

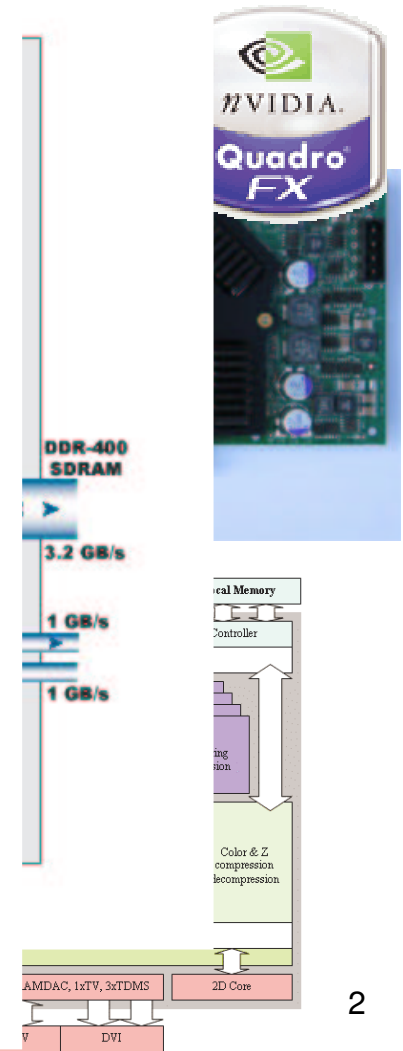
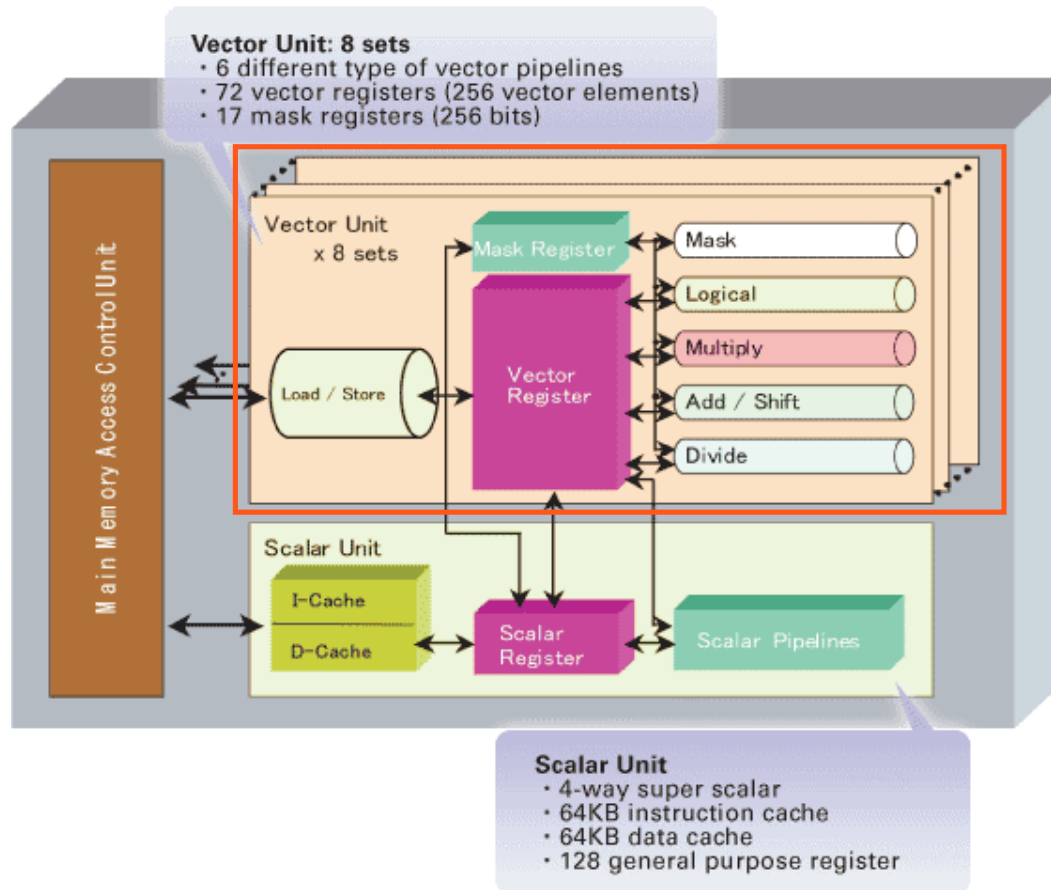
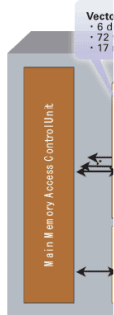
Universal Mechanisms for Data-Parallel Architectures

Karu Sankaralingam

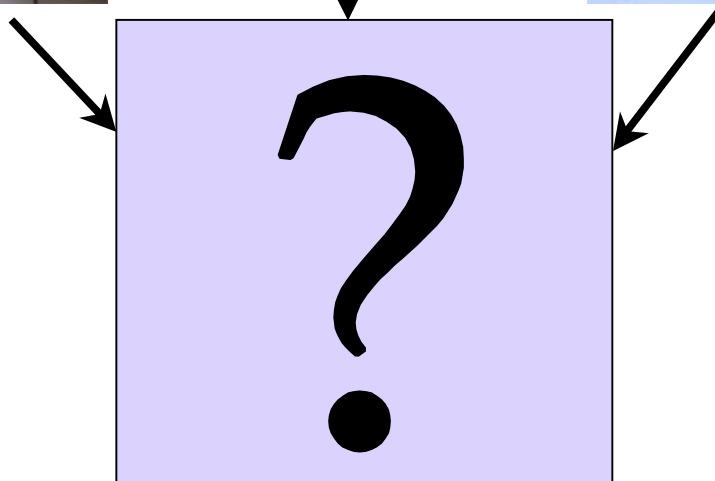
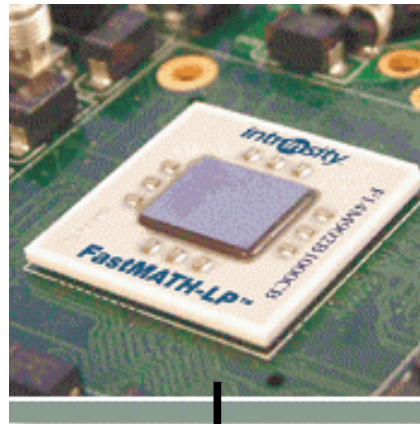
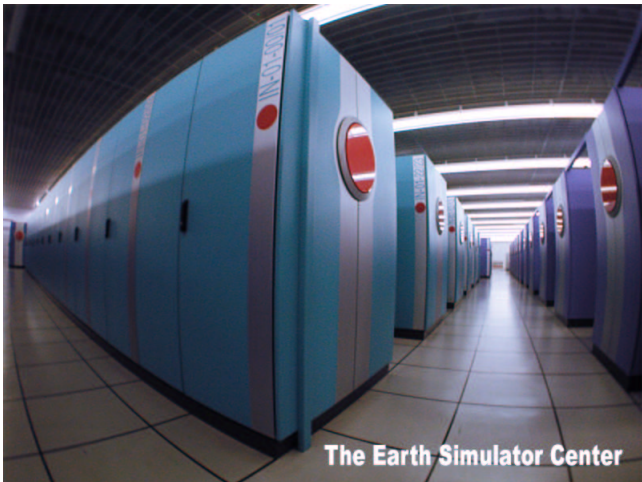
Stephen W. Keckler, William R. Mark, and Doug Burger

Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin

Data-Parallel Systems



One Programmable Architecture?



Conventional Data-Parallel Architectures

- Vector, SIMD, and MIMD architectures
- Specialized narrowly focused hardware
 - MPEG4 decoding
 - Specialized error correction code units, convolution encoding in DSPs

Architecture trends: Programmability

- Programmable DLP hardware is emerging
 - Sony EmotionEngine: 2 specialized vector units
 - Real-time graphics processors: a specialized pixel processor and vertex processor
 - Sony Handheld Engine: a DSP core, 2D graphics core, and ARM core
- But still *specialized*
 - Several types of processors have to be designed
 - Application mix must match composition
 - Integration costs
- **Can we increase the level of programmability to encompass diversity?**

A Unified Systematic Approach to DLP

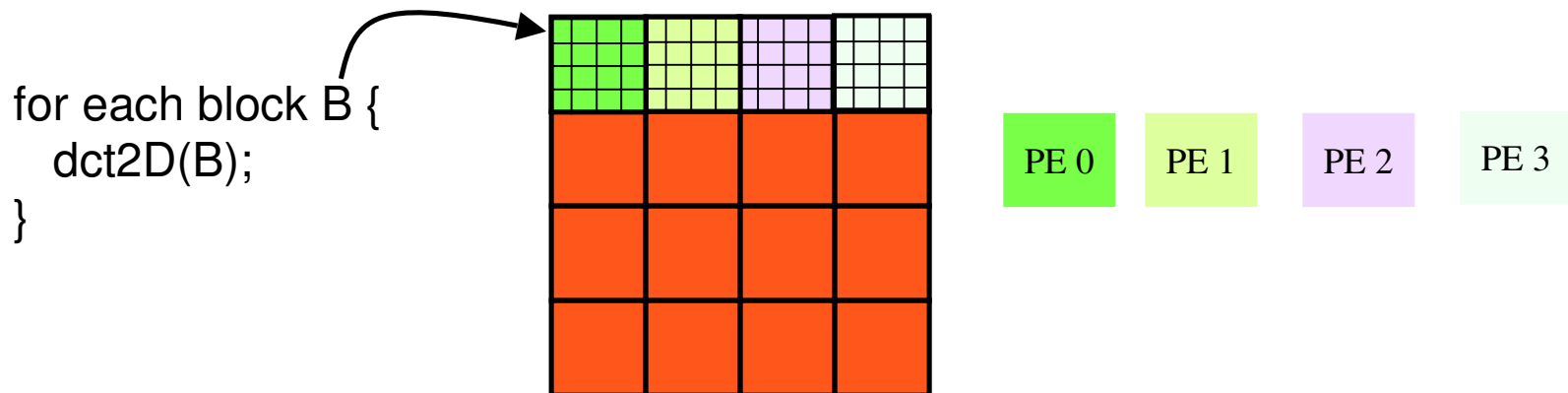
- What are the basic characteristics and demands of DLP applications?
 - Instruction fetch and control
 - Memory
 - Computation
- Design a set of complementary universal mechanisms
 - Dynamically adapt a single architecture based on application demands
 - Can be applied to diverse architectures

Outline

- Application study
 - Application behavior
 - Benchmark suite
 - Benchmark attributes
- Microarchitecture mechanisms
 - Instruction fetch and control
 - Memory system
 - Execution core
- Results
- Conclusions

Application Behavior

- DLP program model: Loops executing on different parts of memory in parallel
- Example 1: DCT of an image
 - Identical computation at each PE
 - Globally synchronous computation



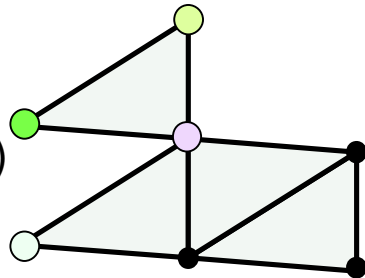
Application Behavior

- DLP program model: Loops executing on different parts of memory in parallel
- Example 2: Vertex skinning
 - Data dependent branching at each PE

```
for each vertex V {
```

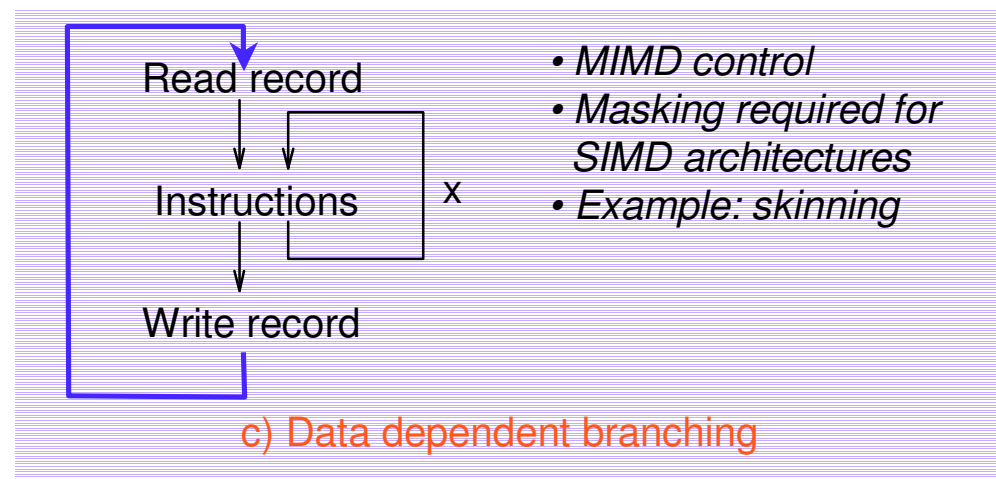
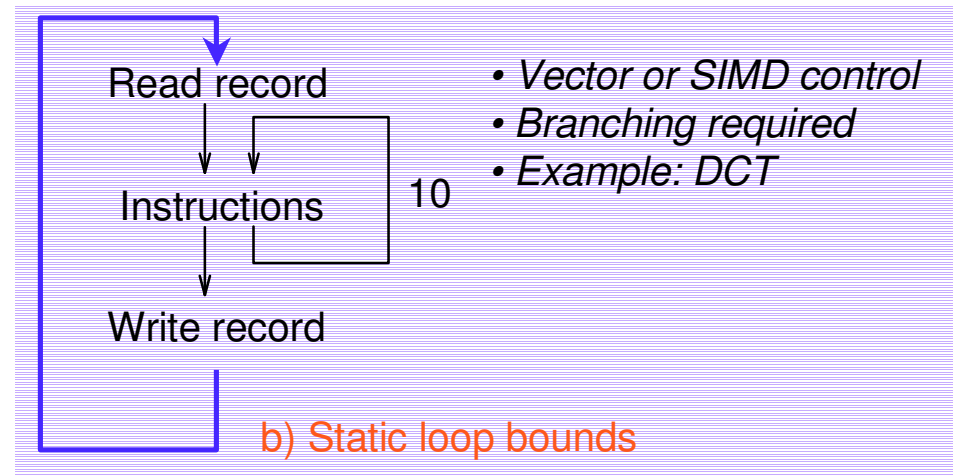
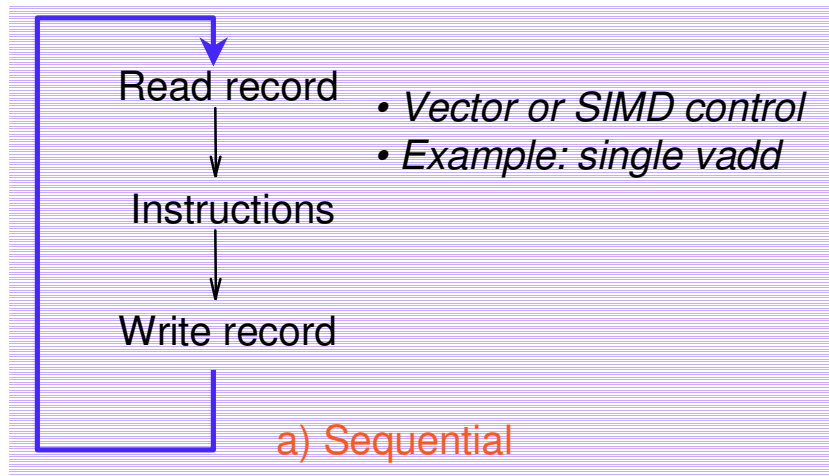
```
  for (j = 0; j < V.ntrans; j++) {  
    Z = Z + product(V.xyz, M[j])
```

```
  }  
}
```



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

Program Attributes: Control

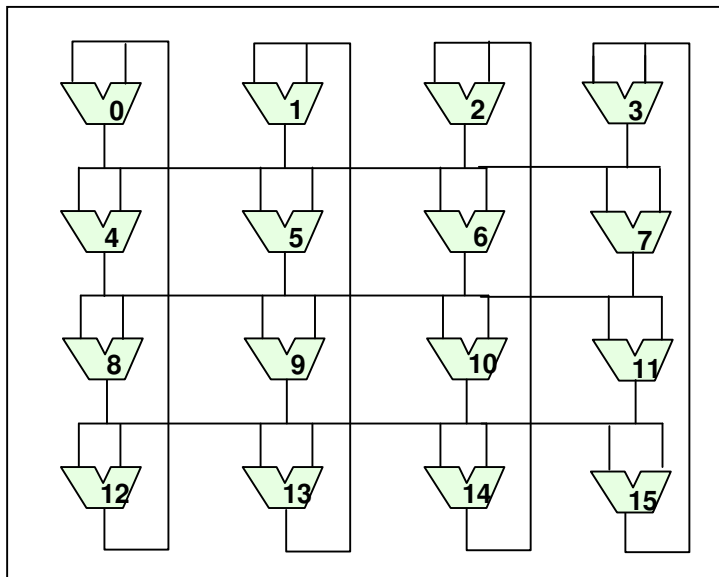


Program Attributes: Memory

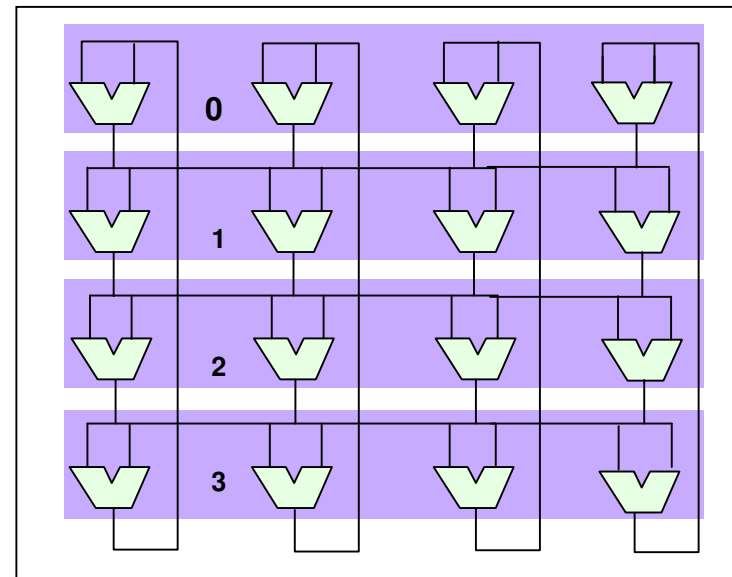
- Regular memory
 - Memory accessed in structured regular fashion
 - Example: Reading image pixels in DCT compression
- Irregular memory accesses
 - Memory accessed in random access fashion
 - Example: Texture accesses in graphics processing
- Scalar constants
 - Run time constants typically saved in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

Program Attributes: Computation

- Instruction level parallelism within kernel
- ALUs per iteration of kernel



Low ILP

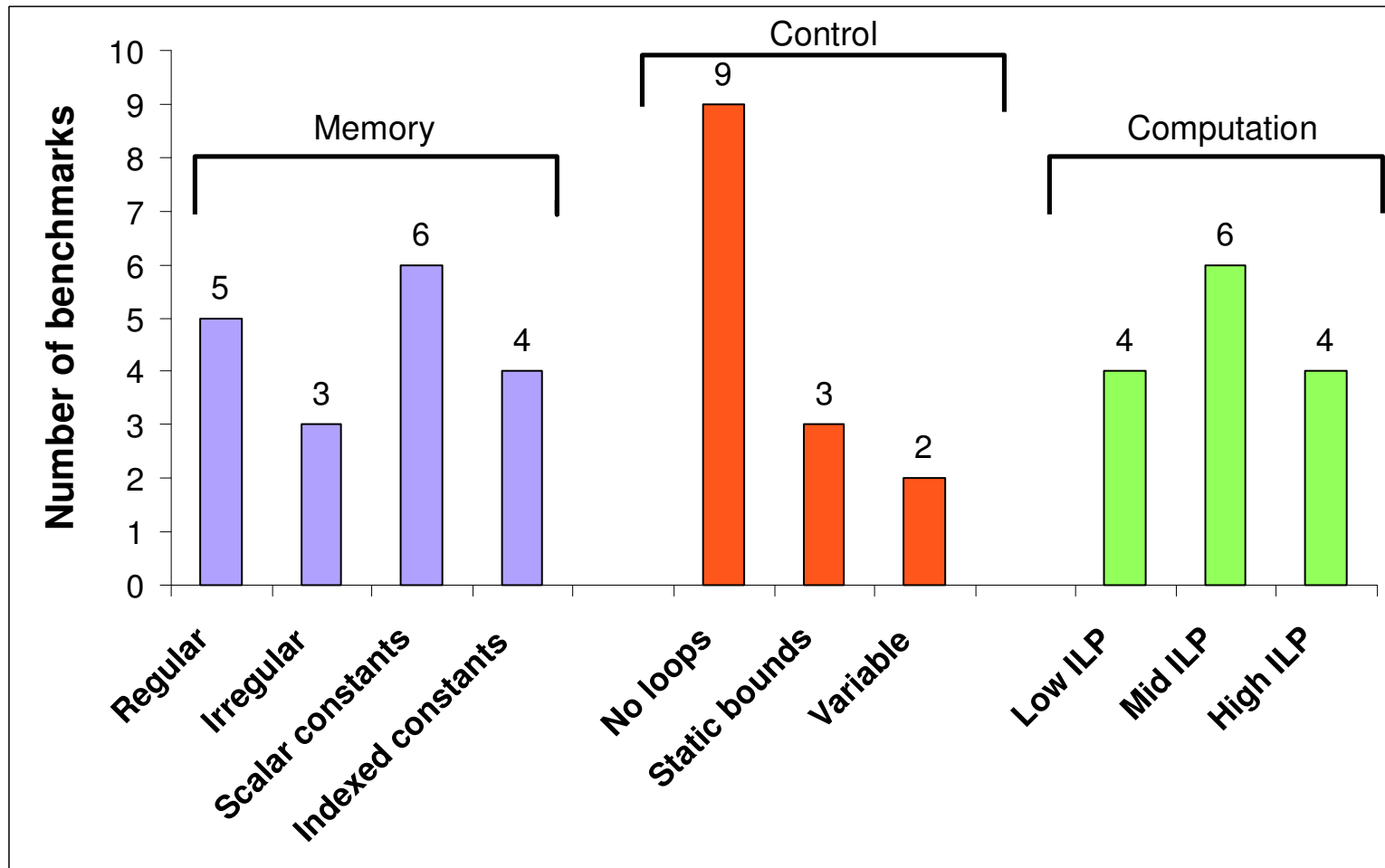


High ILP

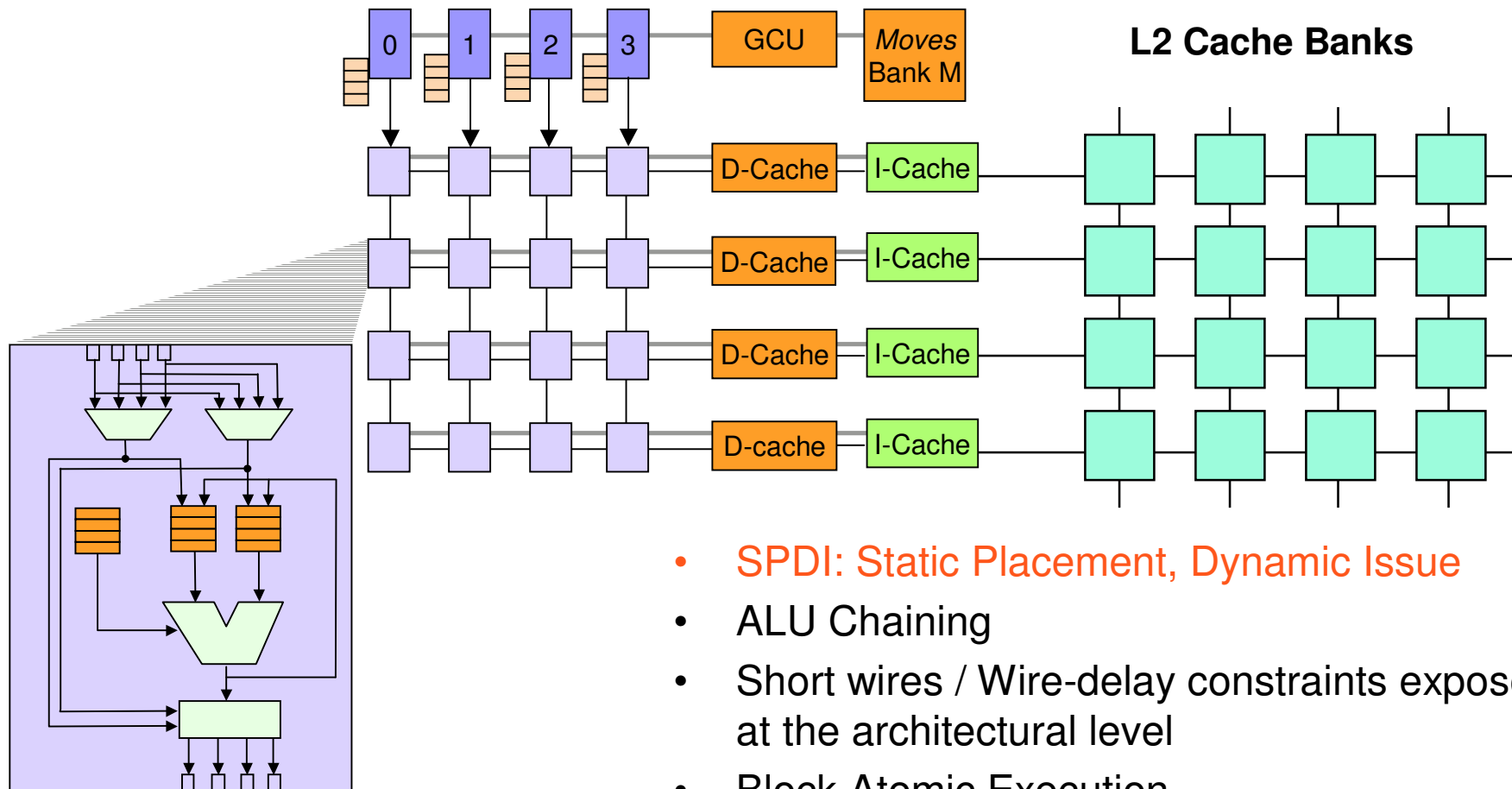
Benchmark Suite

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

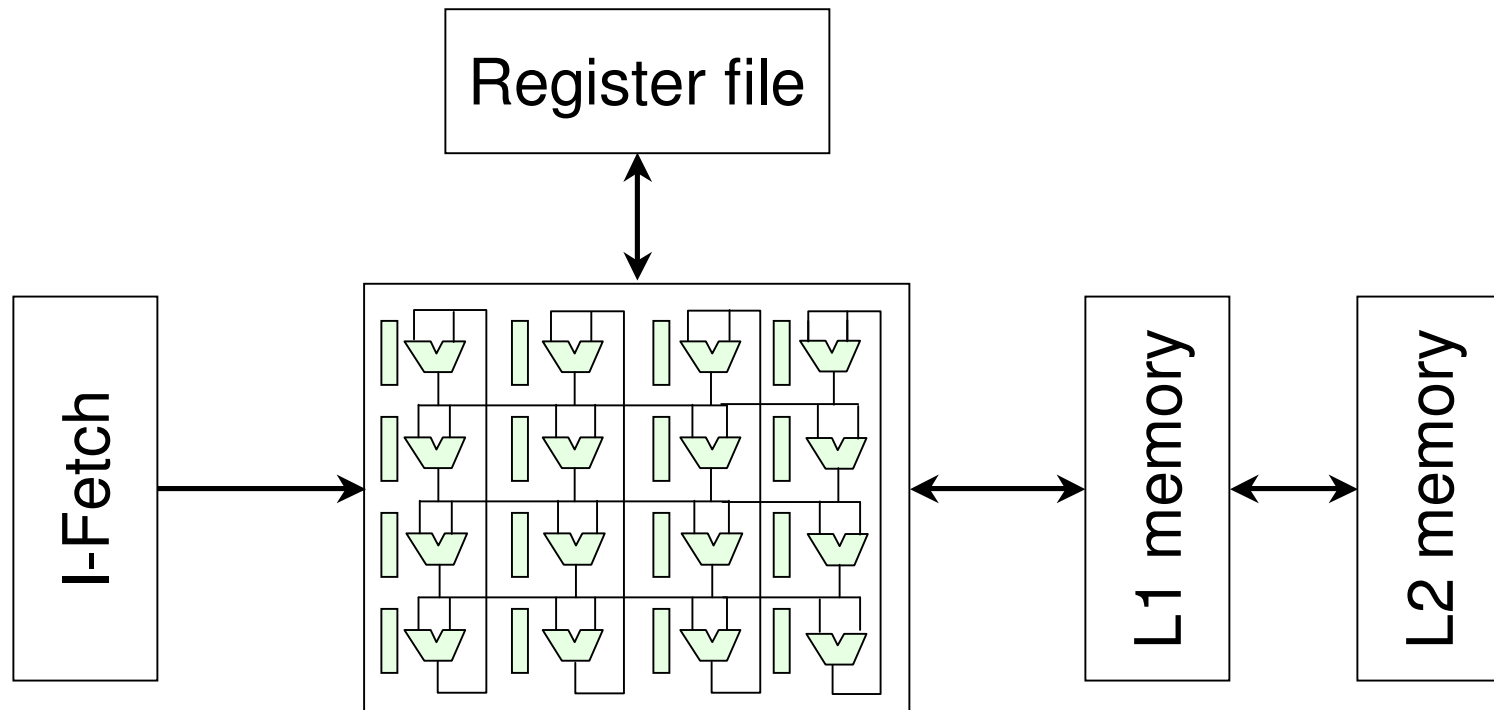


Baseline: TRIPS Processor



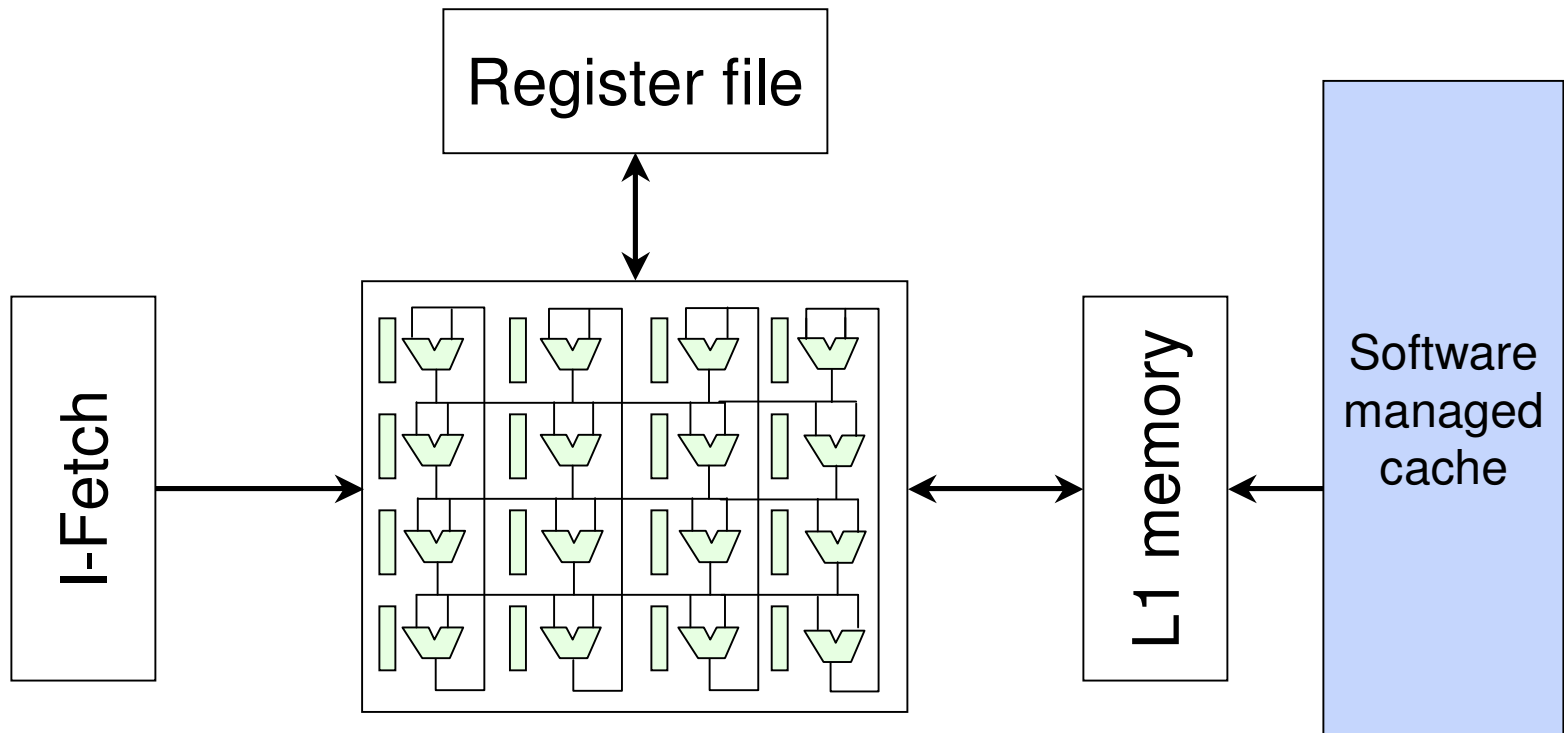
- SPDI: Static Placement, Dynamic Issue
- ALU Chaining
- Short wires / Wire-delay constraints exposed at the architectural level
- Block Atomic Execution

High Level Architecture



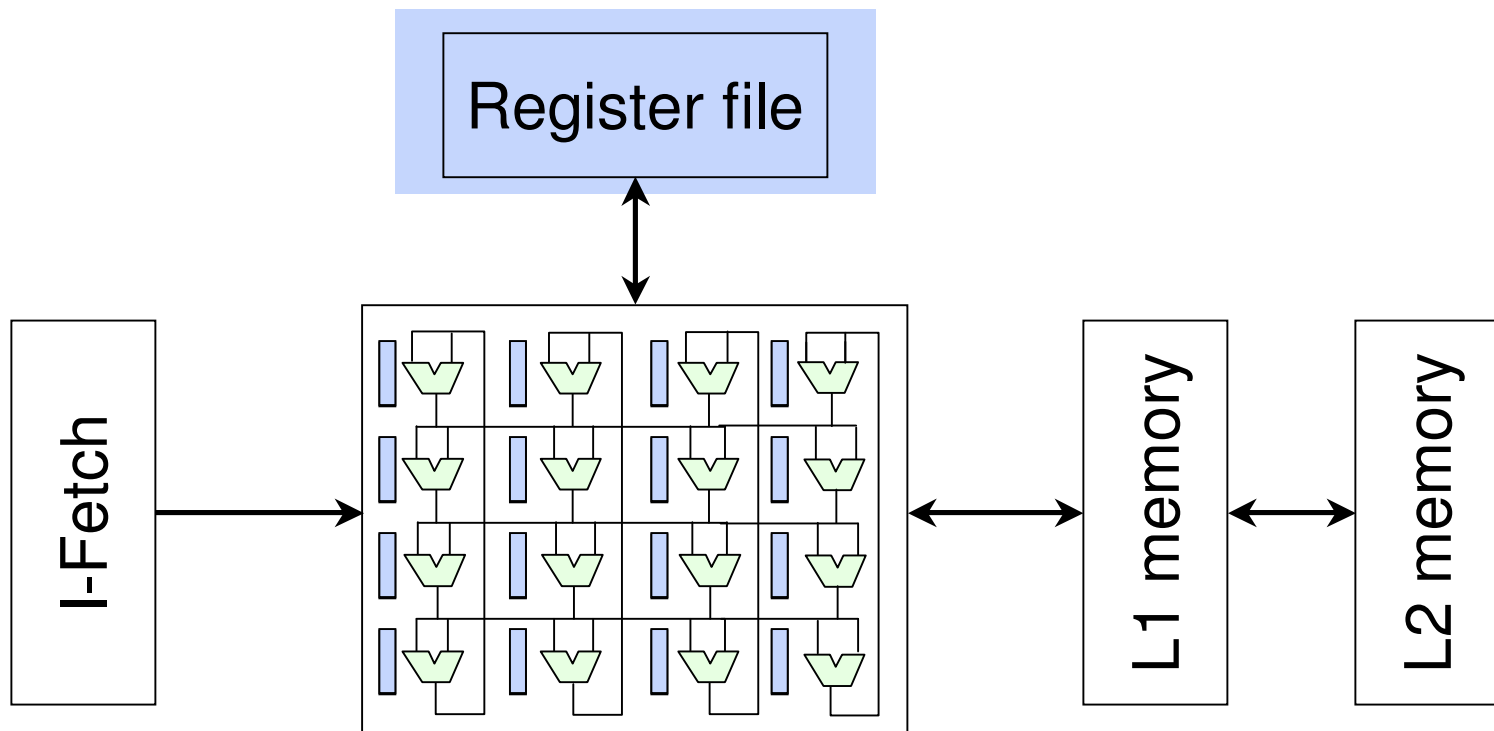
DLP Attributes and Mechanisms

Regular memory accesses



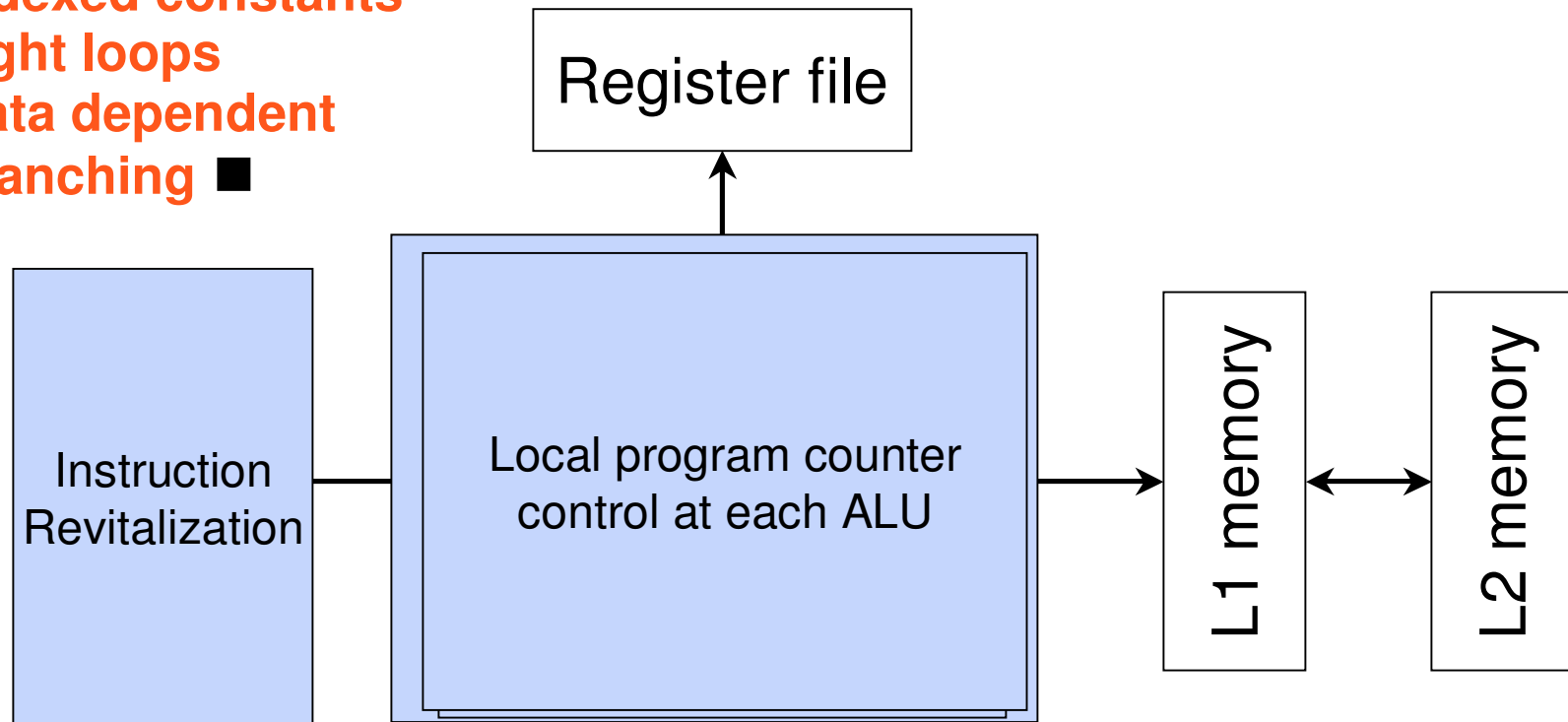
DLP Attributes and Mechanisms

Regular memory accesses
Scalar named constants

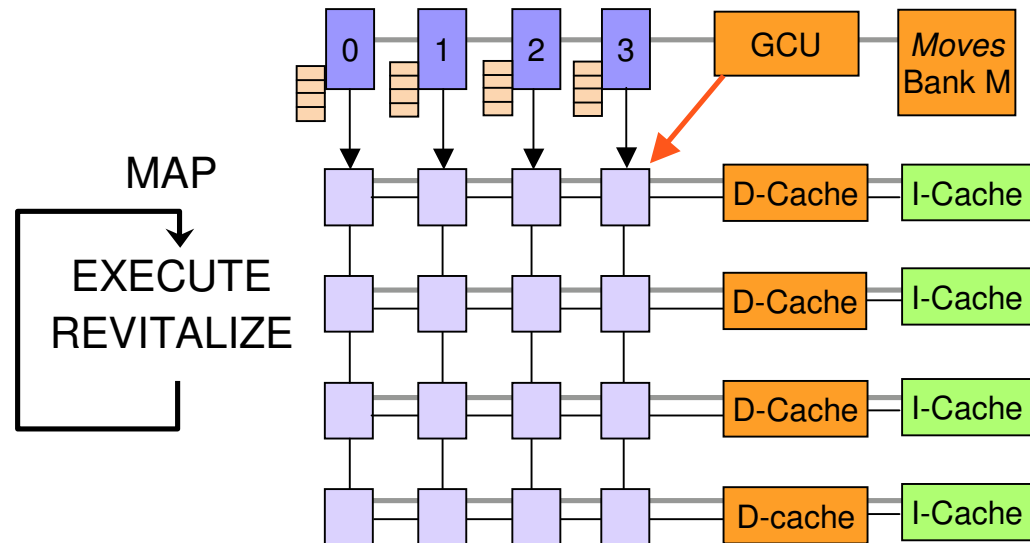
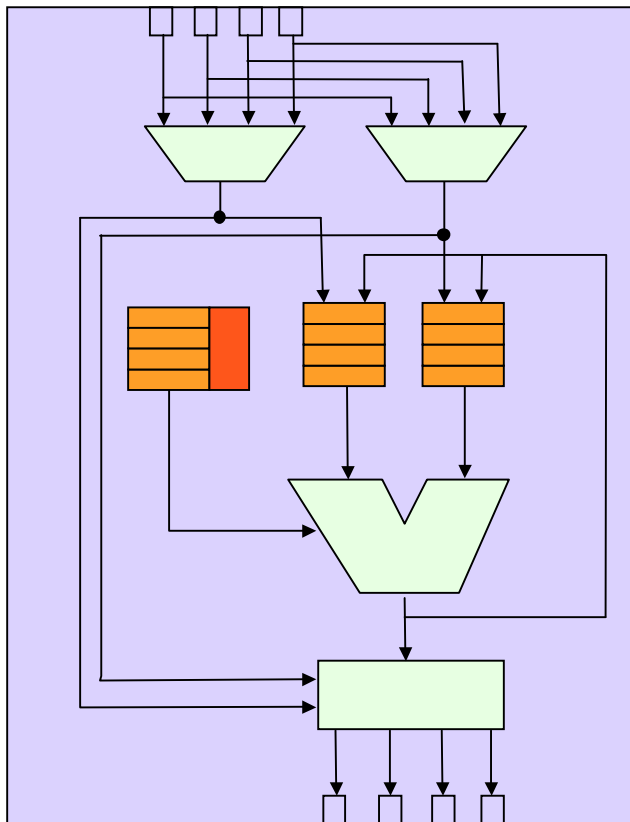


DLP Attributes and Mechanisms

Regular memory accesses
Scalar named constants
Indexed constants
Tight loops
Data dependent
branching ■



I-Fetch and Control Mechanisms(1)



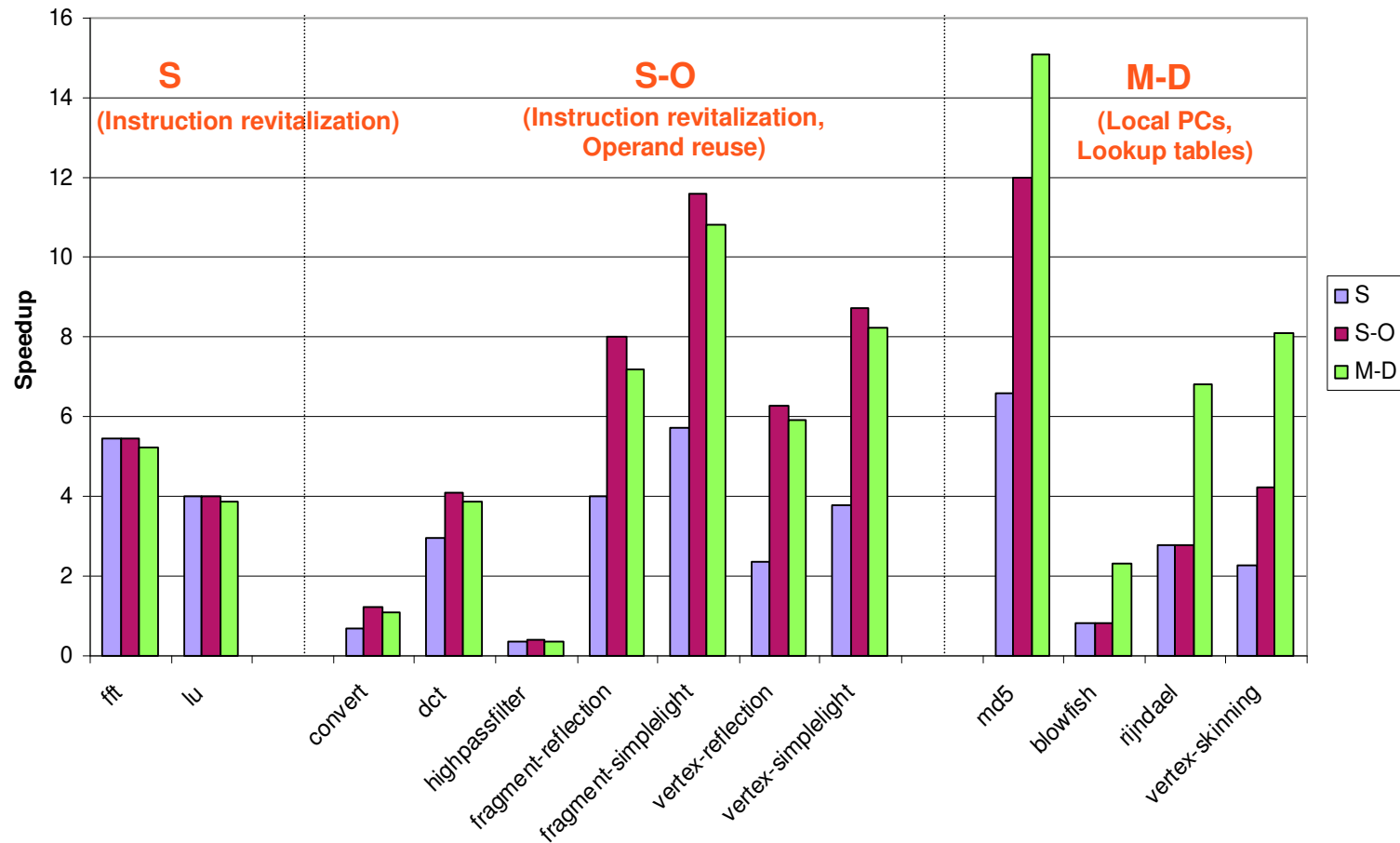
Instruction revitalization to support tight loops

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

Results

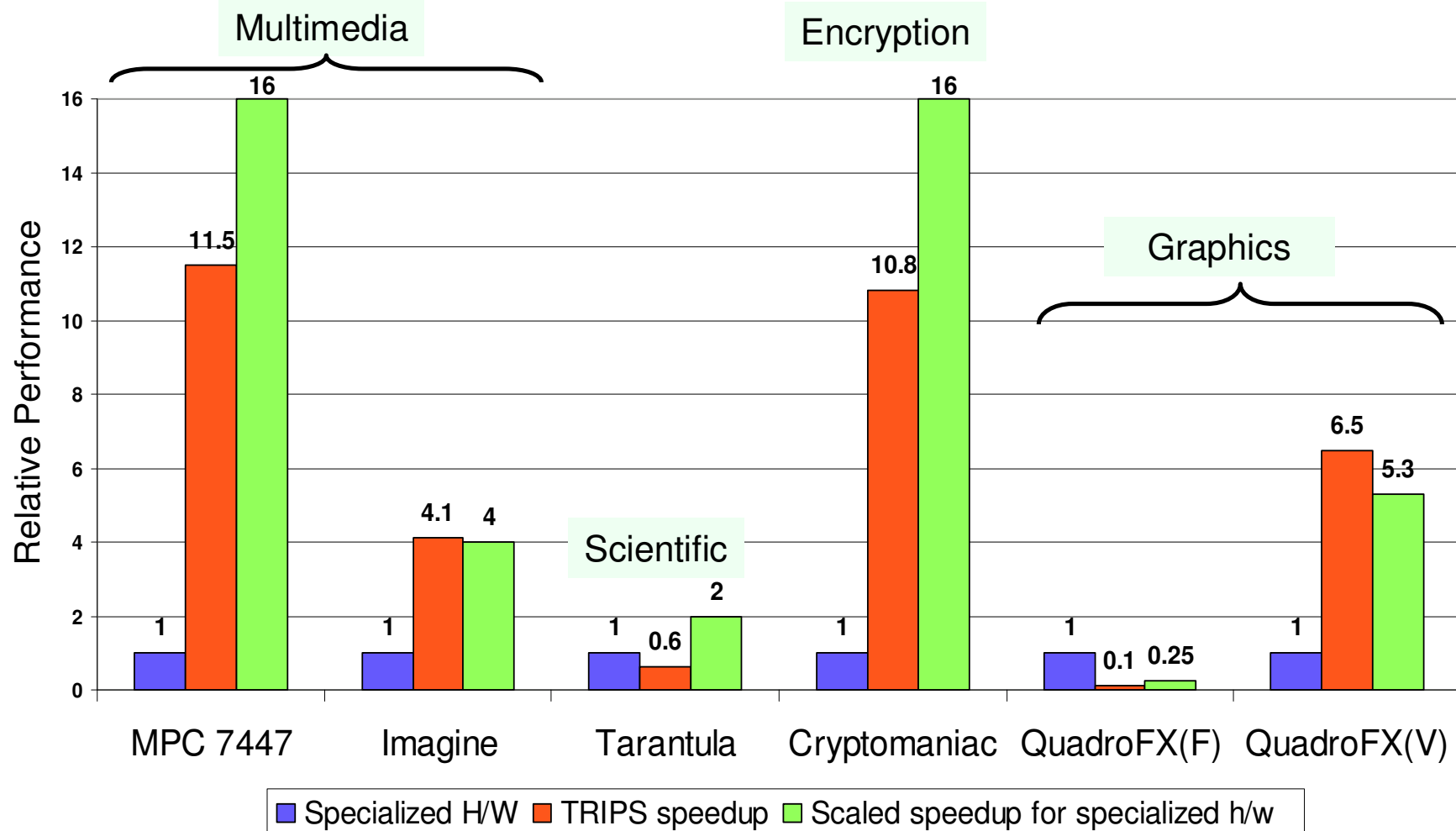
- Baseline Machine:
 - 8x8 TRIPS processor with a mesh interconnect
 - 100nm technology at 10FO4 clock rate
- Kernels hand-coded, placed using custom schedulers
- DLP mechanisms combined to produce 3 configurations
 - Software managed cache + Instruction Revitalization **(S)**
 - Software managed cache + Instruction Revitalization + Operand Reuse **(S-O)**
 - Software managed cache + Local PCs + Lookup table support **(M-D)**
- Performance comparison against specialized hardware

Evaluation of Mechanisms



Baseline: 8x8 TRIPS processor

Comparison to Specialized Hardware



Conclusions

- Key DLP program attributes identified
 - Memory, Control, and Computation
- Complementary universal mechanisms
- Competitive performance compared to specialized processors

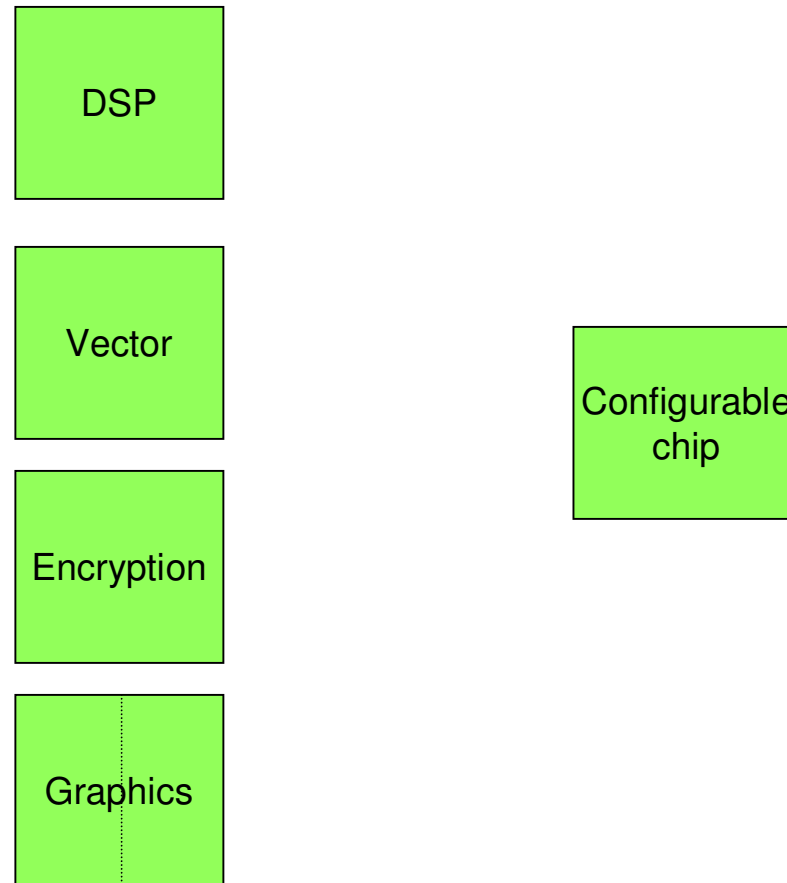
- These mechanisms enable
 - Merging multiple markets using a single design
 - Easier customization for a single market

Questions

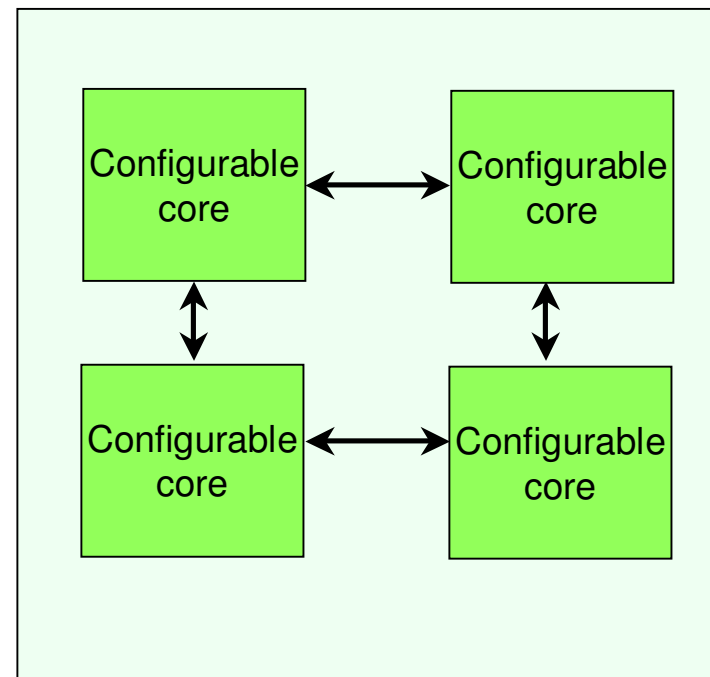
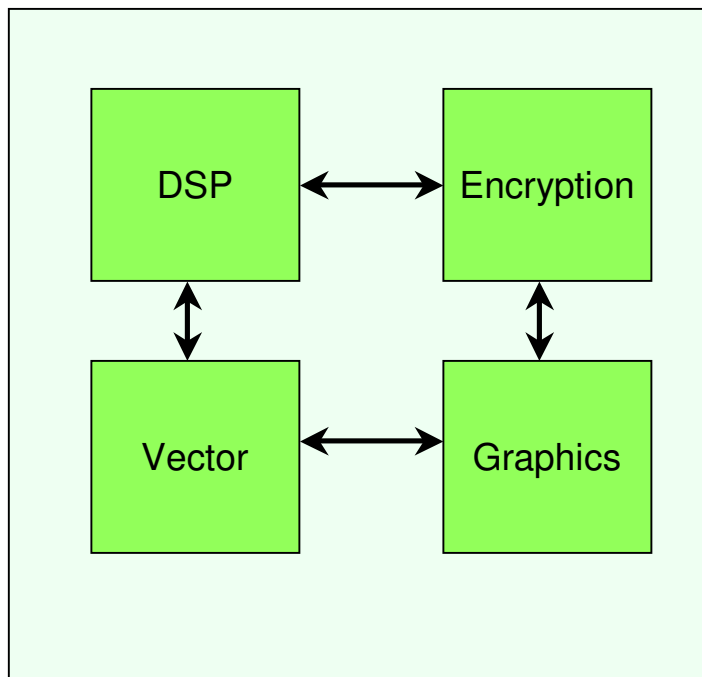
Future Directions

- Future directions
 - How universal? Design complexity and generality tradeoff
 - Applications outside DLP space
 - Loop intensive phases
 - Regular array accesses

Heterogeneous Architecture



Heterogeneous Architecture



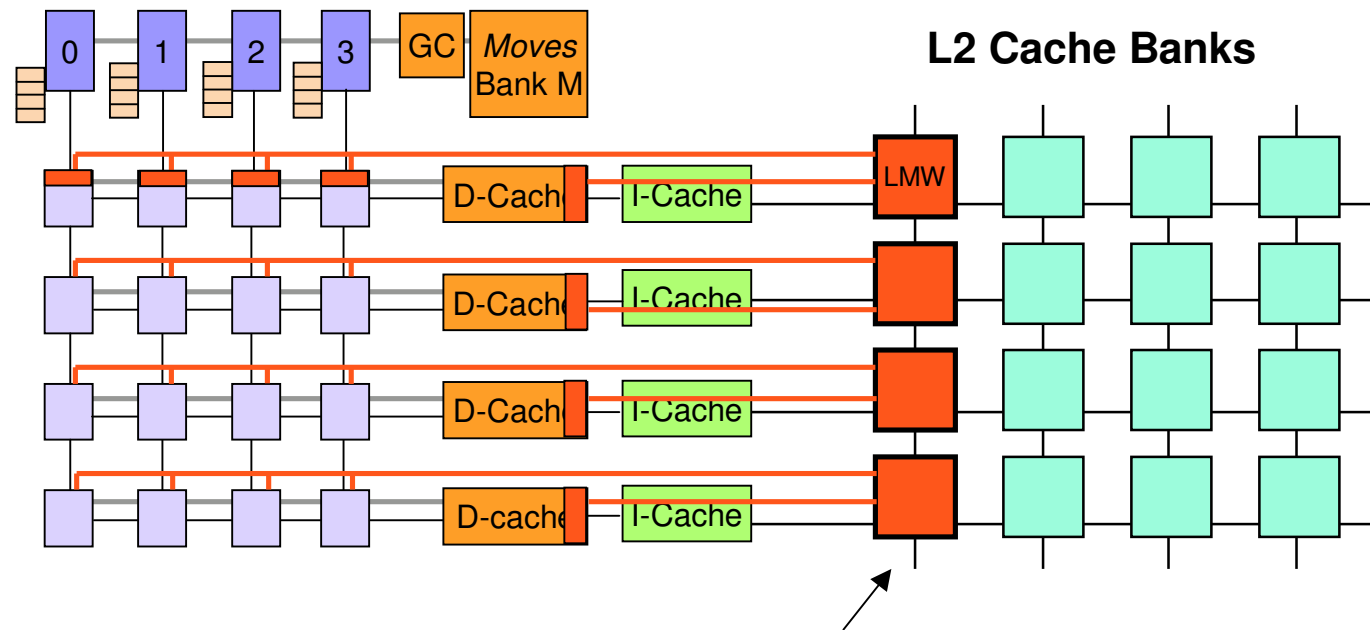
TRIPS Baseline Performance

Benchmark	IPC	Benchmark	IPC
convert	14.1	fragment-reflection	4
dct	10.4	fragment-simple	2.6
highpassfilter	7.4	vertex-reflection	5.2
fft	3.7	vertex-simple	3.6
lu	0.7	vertex-skinning	5.6
md5	2.8		
blowfish	5.1		
rijndael	7.5		

Benchmark Attributes

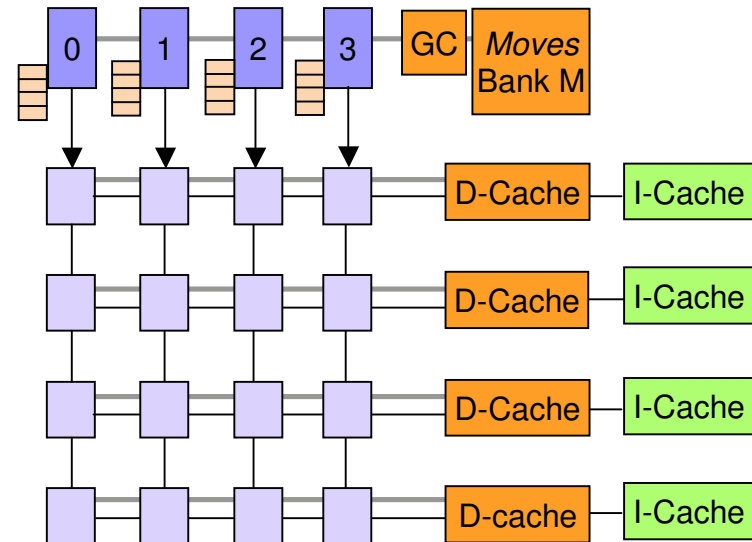
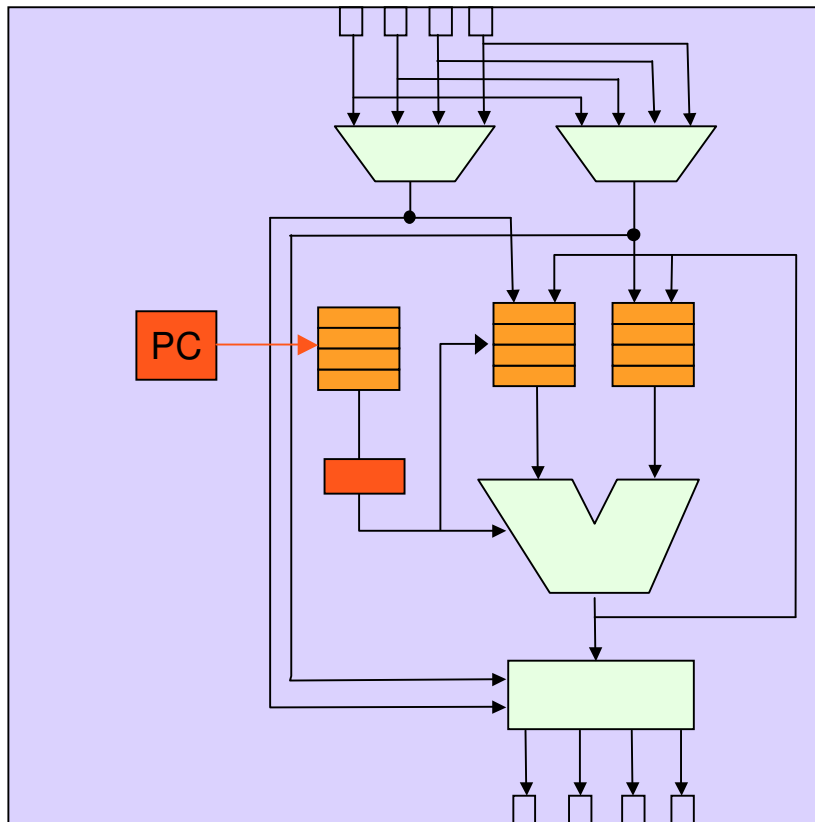
- Computation:
 - # of static instructions in loop body: 2 to about 1800
 - ILP: 1 to 7
- Memory
 - Diverse mix of all 4 types of accesses
 - # of regular accesses: 3 to 128
 - # of irregular accesses: 4 to 50
 - # of constants: 0 to 65
 - # of indexed constants: 128 to 1024
- Control:
 - Sequential: 9 of 14 benchmarks
 - Static loop bounds: 3 of 14 benchmarks
 - Data dependent branching: 2 of 14 benchmarks

Memory System Mechanisms



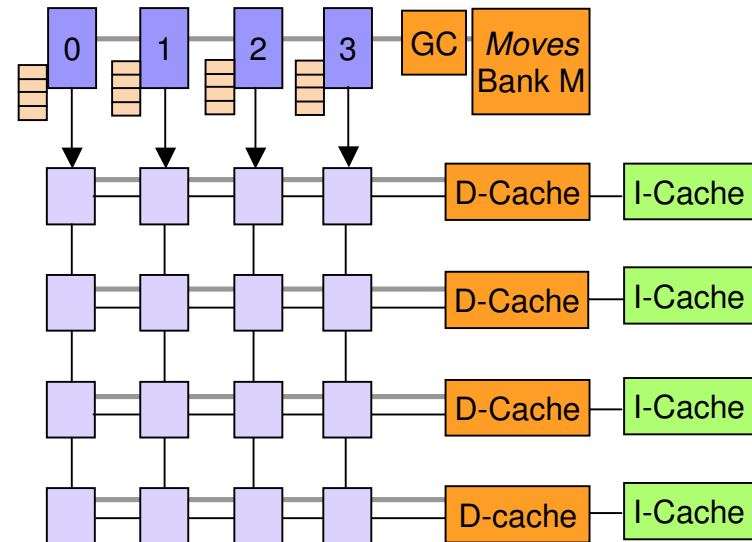
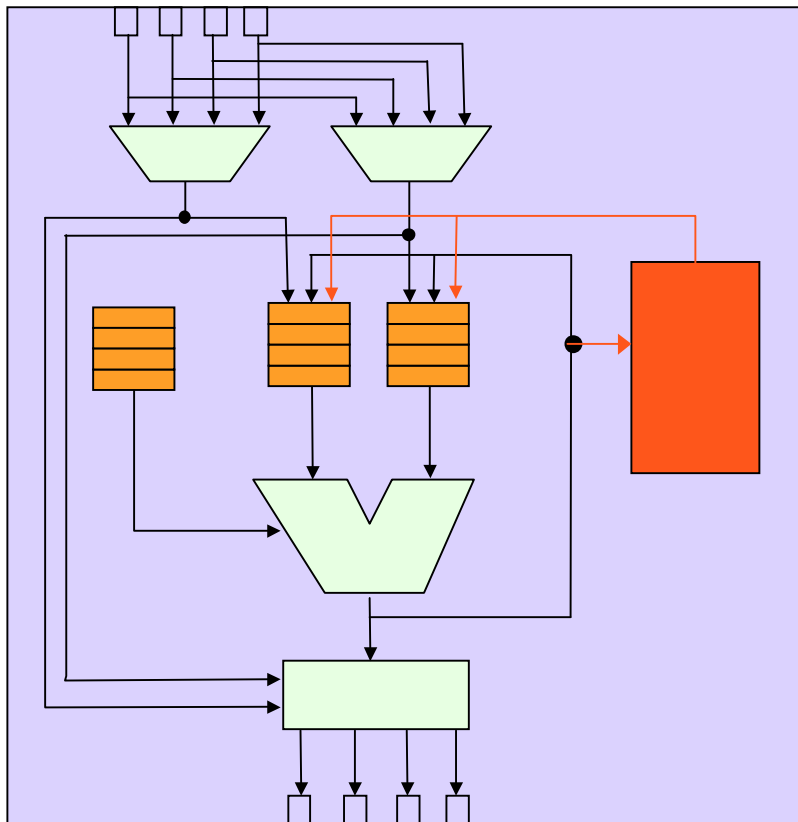
- Software managed cache
 - Programmable DMA engines at each bank
 - Exposed to compiler and programmer
- High bandwidth streaming channels
- Load multi-word and store buffers
- Cached L1-memory

I-Fetch and Control Mechanisms



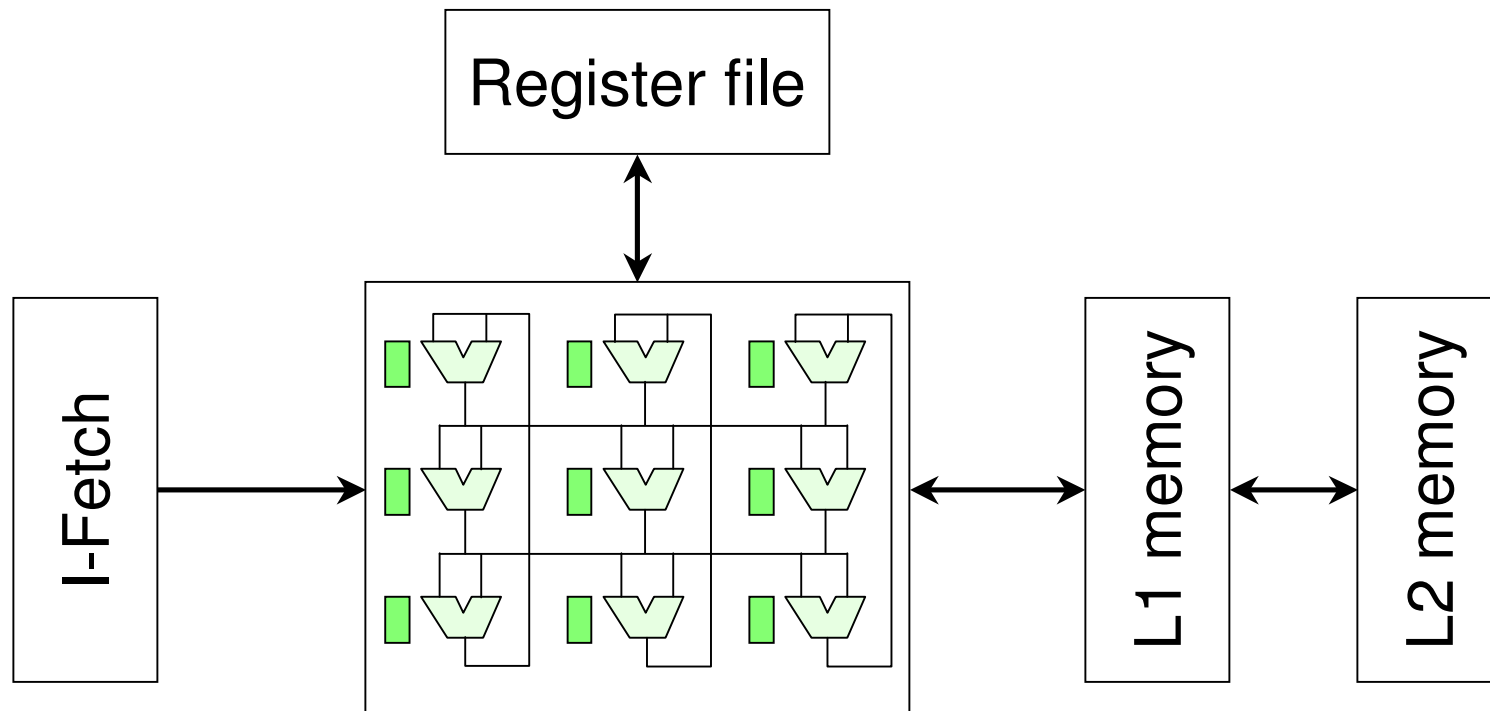
- Local PCs for data dependent branching

Execution Core Mechanisms



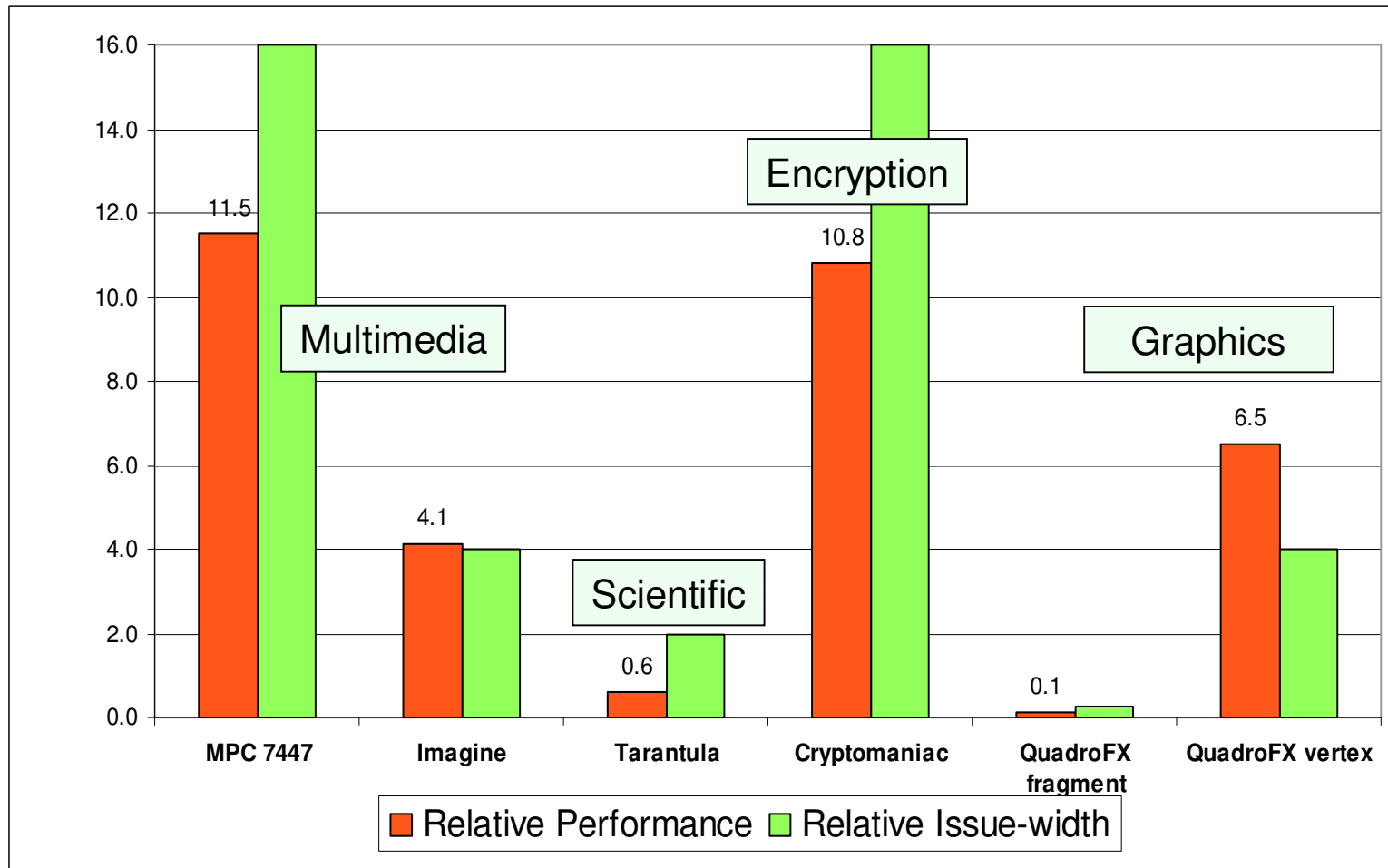
- L0 Data storage to provide lookup tables
- Operand revitalization for scalar constants

Data Parallel Architecture



- Execution substrate with many functional units
- Efficient inter-ALU communication to shuffle data
- Technology scalability

Comparison to Specialized Hardware



Future directions

- Application of mechanisms:
 - Dynamic tuning and selection of mechanisms when multiple classes of applications must be supported
 - Flexibility/simplicity trade-off when only a few applications need be supported
 - DLP behavior can be seen and exploited outside traditional DLP domains
- Evaluation of these mechanisms using cycle time, power, and area metrics
- Comparison of DLP mechanisms to a heterogeneous architecture

DLP Attributes and Mechanisms

Regular memory accesses

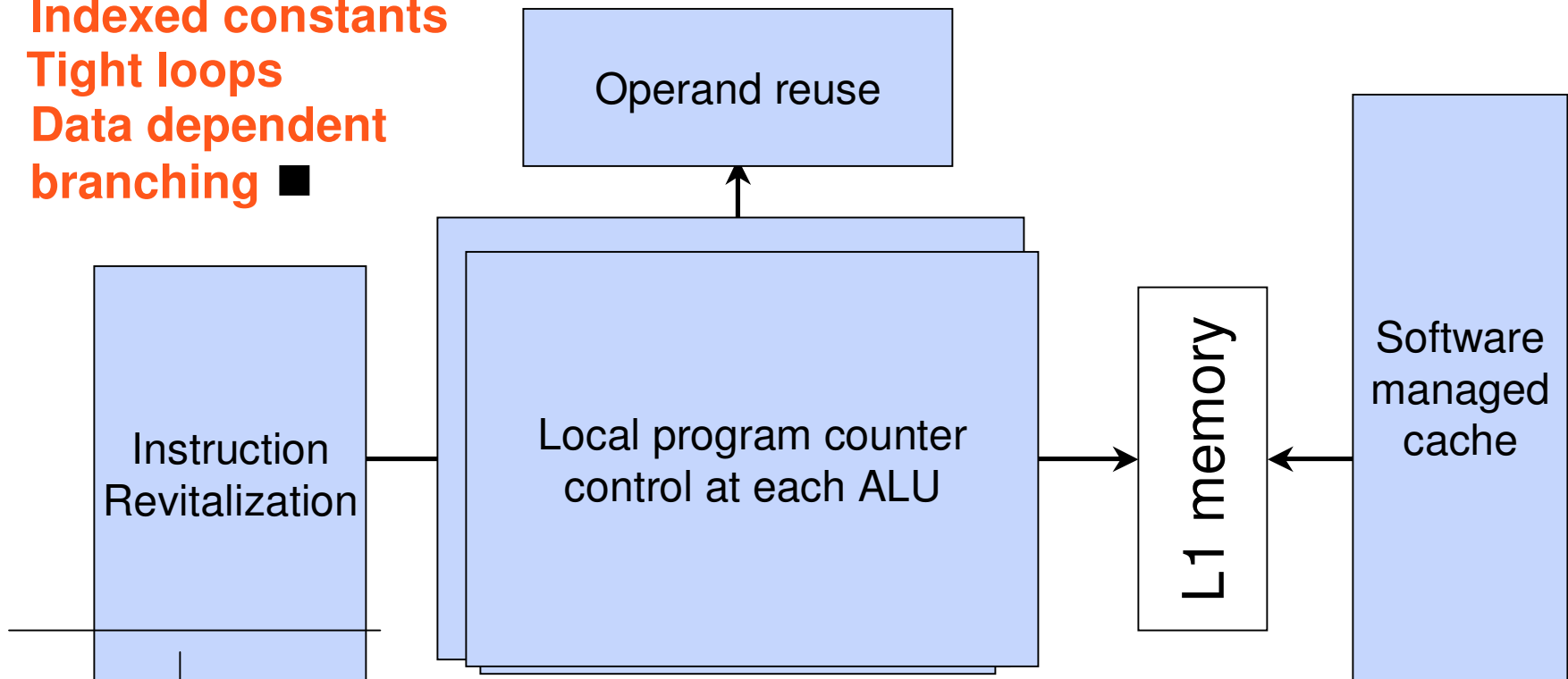
Scalar named constants

Indexed constants

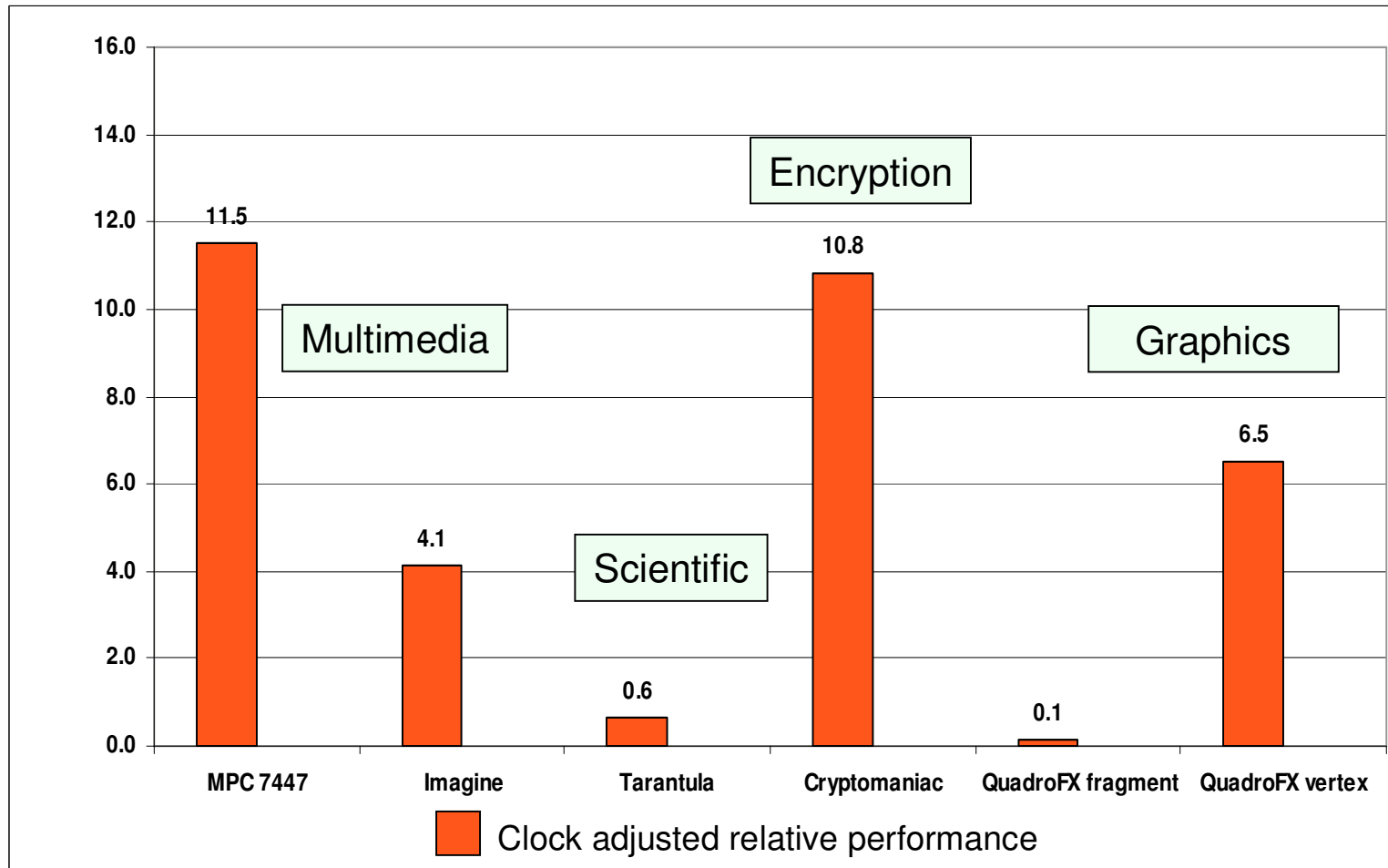
Tight loops

Data dependent

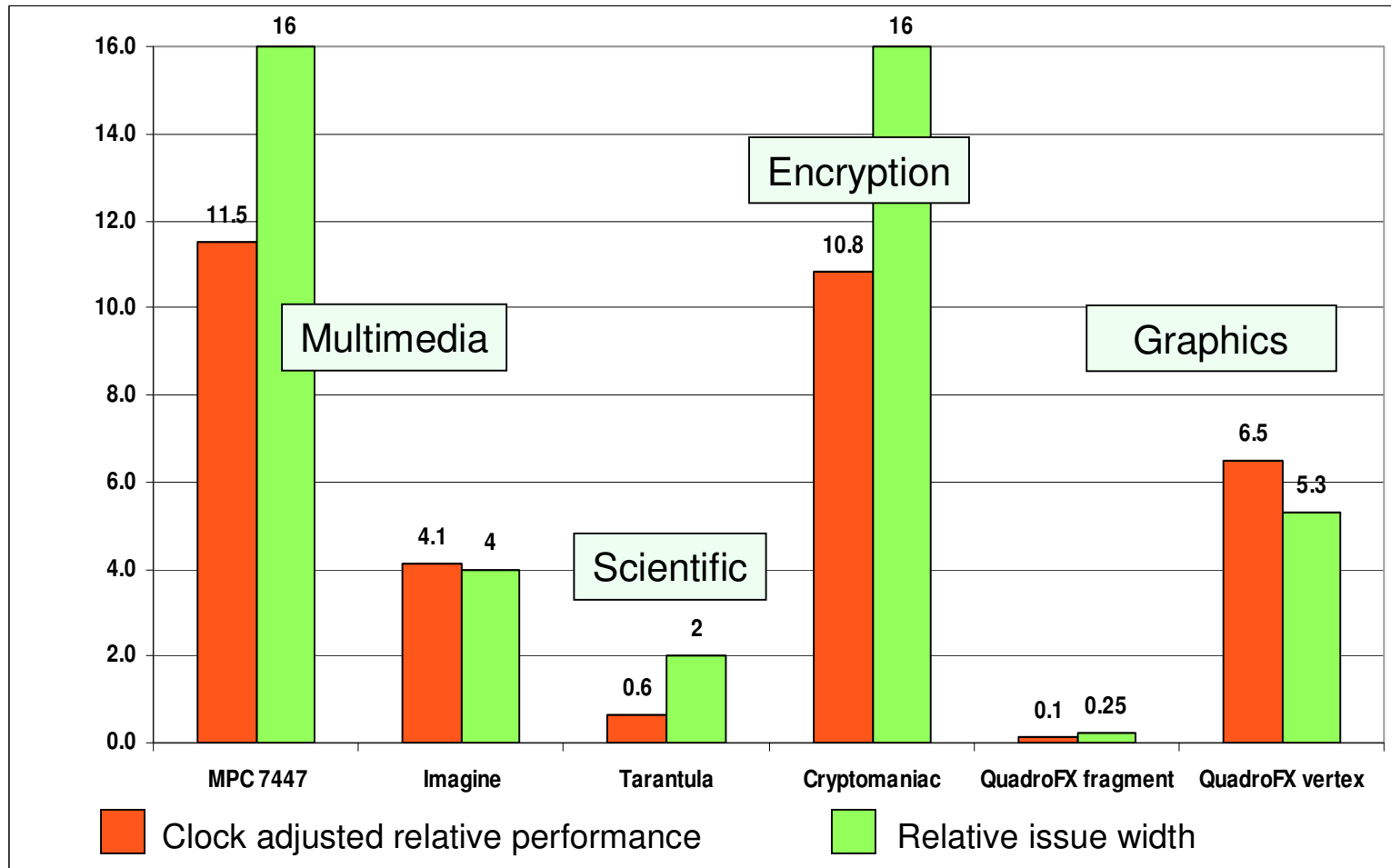
branching ■



Comparison to Specialized Hardware



Comparison to Specialized Hardware



Data-Parallel Applications and Architectures

- Performance
 - 8 Gflops in each Arithmetic Processor of Earth Simulator
 - 15 Gops required for software radios
 - 20 Gflops on GPU (32 programmable FP units on a single chip)
- Spread across diverse domains:
 - High performance computing (HPC), digital signal processing (DSP), real-time graphics, and encryption
- Conventional architecture models: Vector, SIMD, and MIMD
- Specialized narrowly focused hardware
 - MPEG4 decoding
 - DSPs have specialized error correction code units, convolution encoding etc.

Conclusion and Future Directions

- Key DLP program attributes identified:
 - Memory
 - Instruction control
 - Computation core
- Proposed complementary universal mechanisms
 - A single architecture can thus adapt to application
 - Resembling a vector, SIMD, or MIMD machine
 - Competitive performance compared to specialized processors
- Future directions
 - How universal? Design complexity and generality tradeoff
 - Applications outside DLP space
 - Loop intensive phases
 - Regular array accesses

Data-Parallel Applications and Architectures

- Similarities
 - High performance
 - Arithmetic Processor: 8 GFLOPS
 - Software radios: 15 GOPS
 - GPU: 20 GFLOPS (32 programmable FP units on chip)
 - Lots of concurrency
- Differences
 - Diverse domains
 - Types of concurrency
 - Different architecture models: Vector, SIMD, and MIMD
 - Specialized narrowly focused hardware
 - MPEG4 decoding
 - Specialized error correction code units, convolution encoding in DSPs