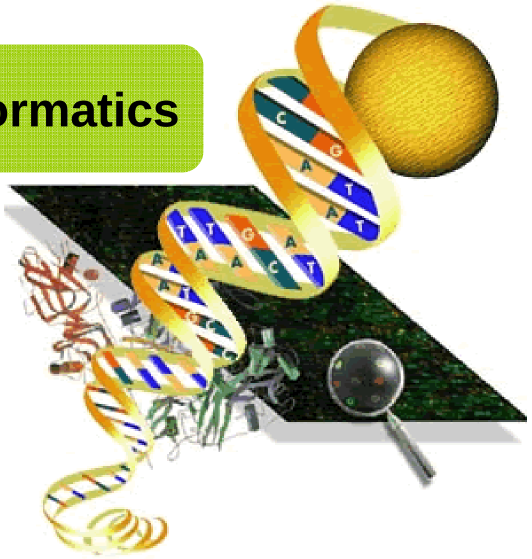
Application Heterogeneity



Face recognition, photo search

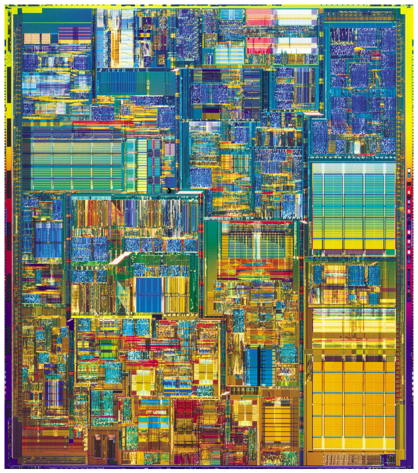
Video editing

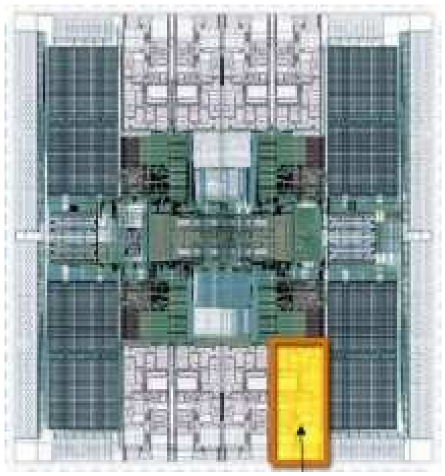
Game physics
Game graphics

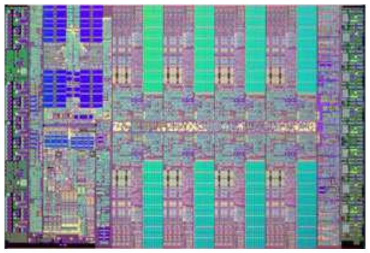


Bio-informatics

Conventional Microarchitectures


Intel Pentium 4


Sun Niagara

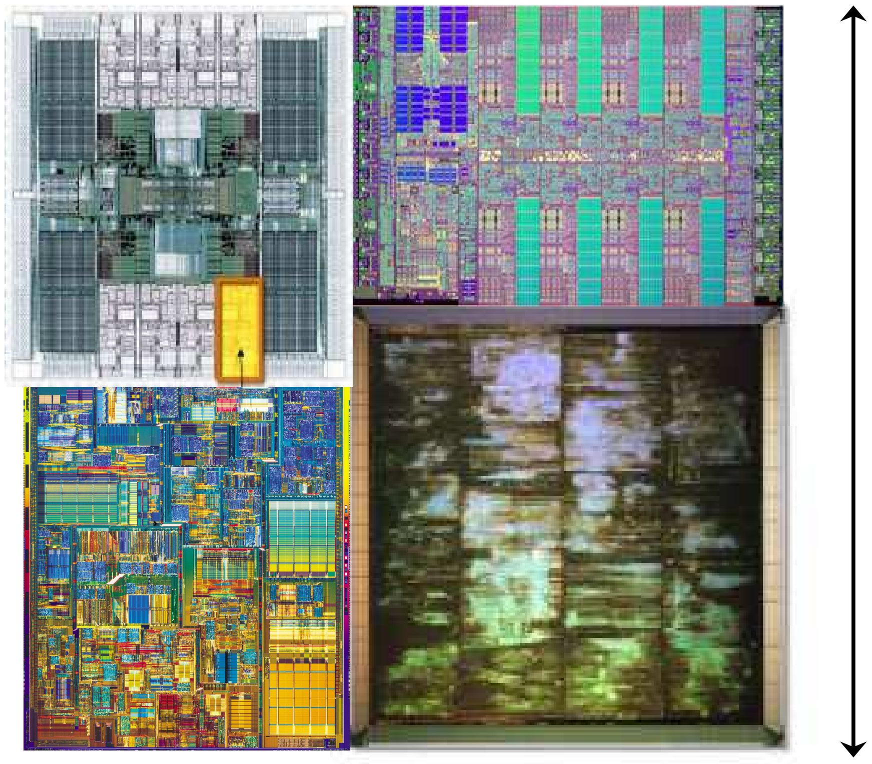

IBM Cell

NVIDIA G40 (graphics chip)

Desktop Server Games/Graphics

Tuned to one type of workload

Integrated Heterogeneity



Poor design reuse and complexity

Thesis Contributions

- Architectural polymorphism
  - Application controlled specialization
  - Coarse grain microarchitectural configuration
- Explicit Data Graph Execution ISA
  - Unifying abstraction layer for all types of concurrency
- Distributed microarchitecture design
  - Micronetworks and protocols
  - TRIPS prototype processor

Outline

- Completed in 2003
  - TRIPS architecture and high level microarchitecture design
  - Preliminary concept of polymorphism
  - Application characterization
- Promised in 2003
  - Detailed application characterization
  - Polymorphism mechanisms
  - TRIPS prototype processor

Outline

- Principles of Polymorphism
- EDGE Architectures and TRIPS prototype
- Instruction-level parallelism
- Thread-level parallelism
- Data-level parallelism
  - Application characterization
  - Mechanisms
  - Evaluation
- Conclusion

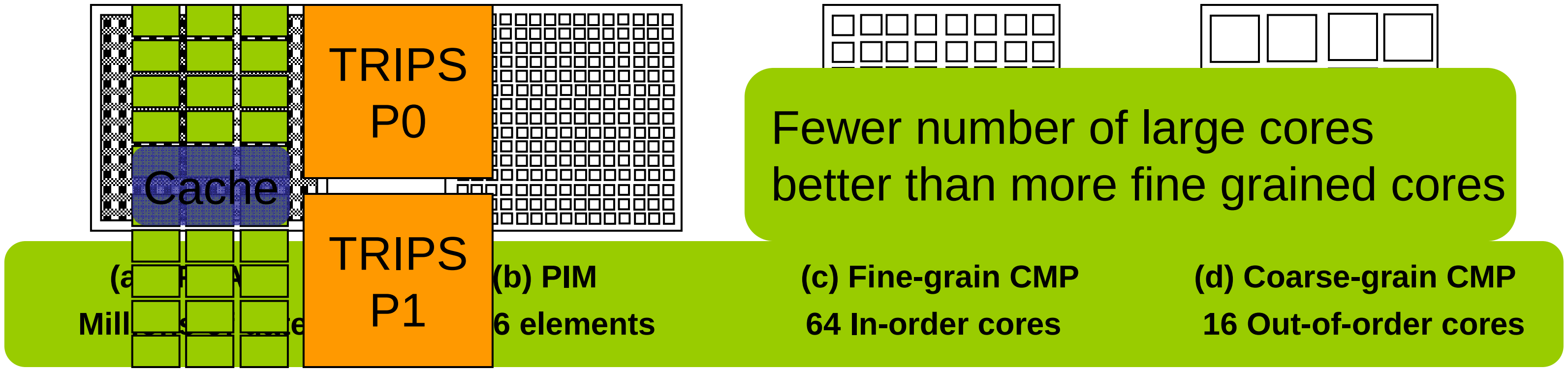
What is Architectural Polymorphism?

The ability to modify the functionality of coarse grain microarchitecture blocks at runtime, by changing control logic but leaving datapath and storage elements largely unmodified, to build a programmable architecture that can be specialized on an application-by-application basis.

- Principles:
  - Adaptivity to different granularities of parallelism
  - Economy of mechanisms
  - Reconfiguration of coarse grain blocks

System Design

- Granularity of processor core



- Granularity of parallelism
  - To first order differentiates application classes
  - Instruction-level parallelism (ILP)
  - Thread-level parallelism (TLP)
  - Data-level parallelism (DLP)
- Technology constraints
  - Modularity, reduced complexity, and energy efficiency

Taxonomy of Architecture Principles

Architecture type	Processing core type	Processor granularity	Configuration granularity
Programmable h/w	Homogeneous	Coarse-grain	Coarse-grain
App. specific h/w	Heterogeneous	Fine-grain	Fine-grain
Polymorphous Architectures			
Programmable	Homogeneous or Heterogeneous	Coarse or fine	Coarse grain
TRIPS and this Dissertation			
Programmable	Homogeneous	Coarse	Coarse
FPGA, Piperench, and ASH			
App. specific h/w	Homogeneous	Fine-grain	Fine grain
Tarantula			
Programmable	Heterogeneous	Coarse-grain	-

Outline

- Principles of Polymorphism
- EDGE Architectures and TRIPS prototype
- Instruction-level parallelism
- Thread-level parallelism
- Data-level parallelism
  - Application characterization
  - Mechanisms
  - Evaluation
- Conclusion

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

EDGE Architectures and Polymorphism

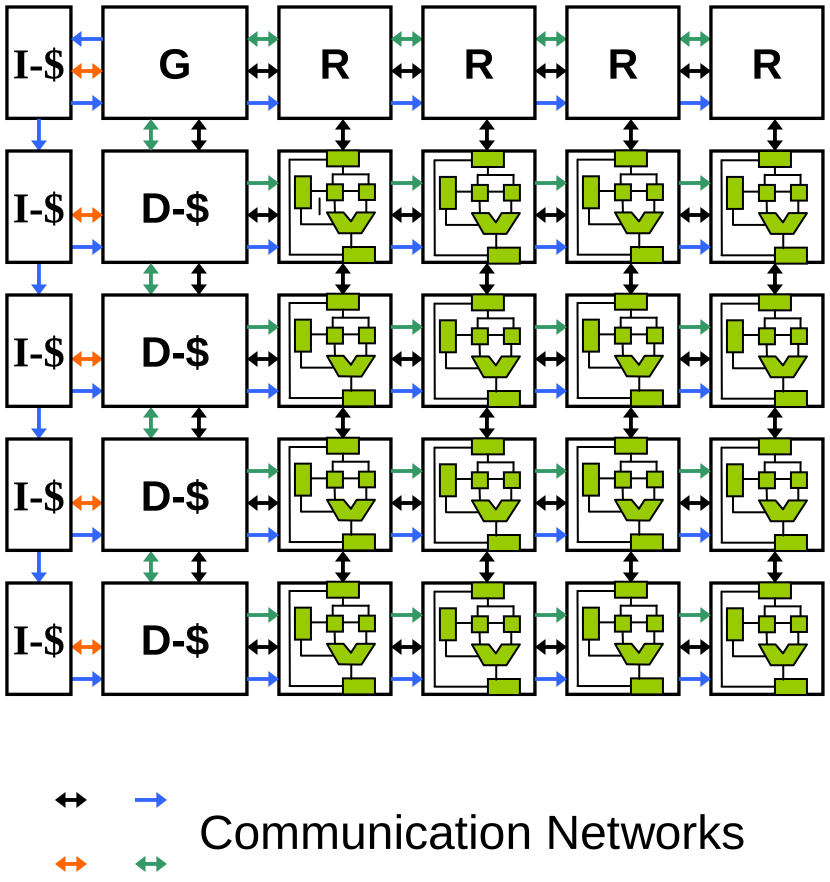
- The dataflow graph expresses concurrency efficiently
- ILP
  - Blocks express limited parallelism
  - Control speculation in h/w mines more
- TLP
  - Similar to ILP
- DLP
  - Ample parallelism is efficiently encoded
  - RISC: hardware rediscovers parallelism

C to TRIPS Binaries

- Control flow analysis creates hyperblocks
  - [Smith, CGO 2006] and [Maher, MICRO 2006]
- Scheduler assigns instructions to slots
  - ISA defines 128 slots
  - Scheduling is like a microarchitectural optimization
  - [Nagarajan, PACT 2005], and [Coons, ASPLOS 2006]
- Complete software toolchain
  - GNU binuntils based
  - TRIPS compiler builds EEMBC and SPEC CPU2000

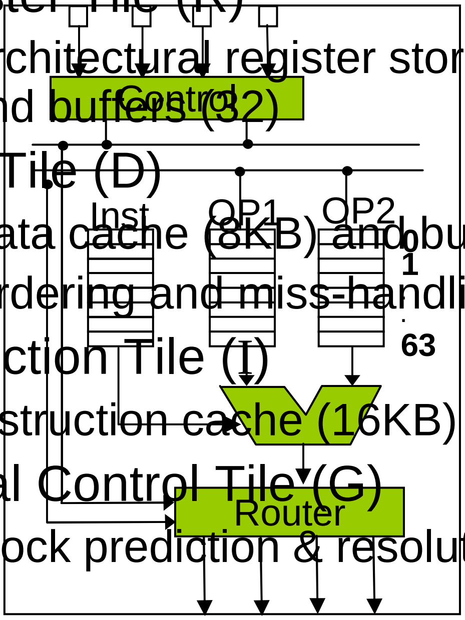
TRIPS Microarchitecture Principles

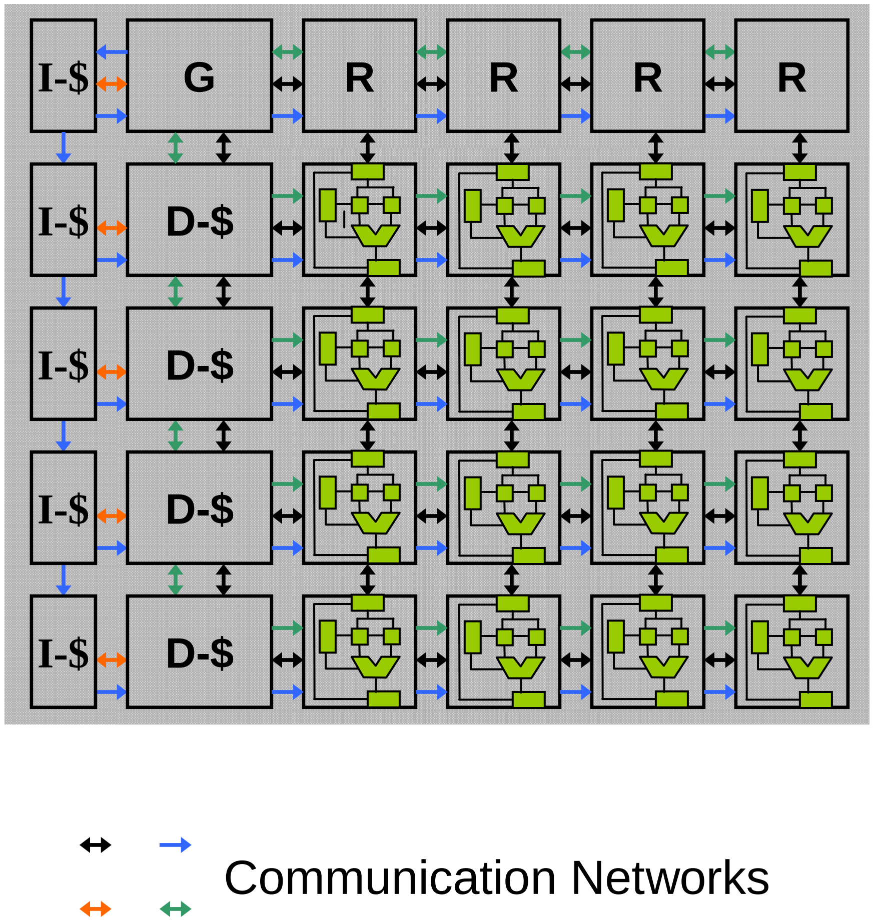
- Limit wire lengths
  - Architecture is partitioned and distributed
  - No centralized resources
  - Local wires are short
  - Networks connect only nearest neighbors
- Design for scalability
  - Design productivity by replicating tiles
  - Communication through well-defined control and data networks



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

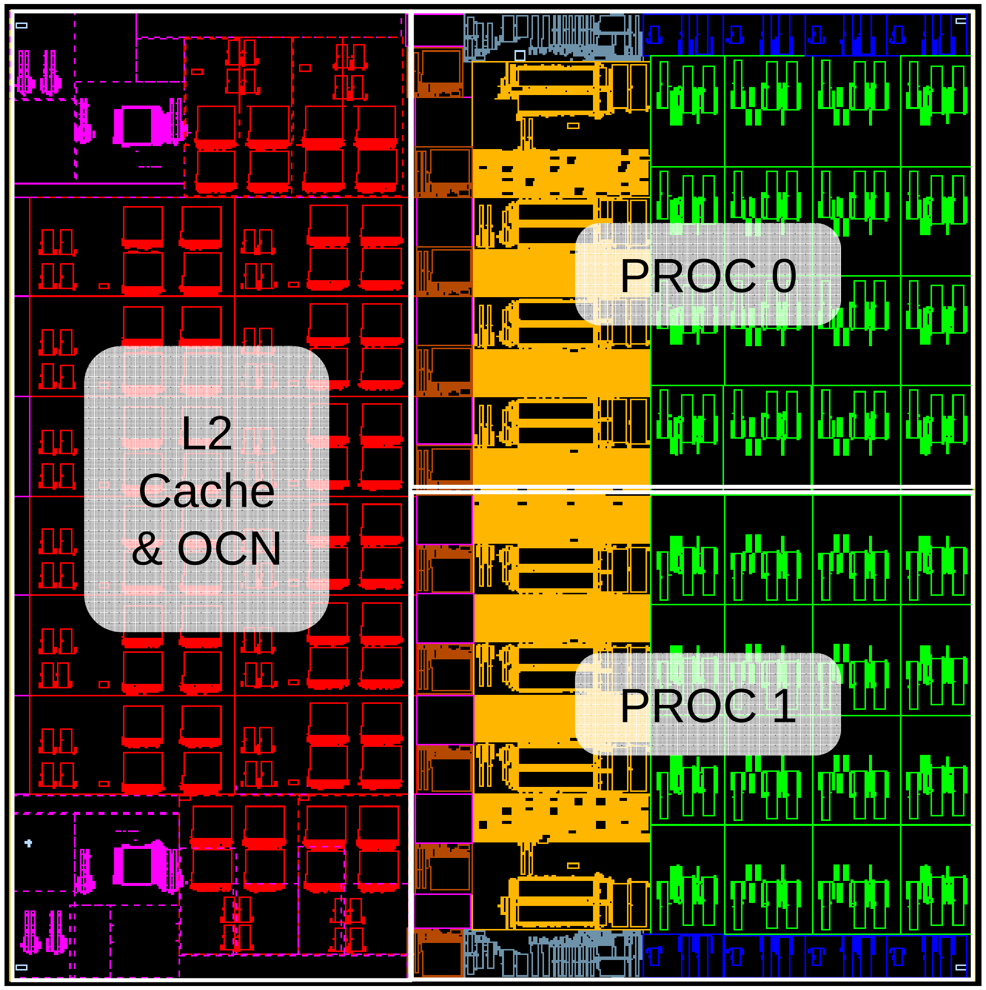




TRIPS Micronetworks and Protocols

Micronetwork	Function
Operand n/w: OPN	Pass operands
Global dispatch n/w :GDN	Dispatch instructions
Global status n/w: GSN	Block completion information
Global refill n/w: GRN	I-cache miss refills
Data status n/w: DSN	Store completion status
External store n/w: ESN	Store completion status in L2

TRIPS Chip



130 nm 7LM IBM ASIC process
335 mm^2 die
~170 million transistors

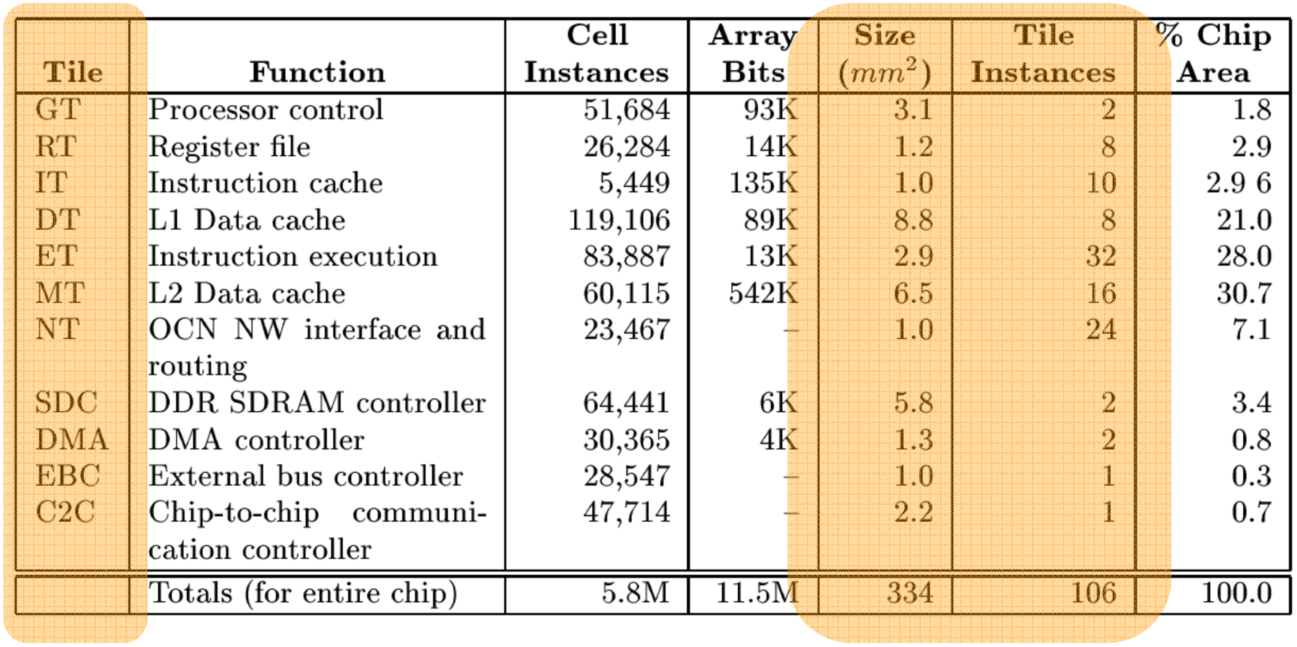
Overall Chip Area:
29% - Processor 0
29% - Processor 1
21% - Level 2 Cache
14% - On-Chip Network
7% - Other

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

Prototype Design

- Design
  - Modularity reduced complexity: Specification $\rightarrow$ 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)



Tile	Function	Cell Instances	Array Bits	Size (mm^2)	Tile Instances	% Chip Area
GT	Processor control	51,684	93K	3.1	2	1.8
RT	Register file	26,284	14K	1.2	8	2.9
IT	Instruction cache	5,449	135K	1.0	10	2.9 6
DT	L1 Data cache	119,106	89K	8.8	8	21.0
ET	Instruction execution	83,887	13K	2.9	32	28.0
MT	L2 Data cache	60,115	542K	6.5	16	30.7
NT	OCN NW interface and routing	23,467	–	1.0	24	7.1
SDC	DDR SDRAM controller	64,441	6K	5.8	2	3.4
DMA	DMA controller	30,365	4K	1.3	2	0.8
EBC	External bus controller	28,547	–	1.0	1	0.3
C2C	Chip-to-chip communication controller	47,714	–	2.2	1	0.7
	Totals (for entire chip)	5.8M	11.5M	334	106	100.0

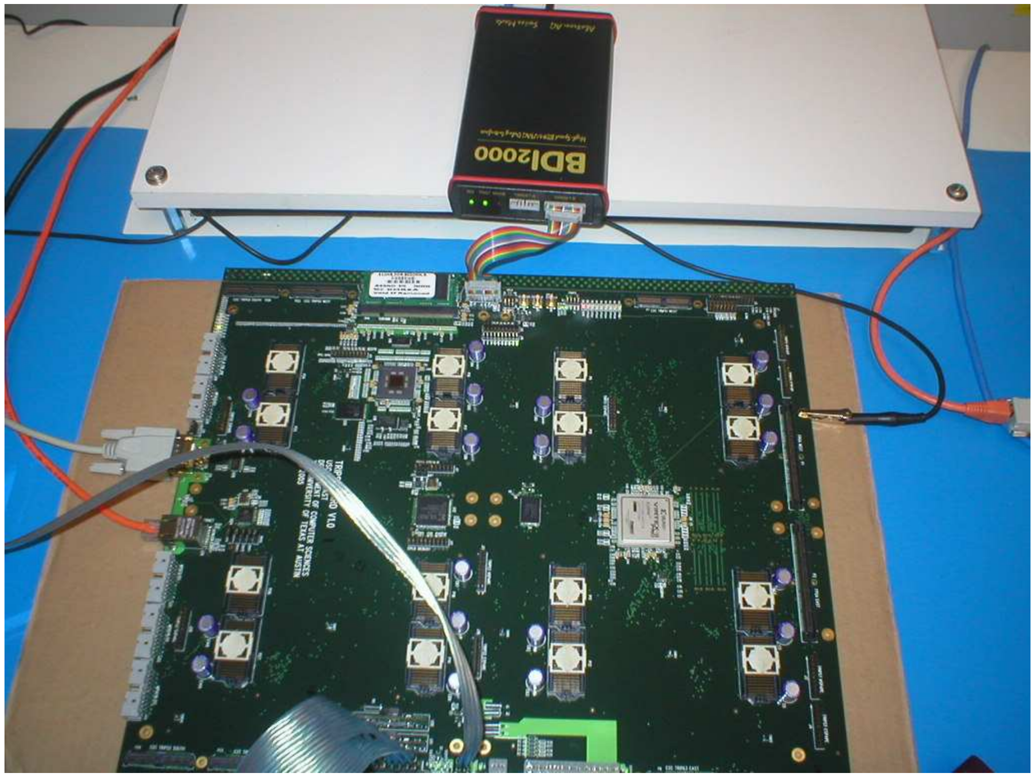
Prototype Design Lessons

- + 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

TRIPS Motherboard

- 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



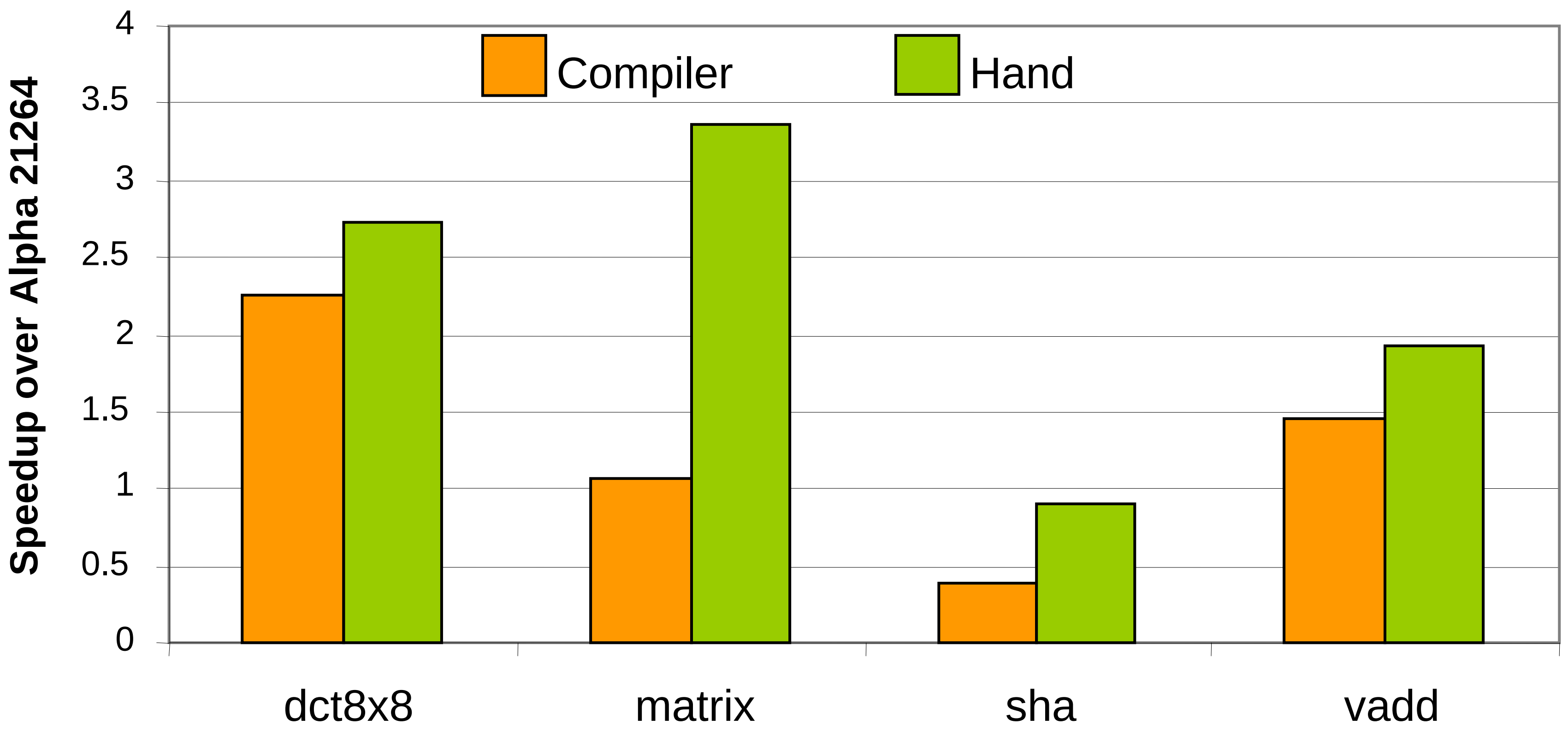
Outline

- Principles of Polymorphism
- EDGE Architectures and TRIPS prototype
- Instruction-level parallelism
- Thread-level parallelism
- Data-level parallelism
  - Application characterization
  - Mechanisms
  - Evaluation
- Conclusion

Instruction-Level Parallelism

- Control speculation exposes parallelism
- Register renaming and load/store pairs build program level DFG

ILP Results (Microbenchmarks)



Demonstrates potential
Can compiler generate high quality code?

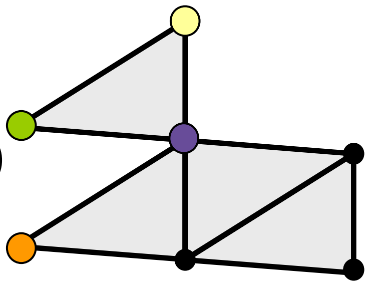
Thread-level Parallelism

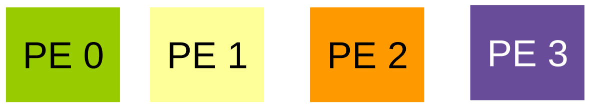
- Execution Tiles:
  - Reservation stations divided between threads
- Register Tiles:
  - Register renaming augmented
  - Extra physical register storage for each thread
- Global Tile:
  - Instruction fetch cycles between threads
  - Small amount of block predictor storage added
- Results:
  - High processor utilization: average IPC of 3.0
  - 2X speedup when executing 4 threads
  - Inter-thread contention in general low: ~20%
  - But dominates for highly concurrent programs

Data-level Parallelism

- Many common attributes:
  - High computation intensity and memory b/w
  - Loops executing on parts of memory in parallel
- But,
  - Memory access patterns can vary
  - Loops sizes can vary
  - Control flow can vary

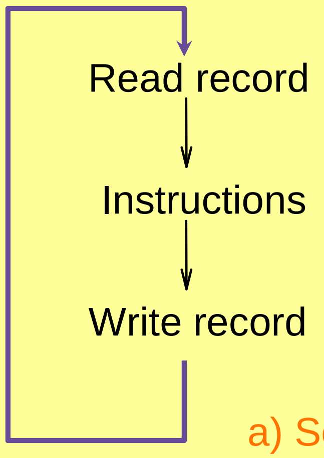
```
for each vertex V {
  for (j = 0; j < V.ntrans; j++) {
    Z = Z + product(V.xyz, M[j])
  }
}
```





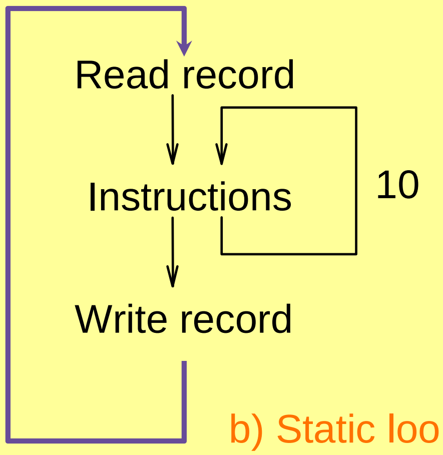
Characterize applications by the different parts of the architecture they affect.

Program Attributes: Control



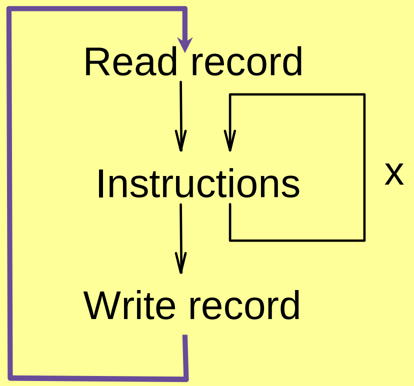
- *Vector or SIMD control*
- *Example: single vadd*

a) Sequential



- *Vector or SIMD control*
- *Branching required*
- *Example: DCT*

b) Static loop bounds



- *MIMD control*
- *Masking required for SIMD architectures*
- *Example: skinning*

c) Data dependent branching

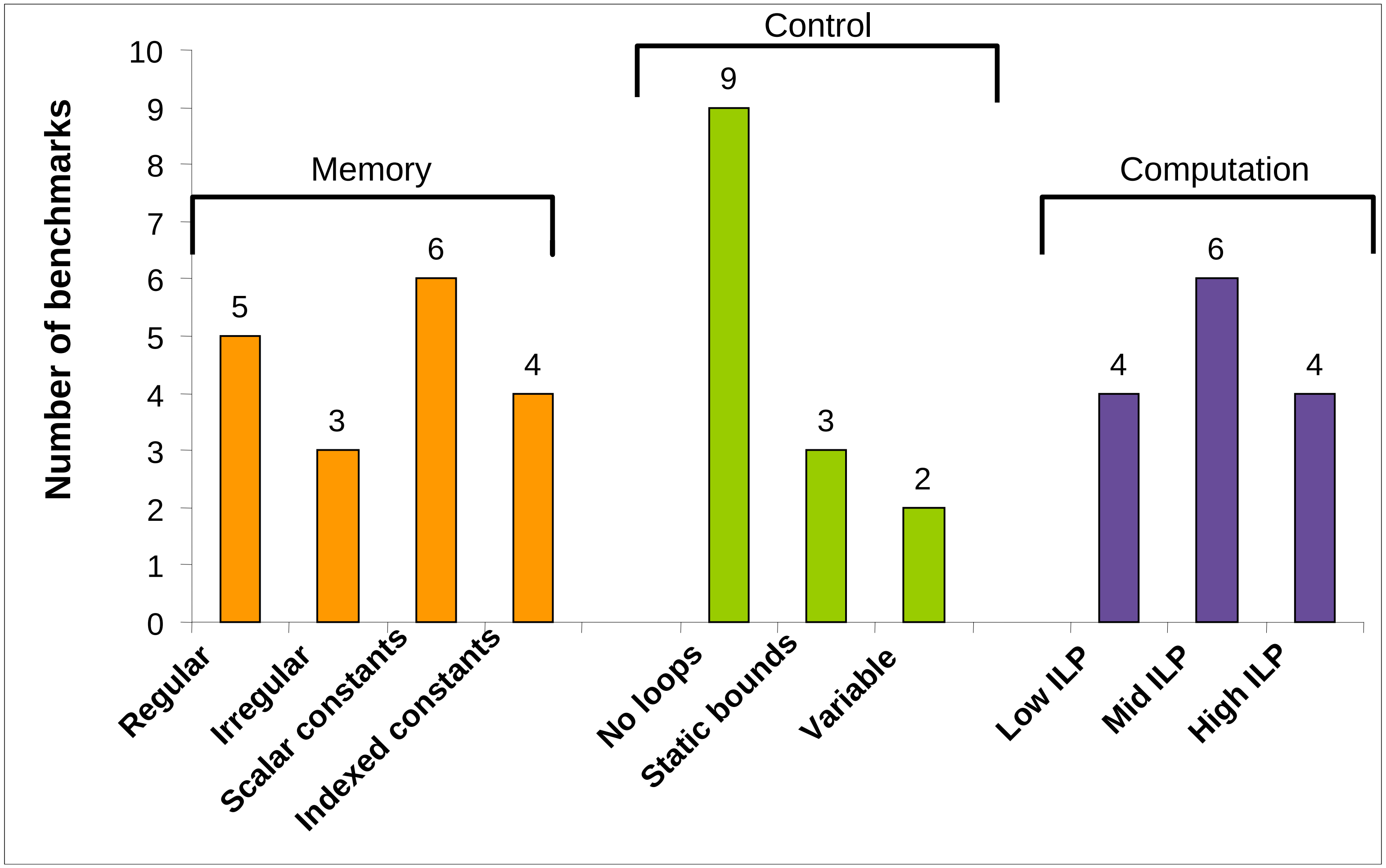
Program Attributes: Memory

- Regular memory
  - Memory accessed in structured regular fashion
  - Example: Reading image pixels in DCT compression
- Irregular memory accesses
  - Memory accessed in random access fashion
  - Example: Texture accesses in graphics processing
- Scalar constants
  - Run time constants typically saved in registers
  - Example: Convolution filter constants in DSP kernels
- Indexed constants
  - Small lookup tables
  - Example: Bit swizzling in encryption

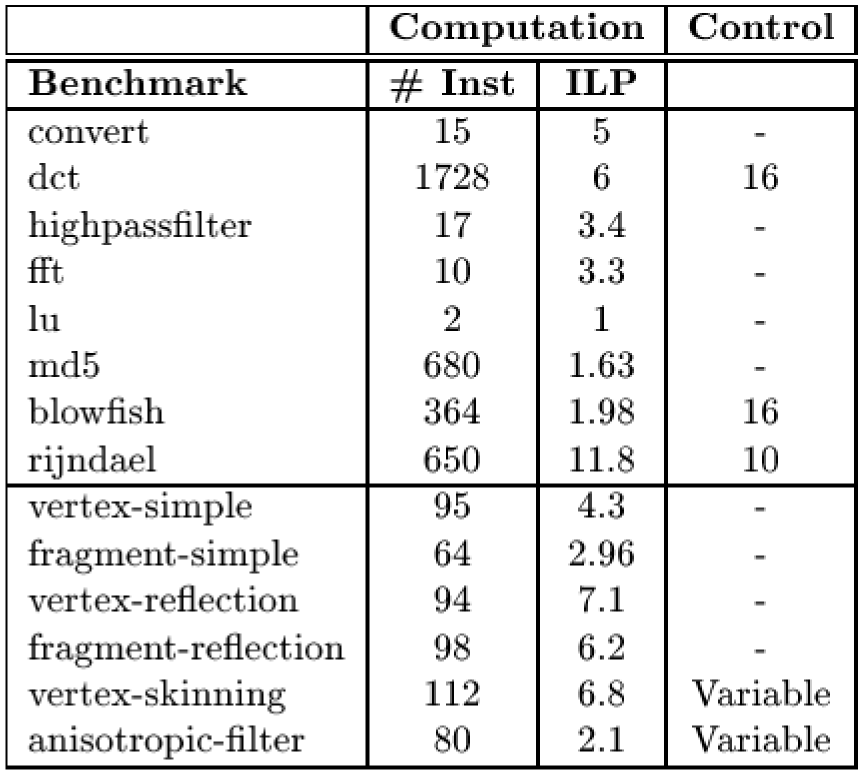
Benchmark Suite

Domain	**Benchmarks**
Multimedia processing	convert, dct, high pass filter
Scientific computing	fft, LU
Network processing, security	md5, rijndael, blowfish
Real-time graphics processing	vertex-simple, vertex-reflection, vertex-skinning, fragment-simple, fragment-reflection, anisotropic-filtering

Benchmark Attributes

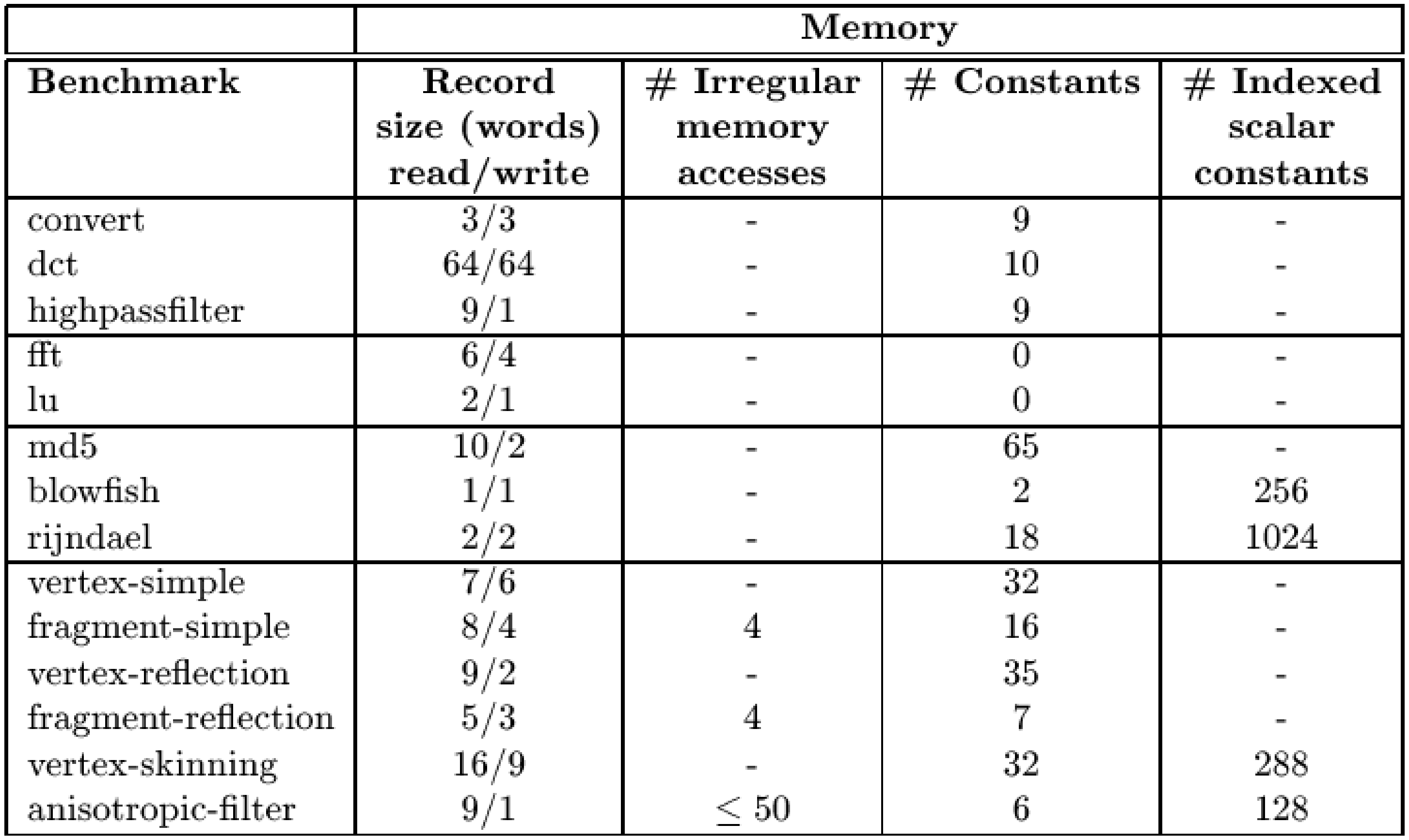


Benchmark Attributes



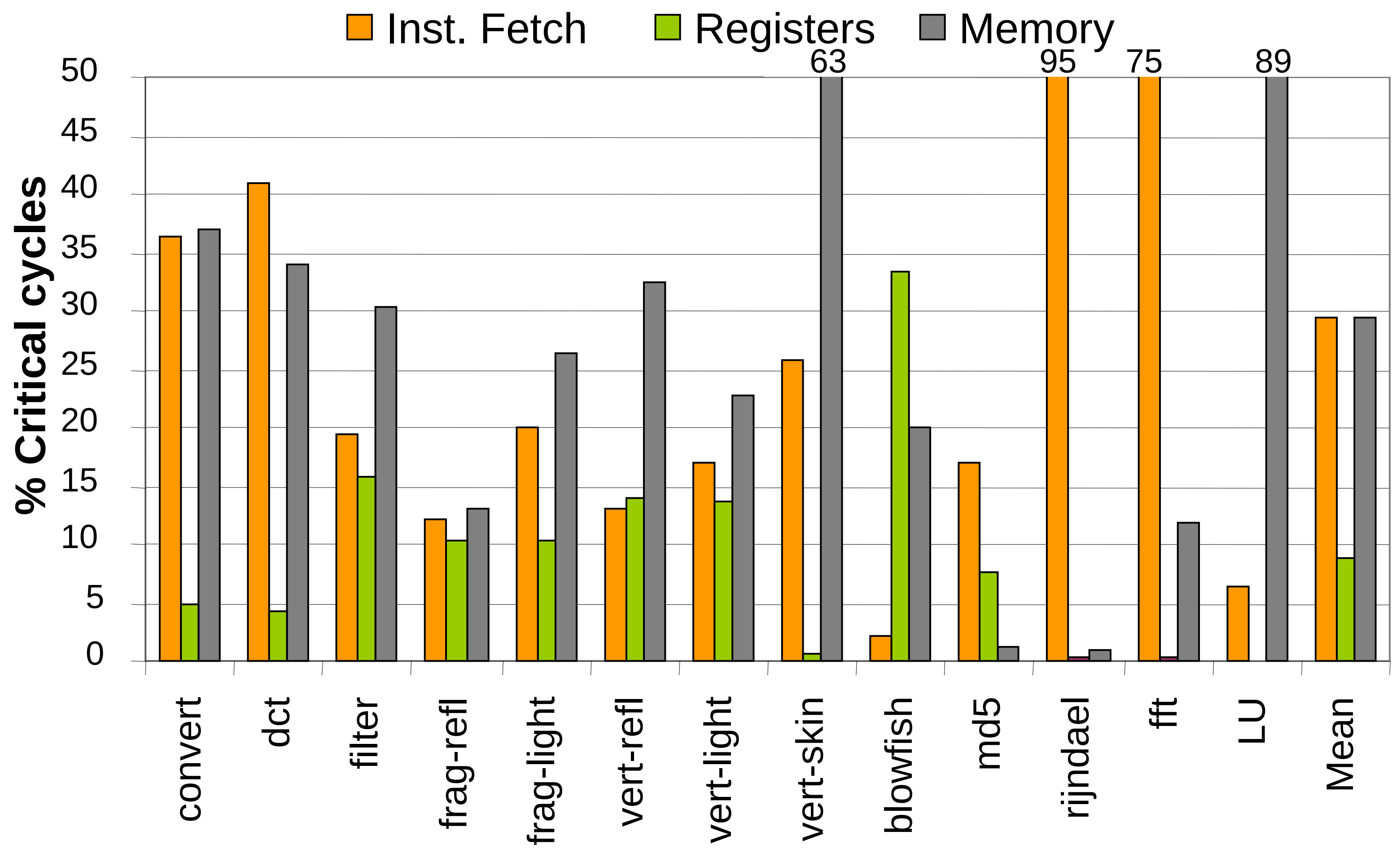
	Computation		Control
Benchmark	**# Inst**	**ILP**	
convert	15	5	-
dct	1728	6	16
highpassfilter	17	3.4	-
fft	10	3.3	-
lu	2	1	-
md5	680	1.63	-
blowfish	364	1.98	16
rijndael	650	11.8	10
vertex-simple	95	4.3	-
fragment-simple	64	2.96	-
vertex-reflection	94	7.1	-
fragment-reflection	98	6.2	-
vertex-skinning	112	6.8	Variable
anisotropic-filter	80	2.1	Variable

Benchmark Attributes

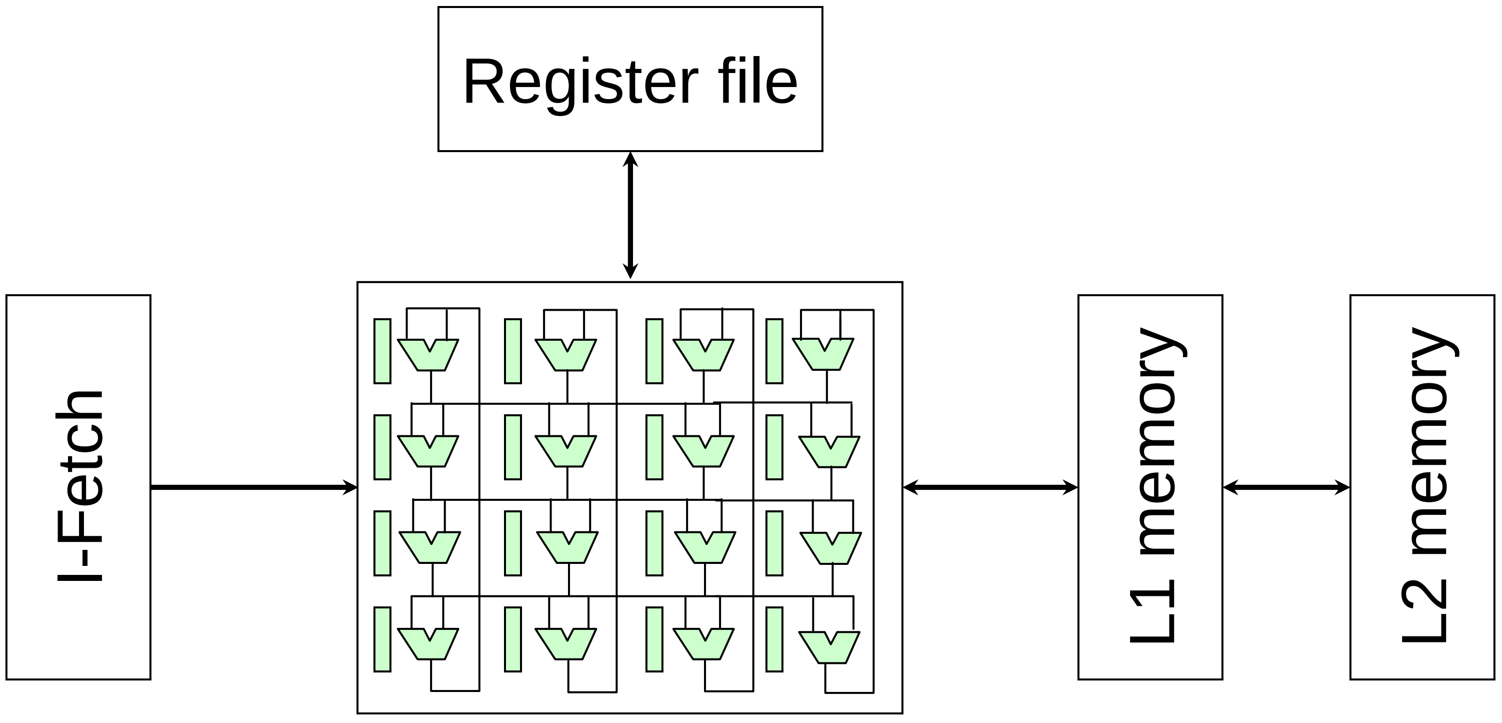


	Memory			
Benchmark	**Record size (words) read/write**	**# Irregular memory accesses**	**# Constants**	**# Indexed scalar constants**
convert	3/3	-	9	-
dct	64/64	-	10	-
highpassfilter	9/1	-	9	-
fft	6/4	-	0	-
lu	2/1	-	0	-
md5	10/2	-	65	-
blowfish	1/1	-	2	256
rijndael	2/2	-	18	1024
vertex-simple	7/6	-	32	-
fragment-simple	8/4	4	16	-
vertex-reflection	9/2	-	35	-
fragment-reflection	5/3	4	7	-
vertex-skinning	16/9	-	32	288
anisotropic-filter	9/1	≤ 50	6	128

DLP Bottlenecks

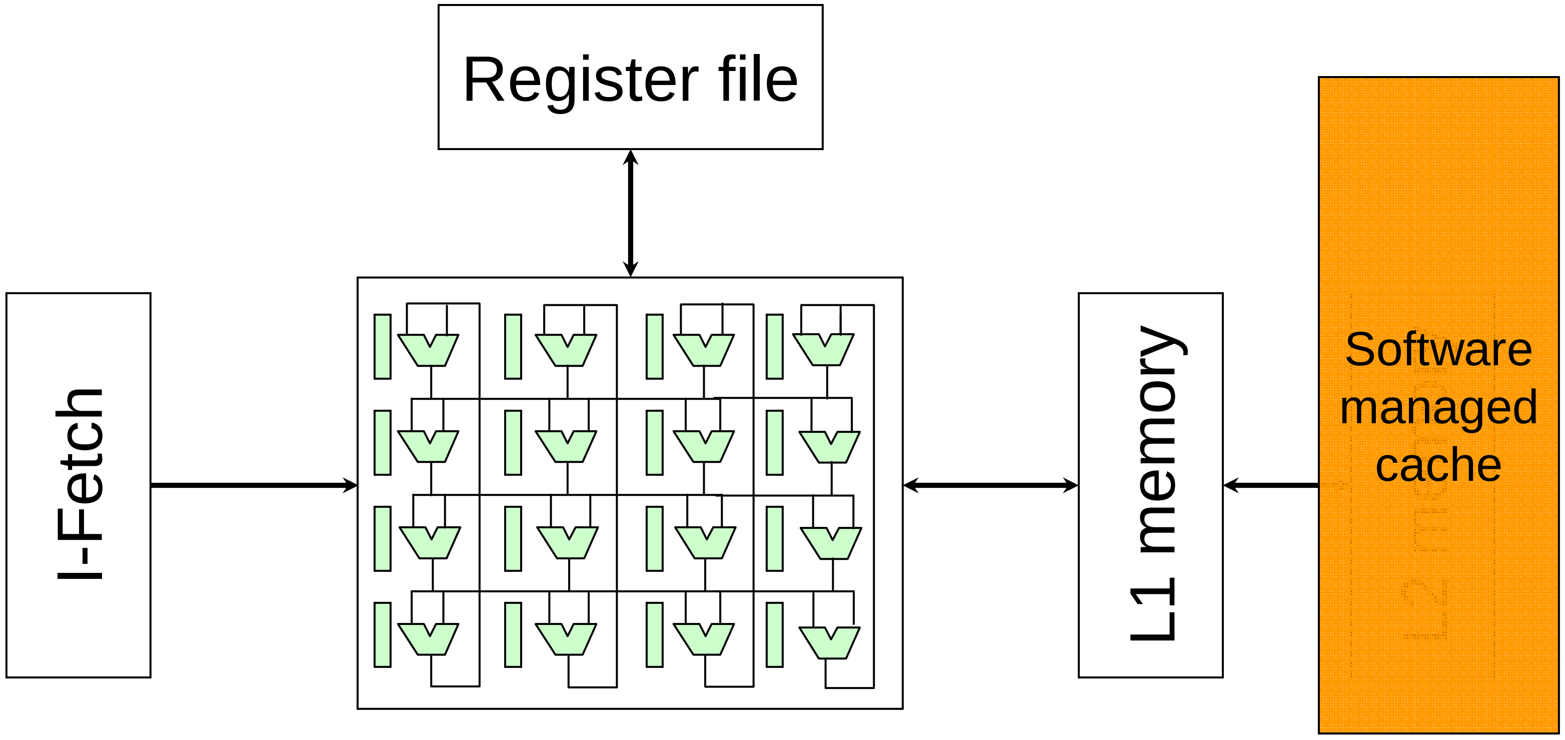


High Level Architecture



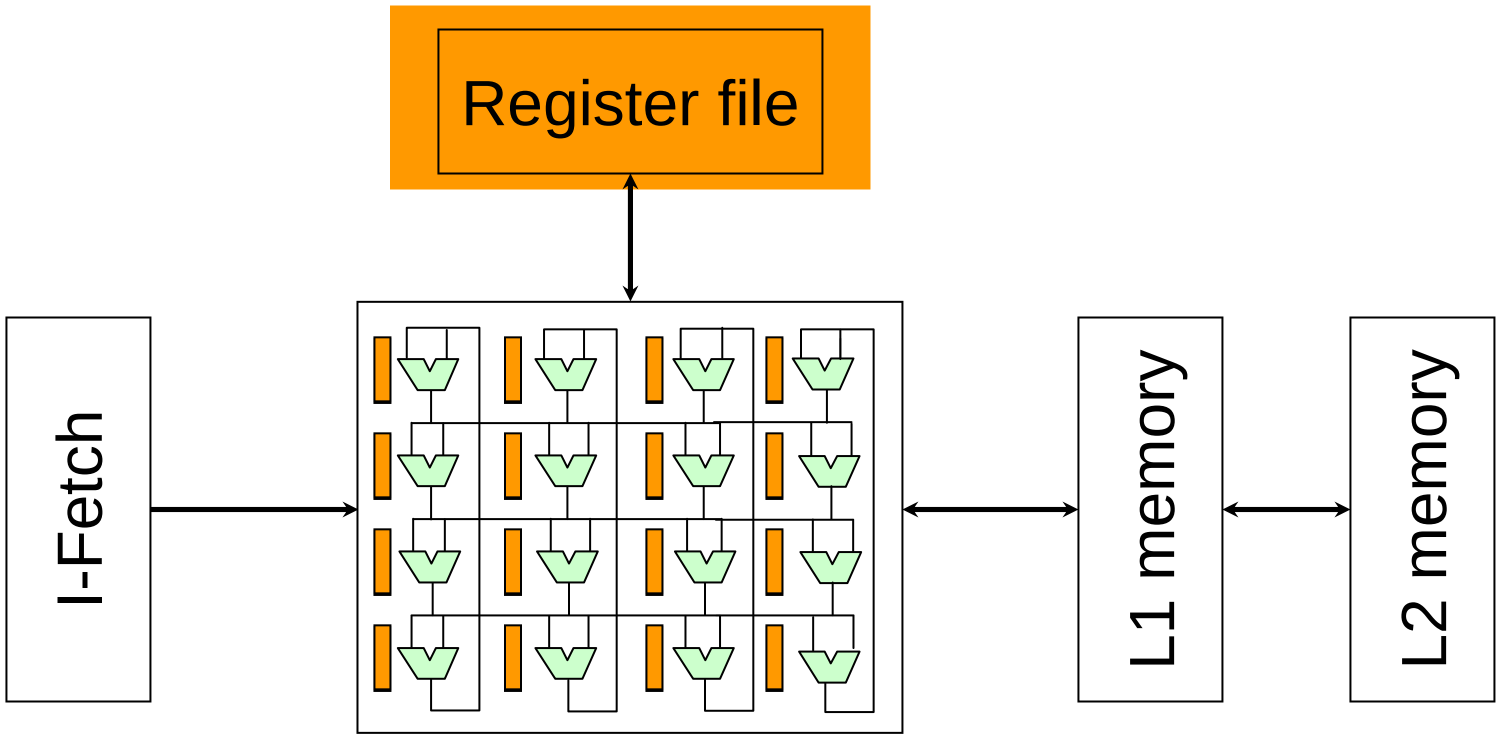
DLP Attributes and Mechanisms

Regular memory accesses



DLP Attributes and Mechanisms

Regular memory accesses
Scalar named constants



DLP Attributes and Mechanisms

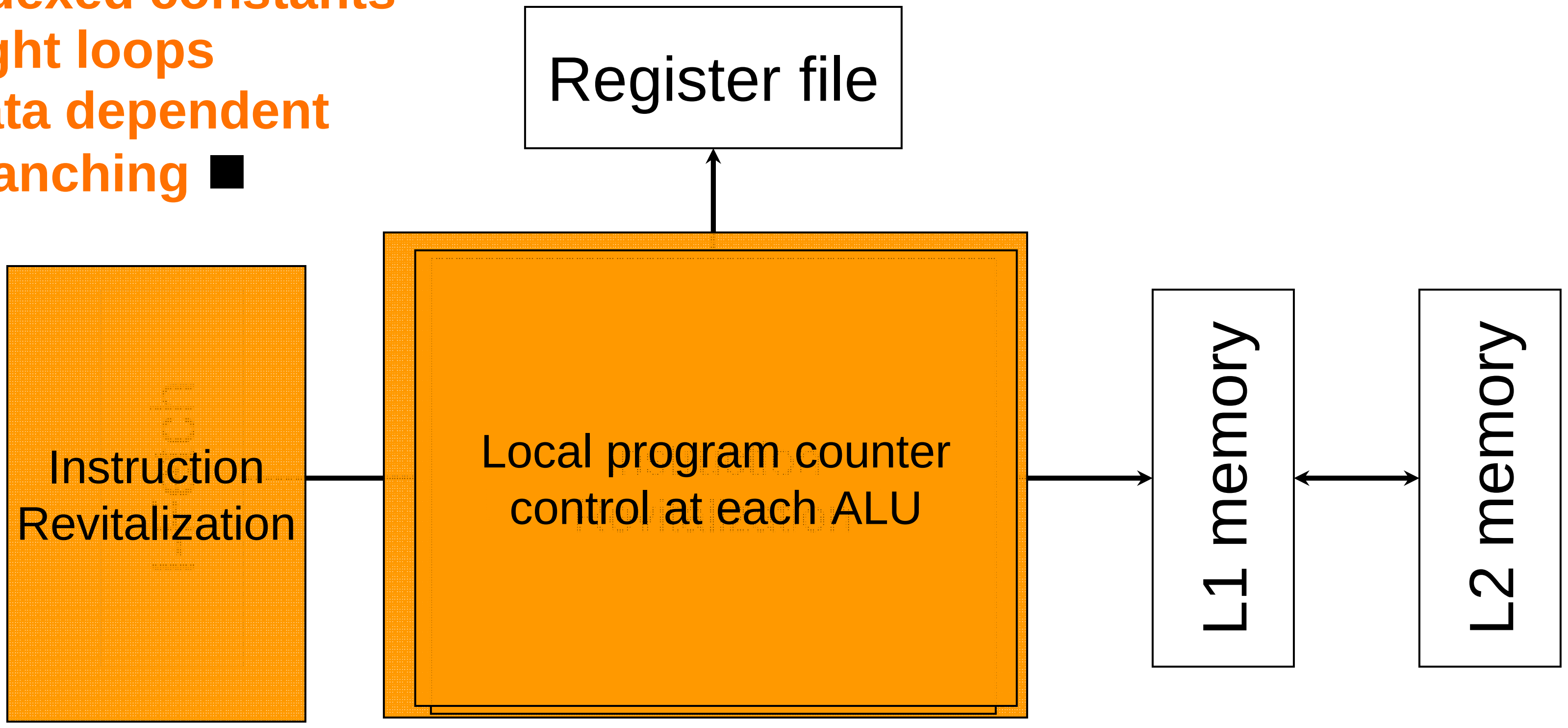
Regular memory accesses

Scalar named constants

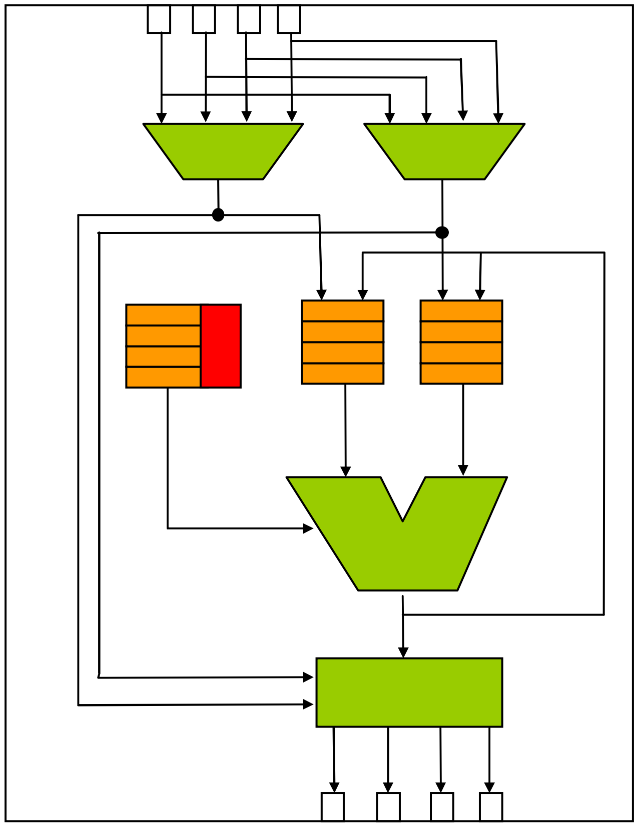
Indexed constants

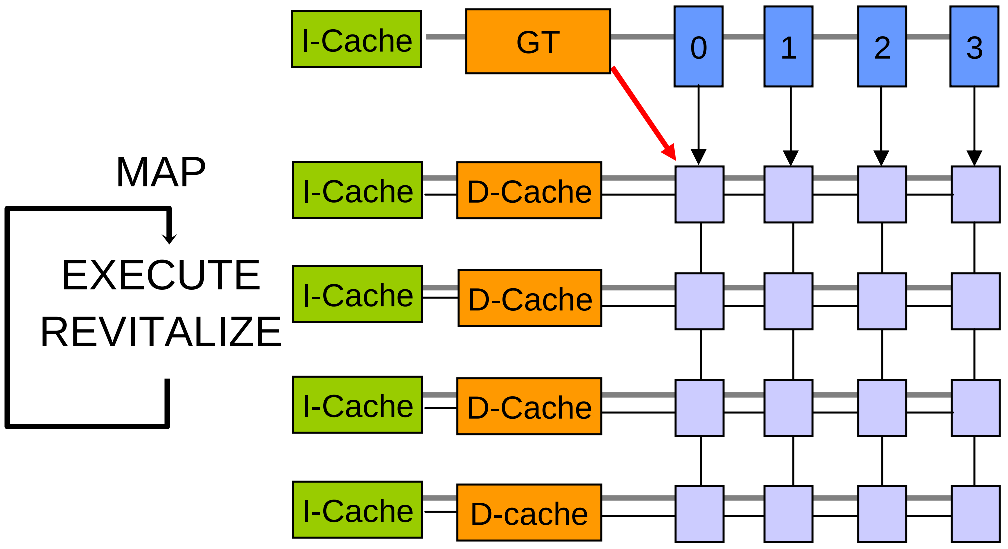
Tight loops

Data dependent branching ■


Register file
Instruction Revitalization
Local program counter control at each ALU
L1 memory
L2 memory

I-Fetch and Control Mechanisms(1)

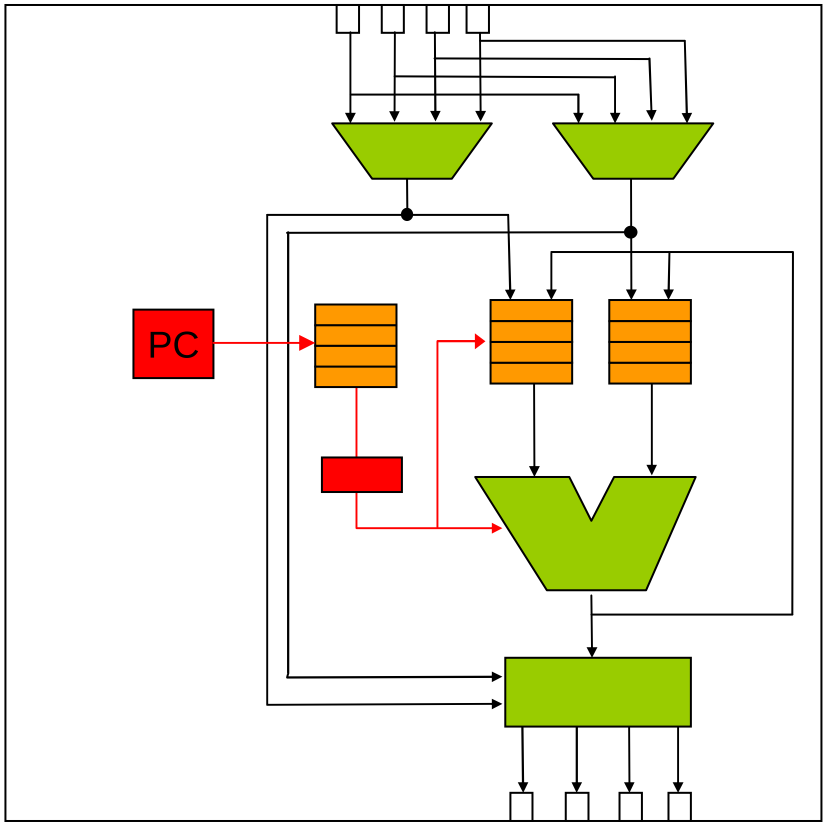


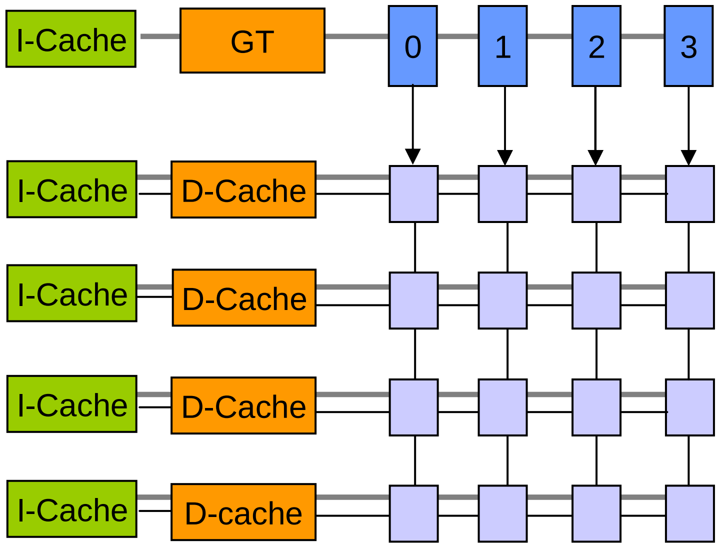


Instruction revitalization to support tight loops

- Dynamically create a loop engine
- Power savings, I-Caches accessed once

I-Fetch and Control Mechanisms(2)





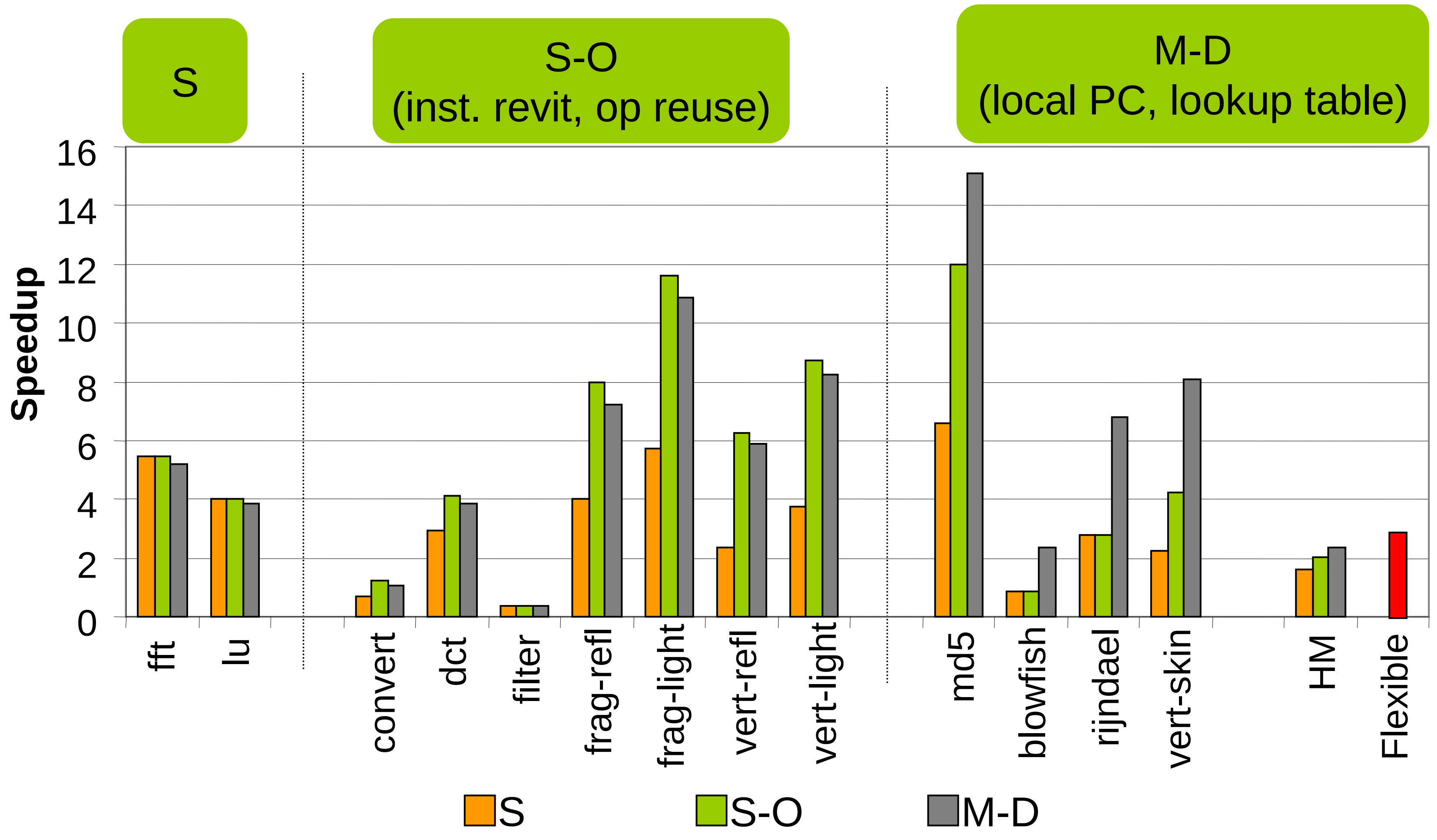
- Local PCs for data dependent branching
- Reservation stations are now I-Caches

MIMD Execution Array

Results

- Baseline Machine:
  - 4x4 TRIPS processor with a mesh interconnect
- Kernels hand-coded, placed using custom schedulers
- DLP mechanisms combined to produce 3 configurations
  - Software managed cache + Instruction Revitalization **(S)**
  - Software managed cache + Instruction Revitalization + Operand Reuse **(S-O)**
  - Software managed cache + Local PCs + Lookup table support **(M-D)**
  - Of possible 20 these are most meaningful; operand reuse without instruction revitalization does not make sense for example
- Performance comparison against specialized hardware

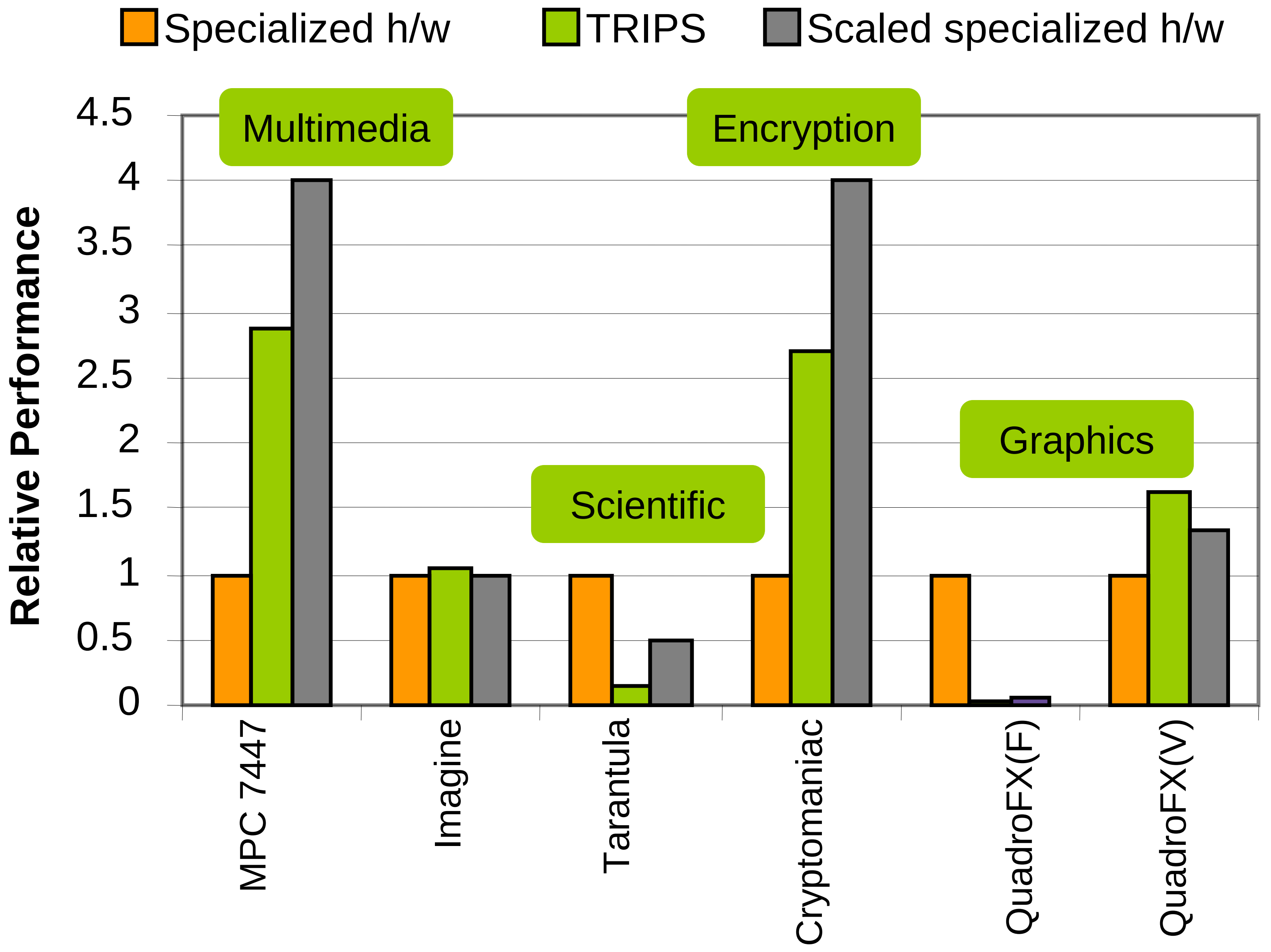
Evaluation of Mechanisms



Comparison to Specialized H/W

- Pick “best” specialized processor for each workload
- Normalize TRIPS clock to specialized processor:
  - Scale both to 10FO4
- Normalize area based on functional units
  - TRIPS is 16-issue, but MPC 7447 is 4-issue
  - Multiply performance of MPC 7447 by 4
- Optimistic scaling of specialized processors

Comparison to Specialized Hardware



Summary

- Architectural polymorphism implemented using small set of mechanisms
  - Effective thread level parallelism support
  - Detailed analysis of DLP
  - Mechanisms provide competitive performance compared to specialized processors
- EDGE ISA
  - Dataflow graph abstraction
- TRIPS prototype processor
  - Distributed microarchitecture design principles

Conclusions

- Challenges
  - Application heterogeneity
  - Technology limitations (power and wire delays)
- Architectural polymorphism
  - Coarse grain microarchitectural reconfiguration
  - Scalable modular blocks provide scalability
- Future work
  - Compilation for polymorphous architectures
  - Polymorphism to achieve higher power and area efficiency

Publications

- Dataflow Predication, *MICRO 2006*
- Distributed Microarchitectural Protocols in the TRIPS Prototype Processor, *MICRO 2006*
- TRIPS: A polymorphous architecture for exploiting ILP, TLP, and DLP, *TACO 2004*
- Universal Mechanisms for Data-Parallel Architectures, *MICRO 2003*
- Routed Inter-ALU Networks for ILP Scalability and Performance, *ICCD 2003*
- Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture, *ISCA 2003*
- A Design Space Exploration of Grid Processor Architectures, *MICRO 2001*

TRIPS Prototype

• *High level microarchitecture*	– Ramdas Nagarajan and Karu Sankaralingam	• 2001
• LSQ	– Simha Sethumadhavan and Raj Desikan	• 2002-2003
• Next block predictor	– Nitya Ranganathan	• 2002-
• *ISA design*	– Ramdas Nagarajan, Robert McDonald, and Karu Sankaralingam	• 2004
• *Prototype microarchitecture spec. and modeling*	– Ramdas Nagarajan, Haiming Liu, Nitya Ranganathan, Simha Sethumadhavan, Premkishore Shivakumar, Diyva Gulati, Heather Hanson, and Karu Sankaralingam	• 2004
• NUCA cache and OCN design	– Changkyu Kim and Paul Gratz	• 2002-2005
• *Logic design and verilog*	– RT, OPN	• 2004-2005
• *Processor level verification*		• 2005
• Chip level verification		• 2005
• *Physical design*		• 2005-2006
• Fabrication		• 2006
• System bringup		• 2006-? (☺)

Questions