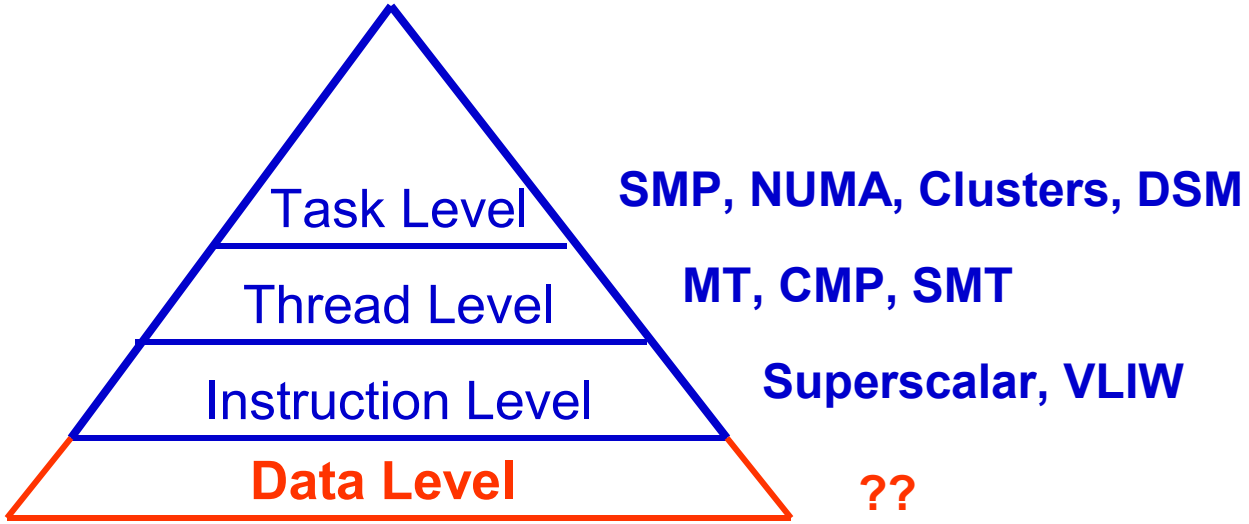
Scalable Vector Media-processors for Embedded Systems

Christoforos Kozyrakis

Ph.D. Dissertation Seminar

May 2nd, 2002

Exploiting Parallelism



- The main job of computer architects
  - Exploit parallelism to design efficient architectures
- Best practice
  - First pick the best solution at each level
  - Then make these solutions work well together
- How about data level parallelism ?
  - Can we do something efficient about it?

Multimedia Applications

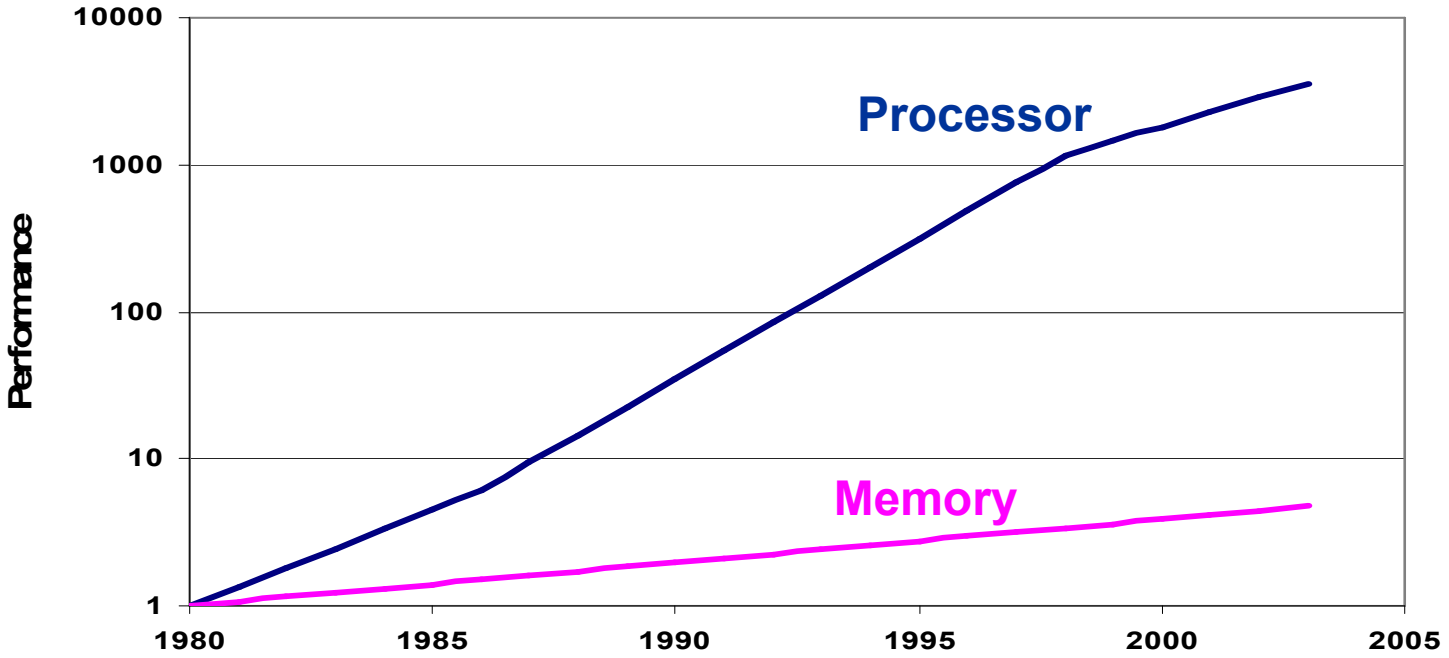
- The killer-app for current and future systems
  - 3D graphics, animation, speech and visual recognition, image and video processing, encryption
- Plenty of data-level parallelism
  - They repeat same function over sequences of data
  - Parallelism is explicit in the applications
- Characteristics to keep in mind
  - Need high performance with real-time response
  - Narrow data types (8 bit, 16 bit, 32 bit)
  - Streaming IO data with little temporal locality

Multimedia on Embedded Systems

- Embedded and portable systems
  - PDAs, set-top-boxes, game consoles, digital cameras, cellular phones
- Mobile-personal computing
  - E.g. PDA with speech recognition

- Realities of consumer electronics
  - Low cost
    - Small code size, low power consumption
  - Low energy consumption for portable devices
  - Short hardware and software development cycles
    - Processor should be easy to design, scale, and program
  - Integration is often the key for new applications

Technology Constraints



- Processor-memory performance gap
- Latency scaling of very long wires
- Exponentially increasing design and verification complexity

Thesis

- It is possible to design efficient microprocessors for embedded multimedia systems with
  - High performance
  - Low energy/power consumption
  - Low design complexity
  - High scalability

- Basic arguments
  - A vector architecture can exploit efficiently the data-level parallelism in multimedia applications
  - We can design on-chip, cost-effective memory systems that provide high bandwidth

Outline

- Motivation and thesis
- The VIRAM instruction set for multimedia processing
  [David Martin, Krste Asanovic, Dave Judd, Rich Fromm]
- The VIRAM-1 vector microarchitecture and prototype media-processor
  [Sam Williams, Joe Gebis]
- The CODE vector microarchitecture
- Conclusions and future work

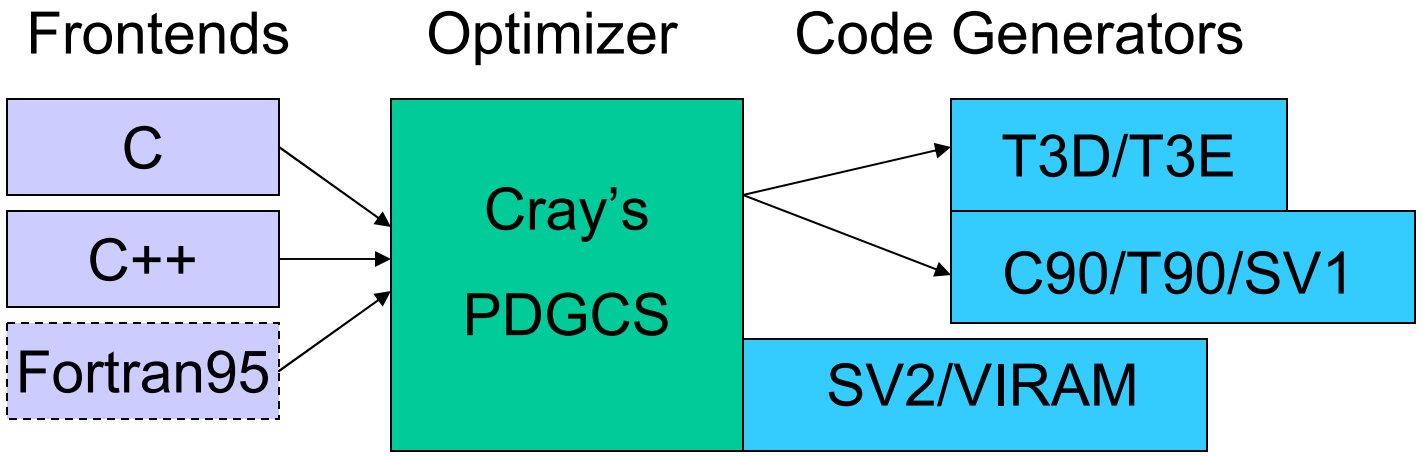
VIRAM Instruction Set

- Vector load-store instruction set
  - Coprocessor extension to MIPS-64 ISA
- Architecture state
  - 32 general-purpose vector registers
  - 16 flag registers
  - Scalar registers for control, addresses, strides, etc
- Vector instructions
  - Arithmetic: integer, floating-point, logical
  - Load-store: unit-stride, strided, indexed
  - Misc: vector processing (pack/unpack), flag processing (pop count)
  - 90 unique instructions, 660 opcodes

VIRAM ISA Enhancements

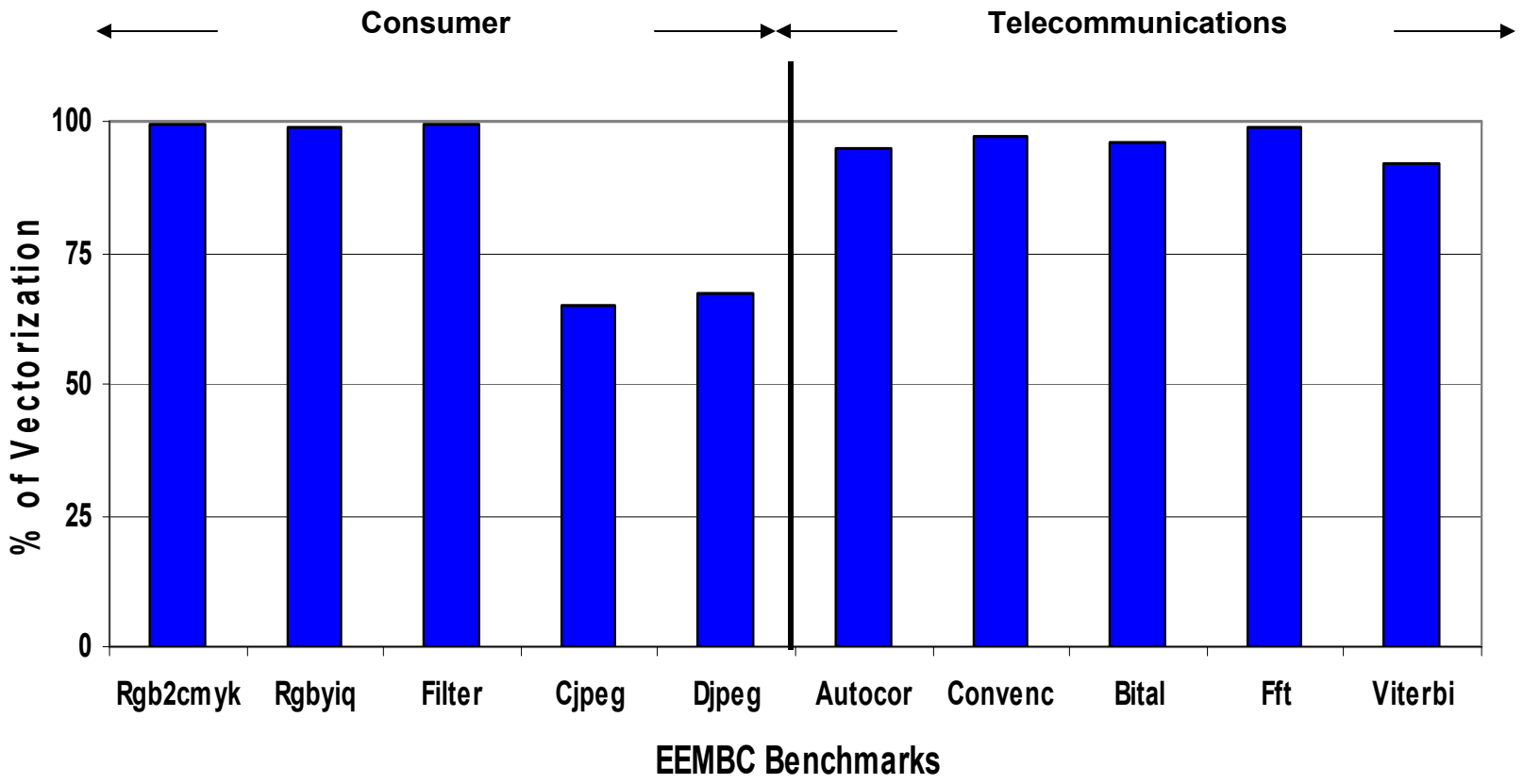
- Multimedia processing
  - Support for multiple data-types (64b/32b/16b)
    - Element/operation width specified with control register
  - Saturated and fixed-point arithmetic
    - Flexible multiply-add model without accumulators
  - Simple element permutations for reductions and FFTs
  - Conditional execution using the flag registers
- General-purpose systems
  - TLB-based virtual memory
    - Separate TLB for vector memory accesses
  - Hardware support for reduced context switch overhead
    - Valid/dirty bits for vector registers
    - Support for “lazy” save/restore of vector state

VIRAM Compiler



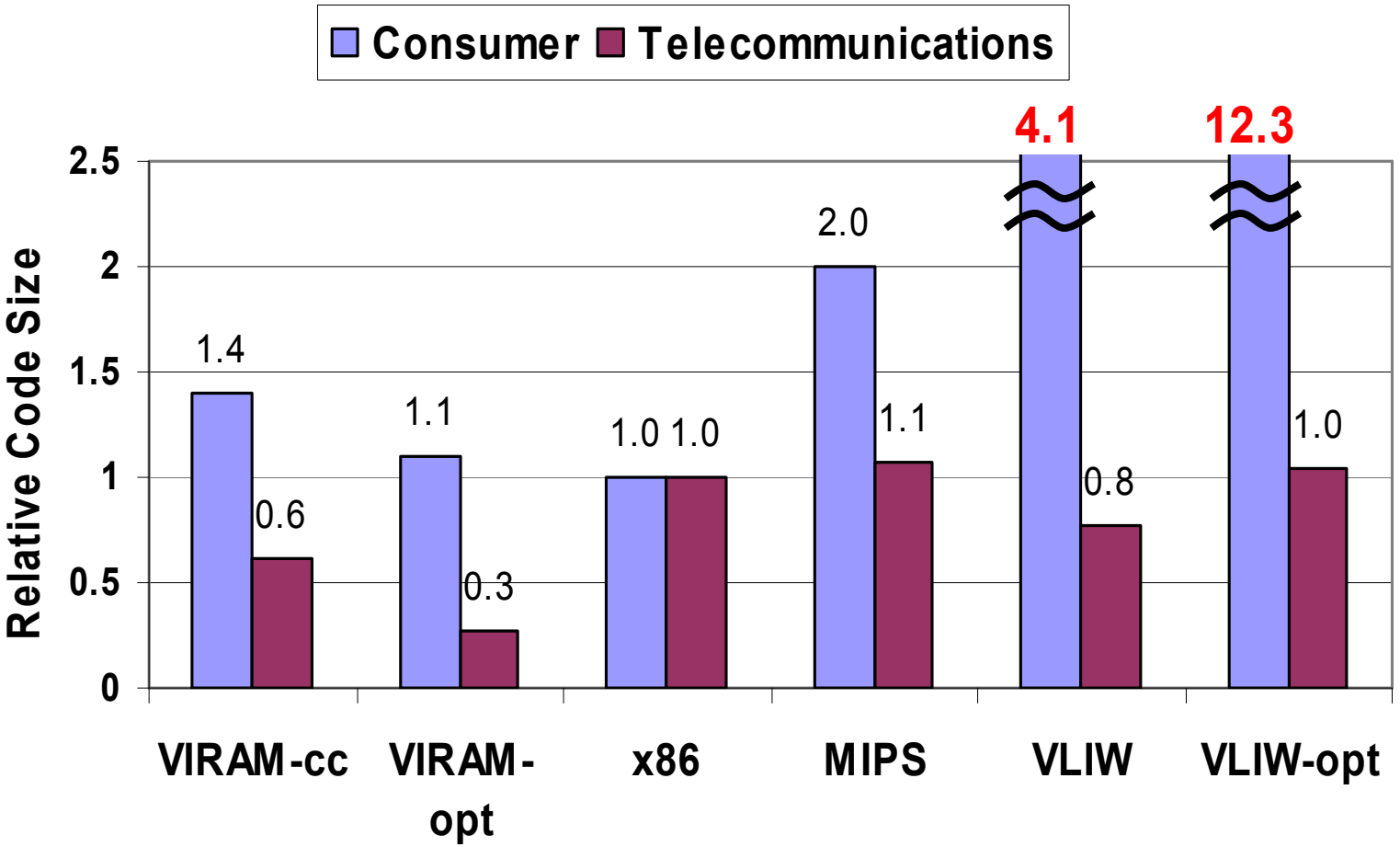
- Based on Cray PDGCS compiler
  - Extensive vectorization capabilities including outer-loop
  - Automatic vectorization of narrow operations and reductions
  - Lacks back-end optimizations
    - Code motion, basic block scheduling

EEMBC: Vectorization



- % of dynamic operations specified by vector instructions

EEMBC: Code Size

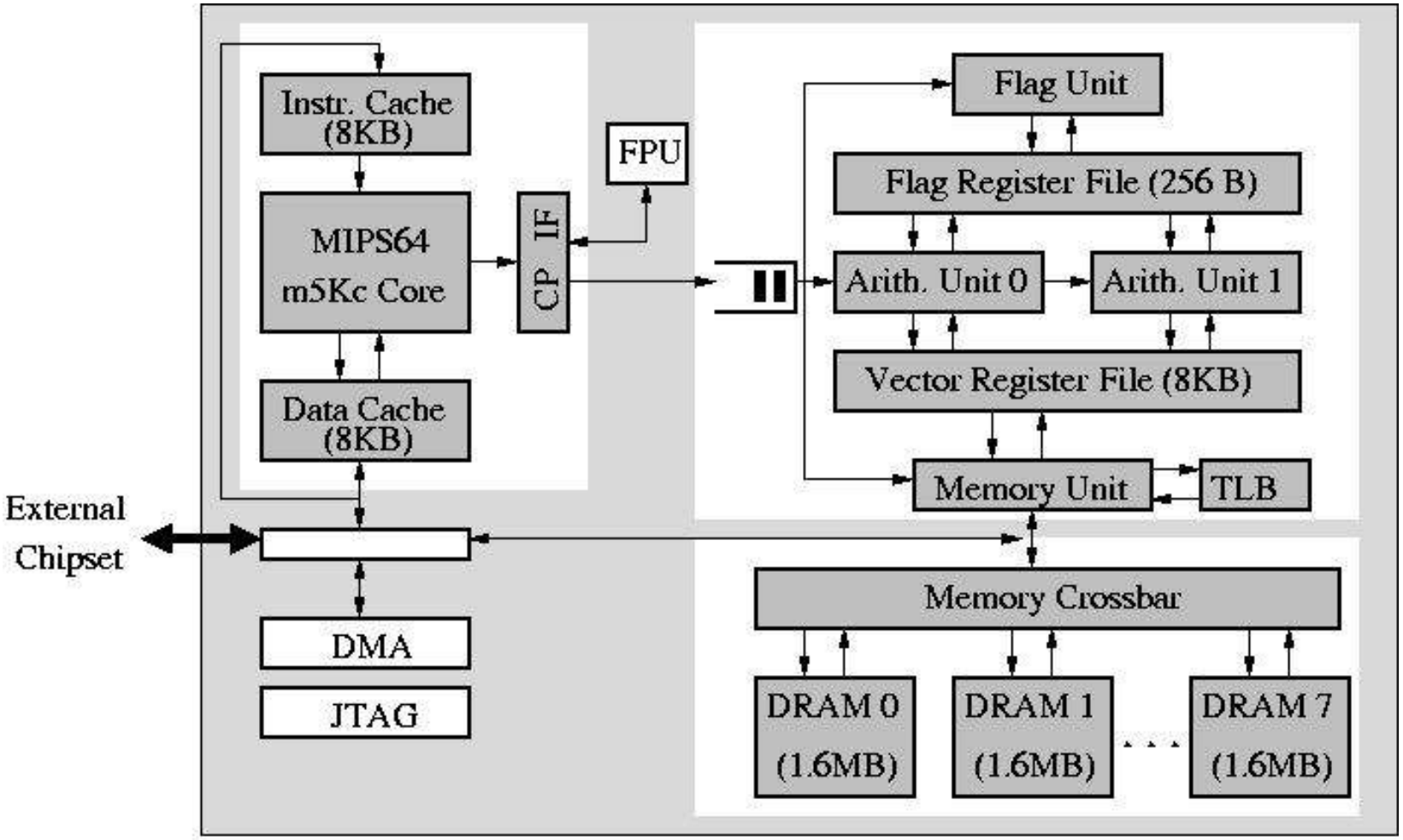


- Average code size, normalized to the x86 architecture

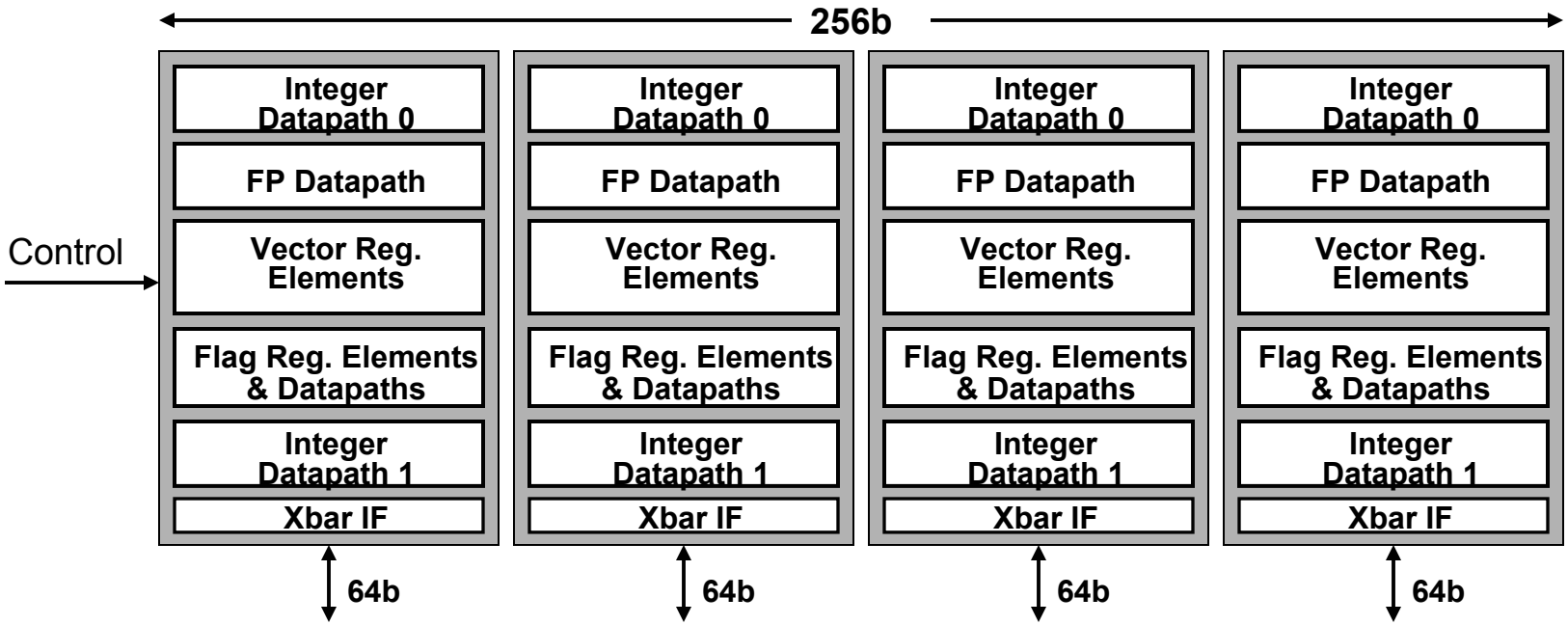
Outline

- Motivation and thesis
- The VIRAM instruction set for multimedia processing
- The VIRAM-1 vector microarchitecture and prototype media-processor
- The CODE vector microarchitecture
- Conclusions and future work

VIRAM-1 Block Diagram

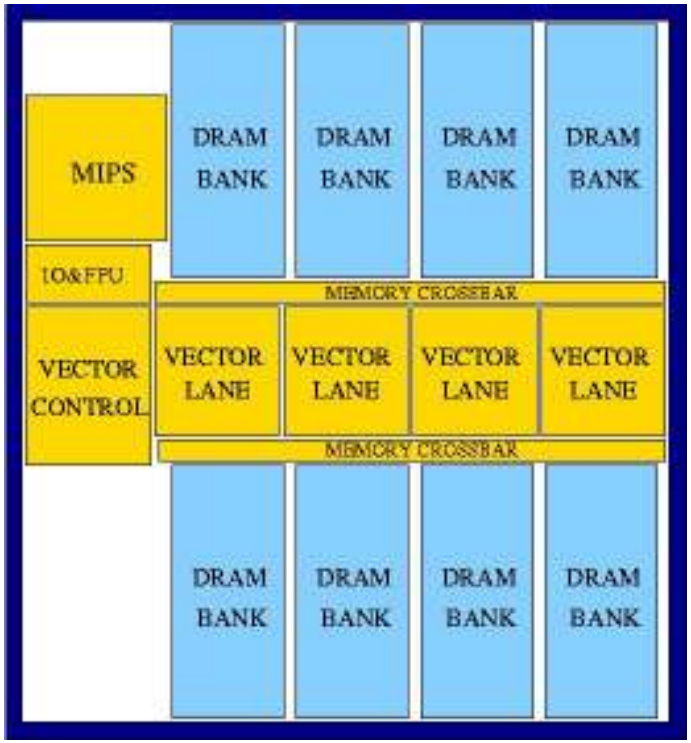


Modular Vector Unit Design



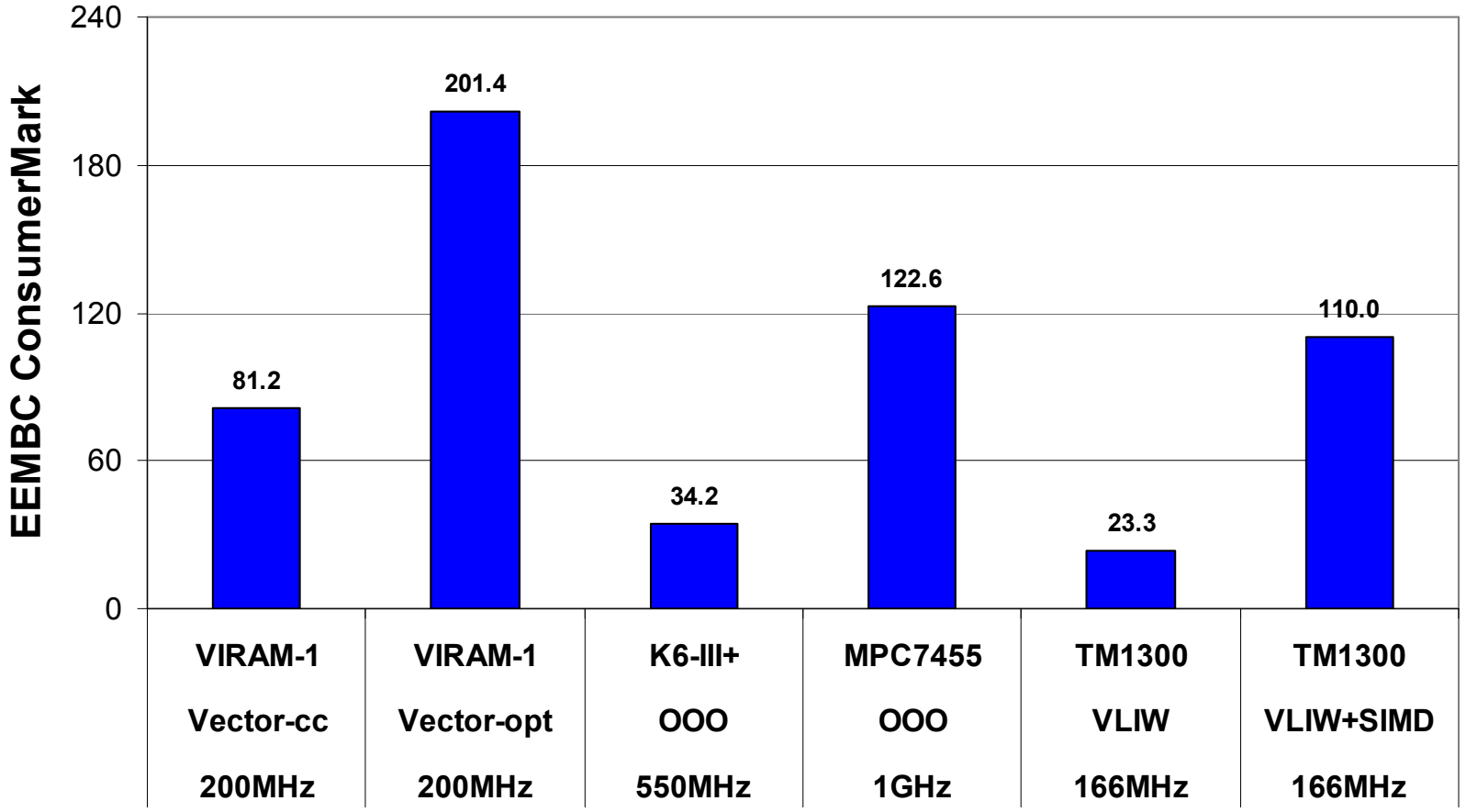
- Single 64b "lane" design replicated multiple times
  - Reduces design and testing time
  - Provides a simple scaling model (up or down) without major control or datapath redesign
- Most instructions require only intra-lane interconnect
  - Tolerance to interconnect delay scaling

VIRAM-1 Chips Statistics

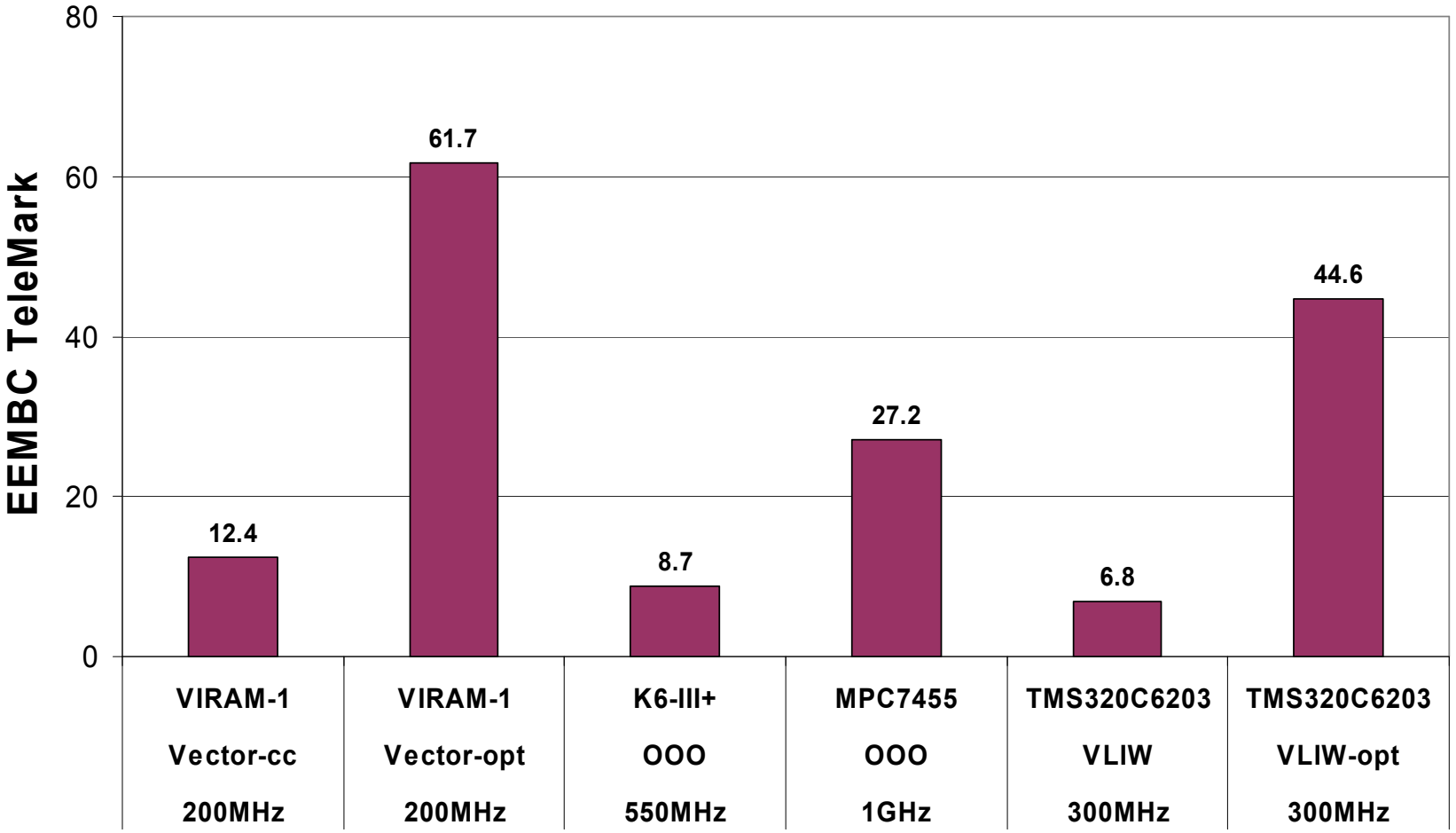


- Technology: 0.18μm CMOS from IBM
  - 6 layers copper, trench DRAM cell
- 335 mm^2 die area
  - 140mm^2 DRAM, 70 mm^2 logic
  - 10mm^2 per vector lane
- 120M transistors
  - 112.5M DRAM, 7.5M logic
- 200 MHz, 2 Watts
- Peak vector performance
  - Integer: 1.6/3.2/6.4 Gop/s (64b/32b/16b)
  - Fixed-point: 2.4/4.8/9.6 Gop/s (64b/32b/16b)
  - Floating-point: 1.6 Gflop/s (32b)

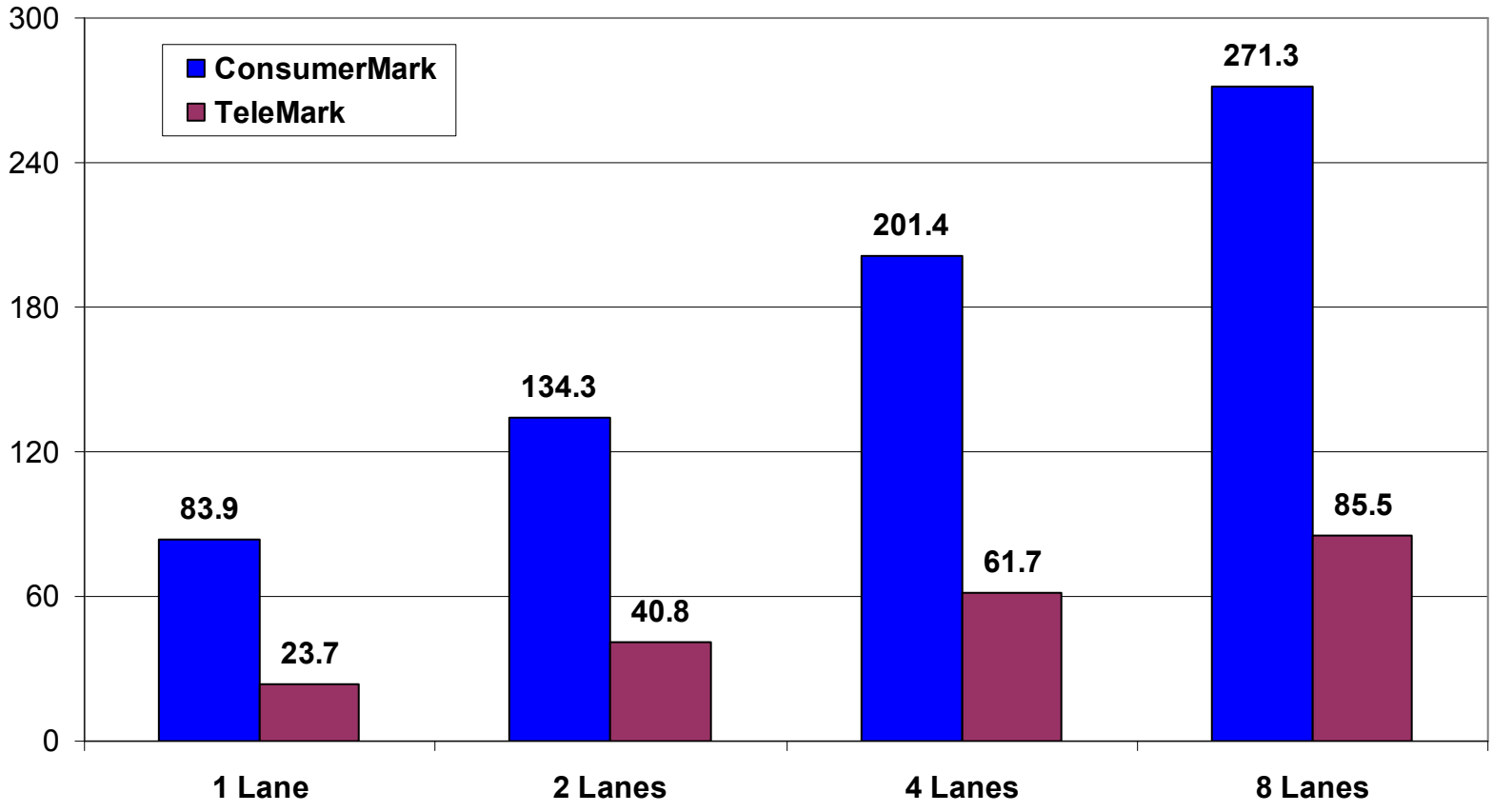
EEMBC: Consumer



EEMBC: Telecommunications



EEMBC: Scalability



- Same clock frequency (200 MHz), same memory system

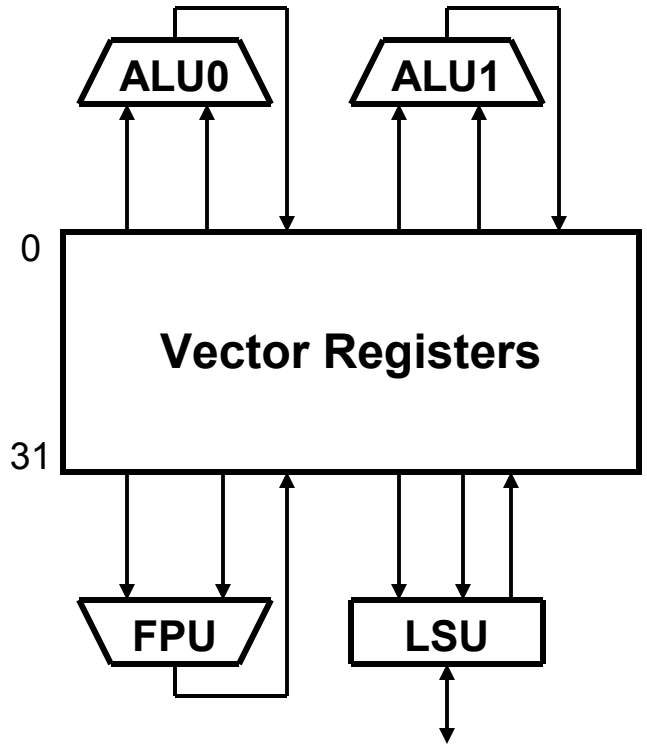
Outline

- Motivation and thesis
- The VIRAM instruction set for multimedia processing
- The VIRAM-1 vector microarchitecture and prototype media-processor
- The CODE vector microarchitecture
- Conclusions and future work

The CODE Microarchitecture

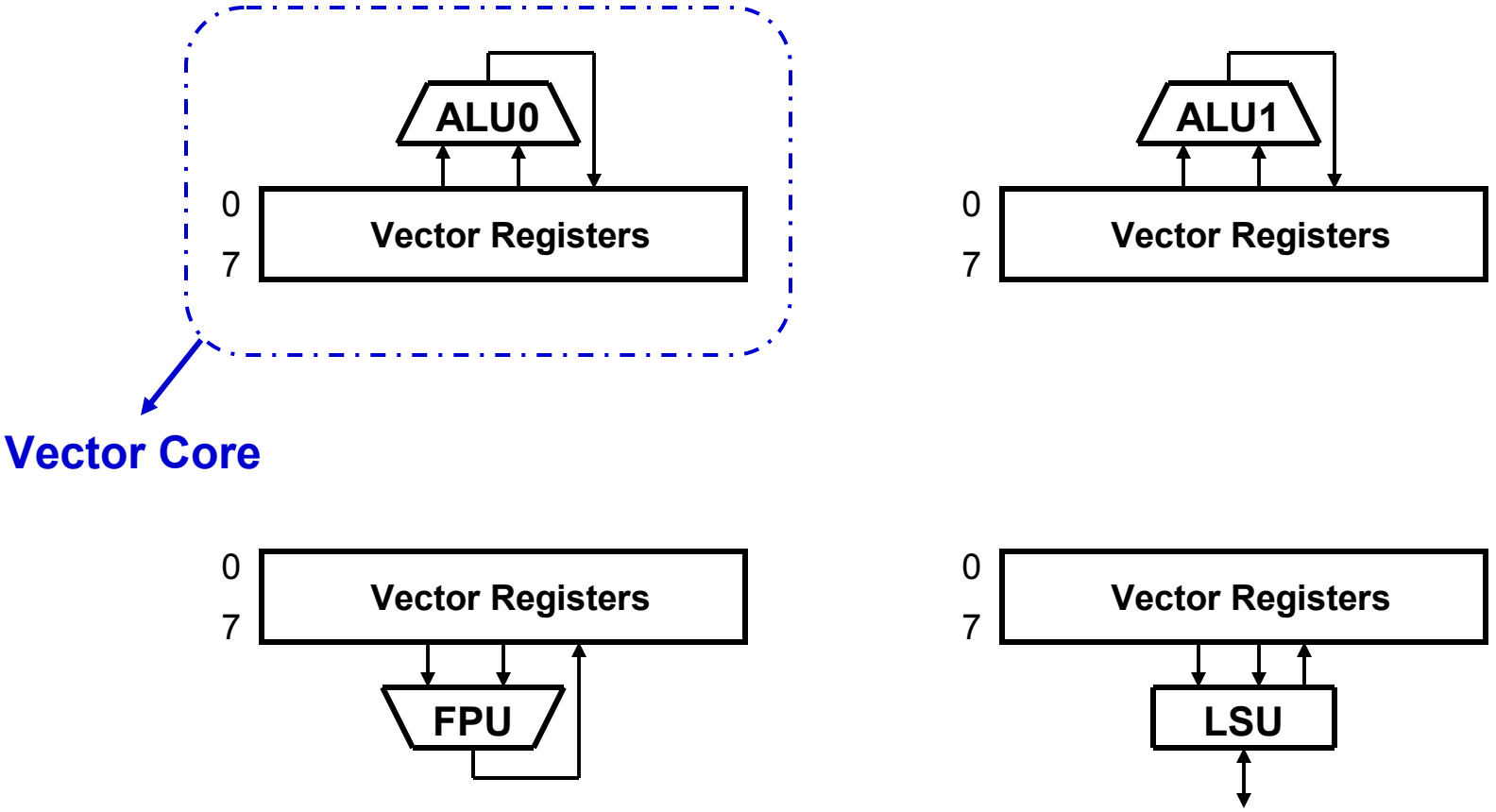
- Goal: improve on VIRAM-1
  - Simplify the vector register file design
    - Reduce the number of access ports per register
    - Allow for more functional units per lane
  - Tolerate higher memory latency
    - Allow for higher clock frequency or slower processor-memory interconnect
- Approach: reorganize vector lanes
  - **Co**mposite organization
    - Assign a small vector register file to each functional unit
  - **De**coupled execution
    - Decouple instruction execution in each functional unit using instruction and data queues

From VIRAM-1 to CODE (1)



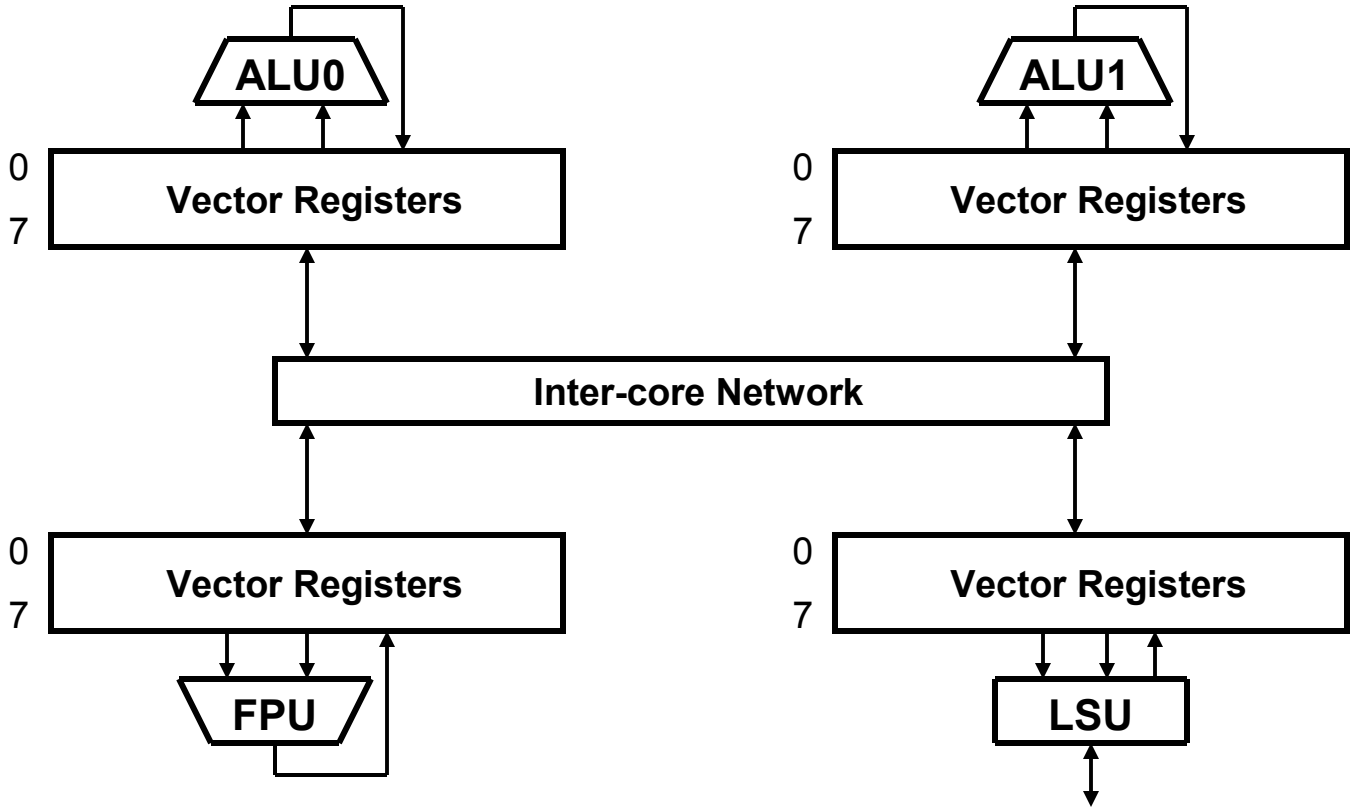
- Vector lane in VIRAM-1
- Centralized vector register file feeds all FUs
- Deeply pipelined functional units

From VIRAM-1 to CODE (2)



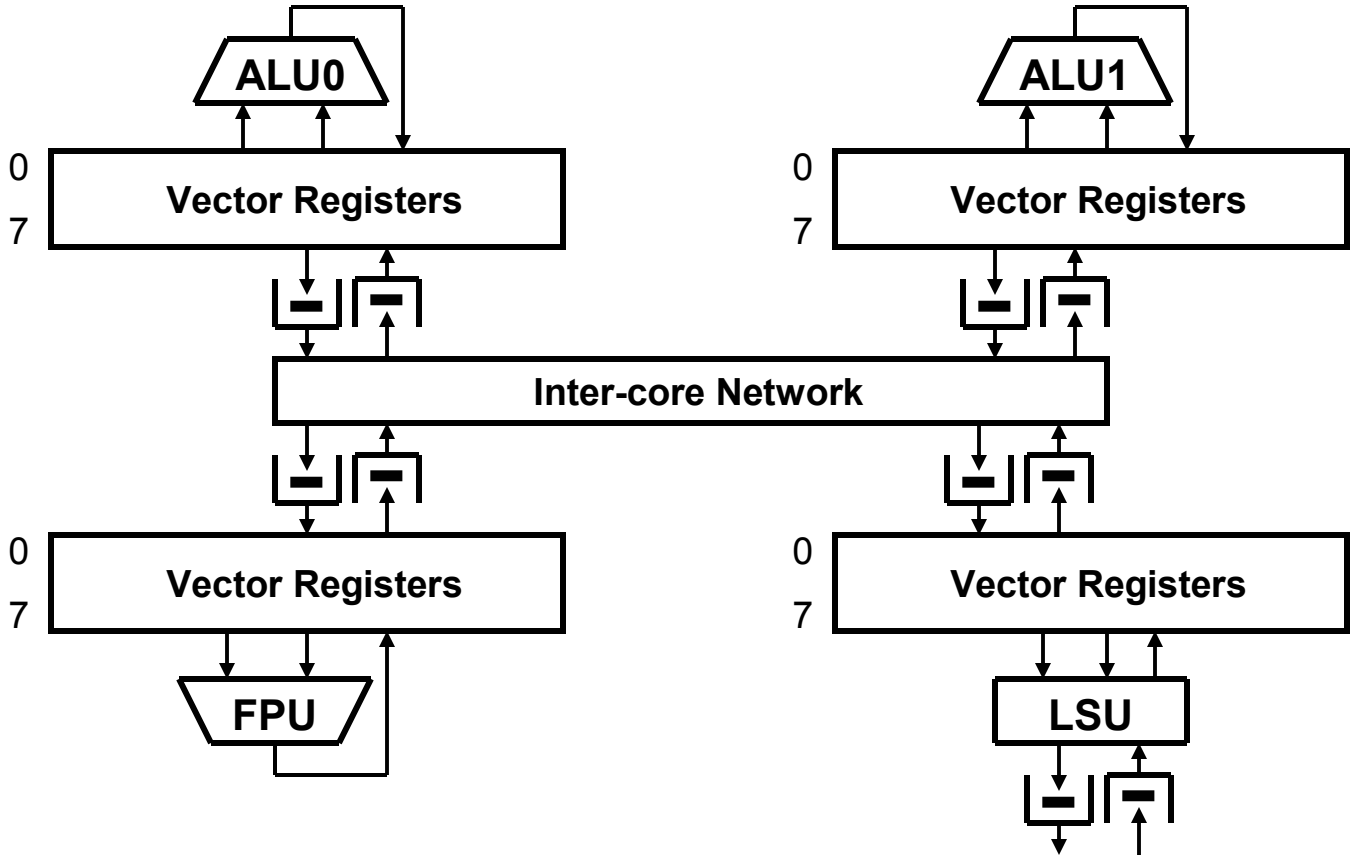
- Associate a few vector registers with each FU
- Local register file has fixed number of access ports

From VIRAM-1 to CODE (3)



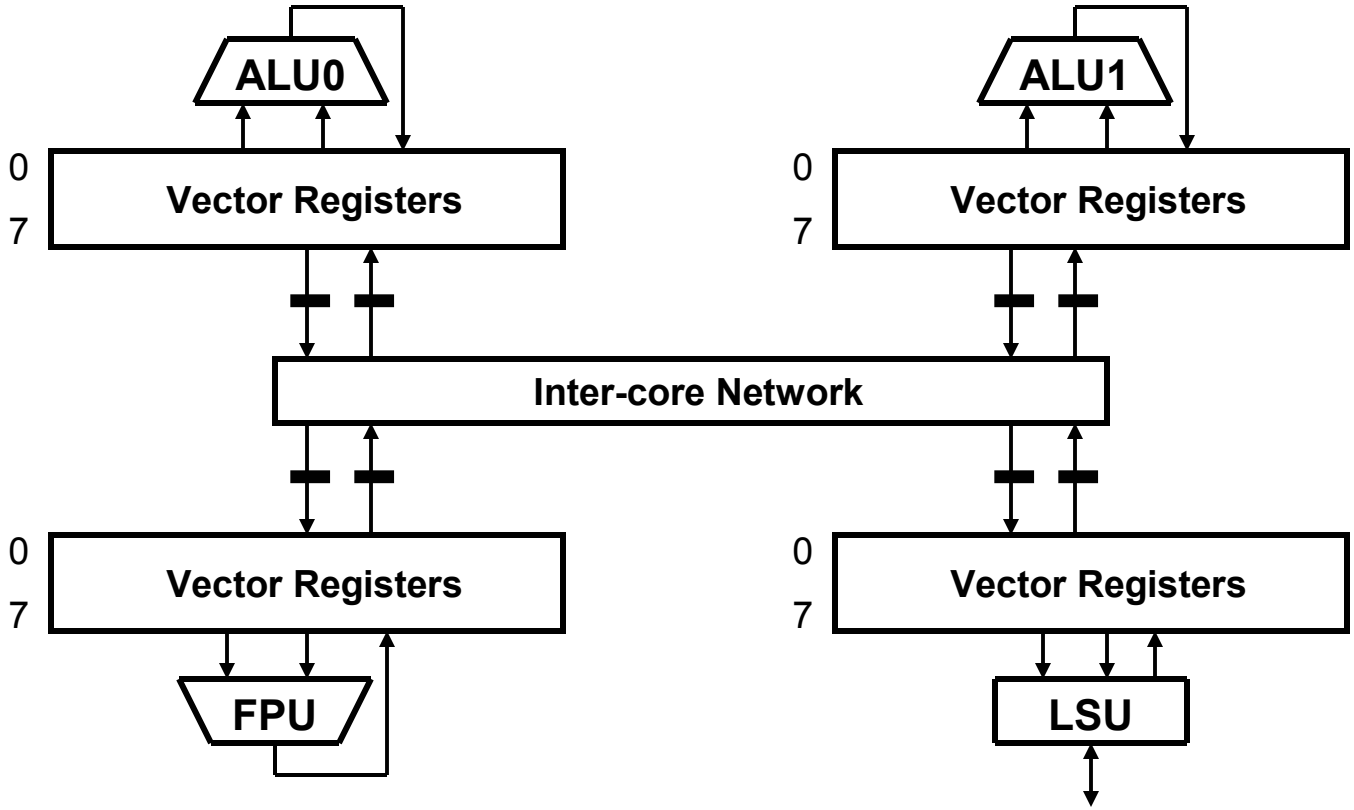
- Network for inter-core vector register transfers
- It can be a bus, a ring, a crossbar, etc…

From VIRAM-1 to CODE (4)



- Data and instruction queues for decoupling
- Decoupling of both memory & inter-core transfers

From VIRAM-1 to CODE (5)

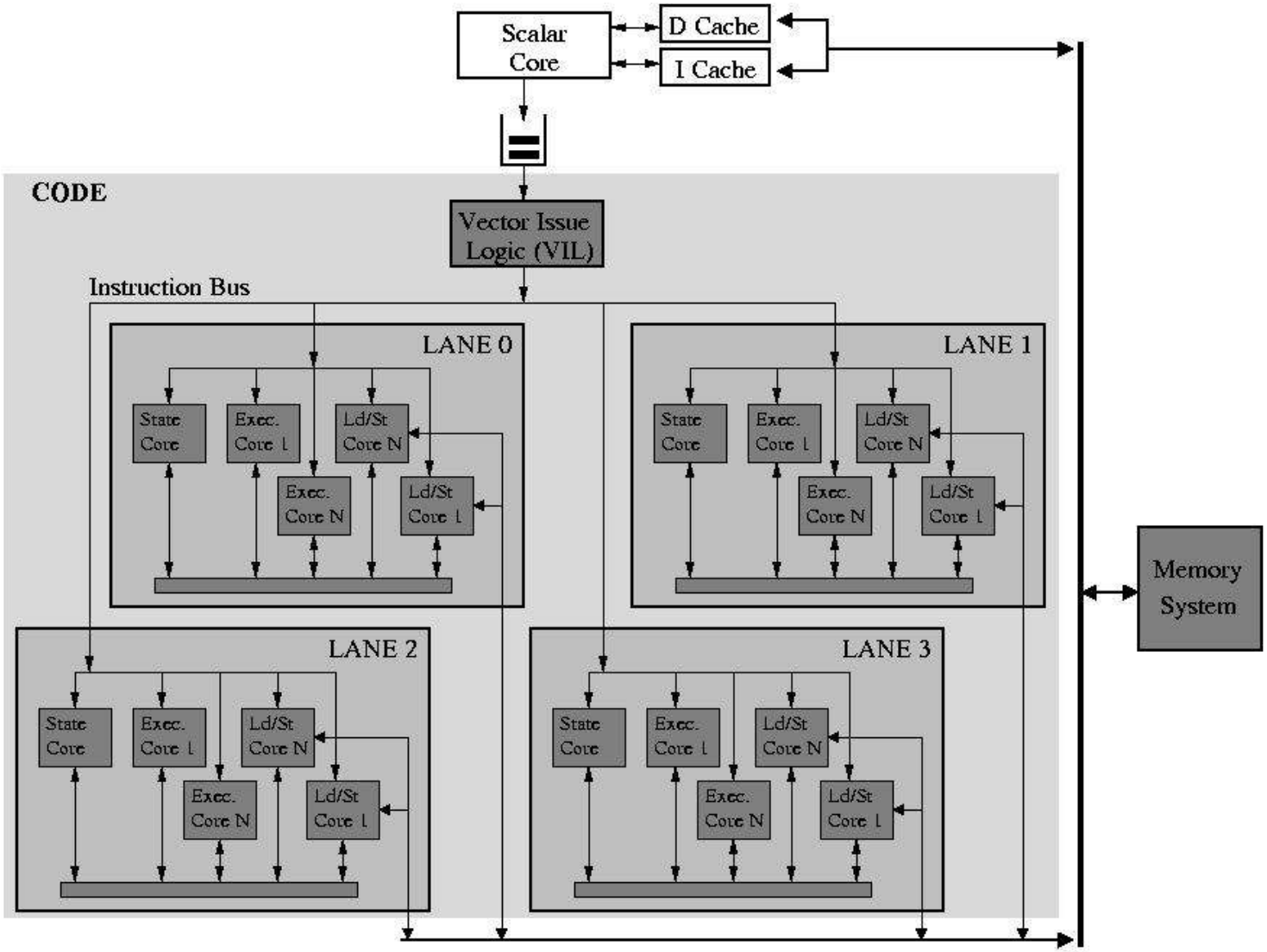


- Use vector registers for data queues
- Saves area, simplifies design and control

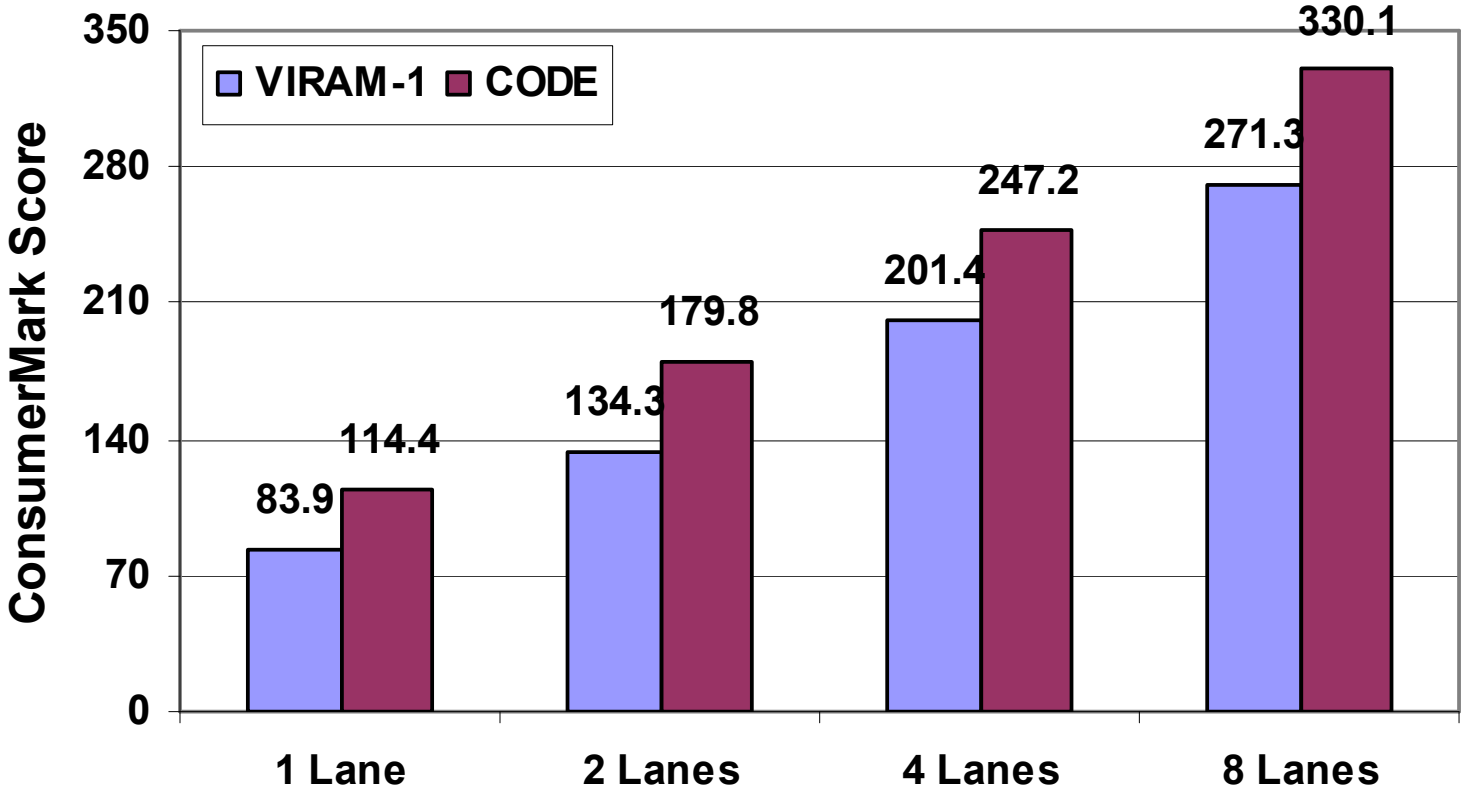
CODE Issue Logic

- Operation
  - Issue instructions to vector cores
  - Allocate core registers for instruction operands
  - Indicate necessary inter-core register transfers
- Data structures
  - Renaming table
    - Maintains the physical location for each architectural registers
  - A free-list for the local vector registers in each core
- It is very simple because
  - It handles one vector instruction per cycle
  - Everything is in-order

CODE Block Diagram

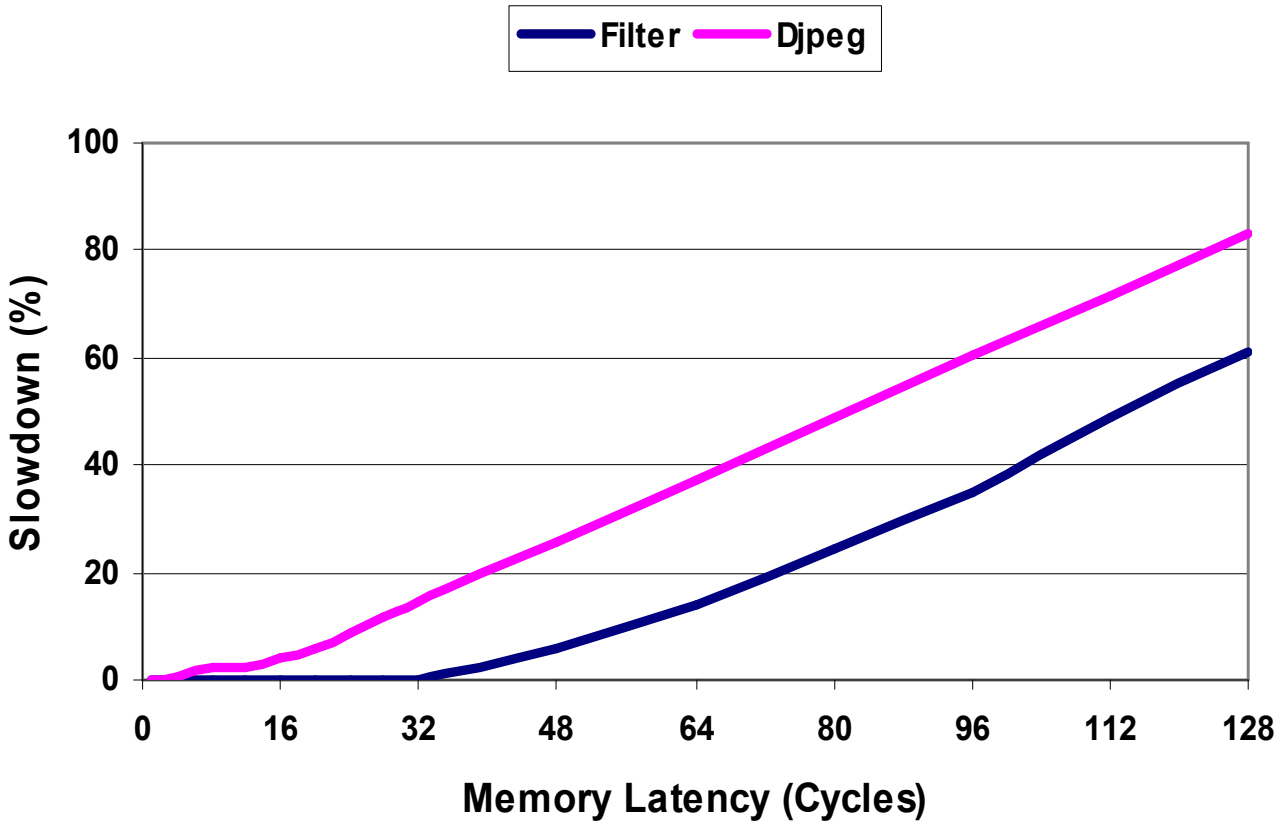


CODE vs. VIRAM-1



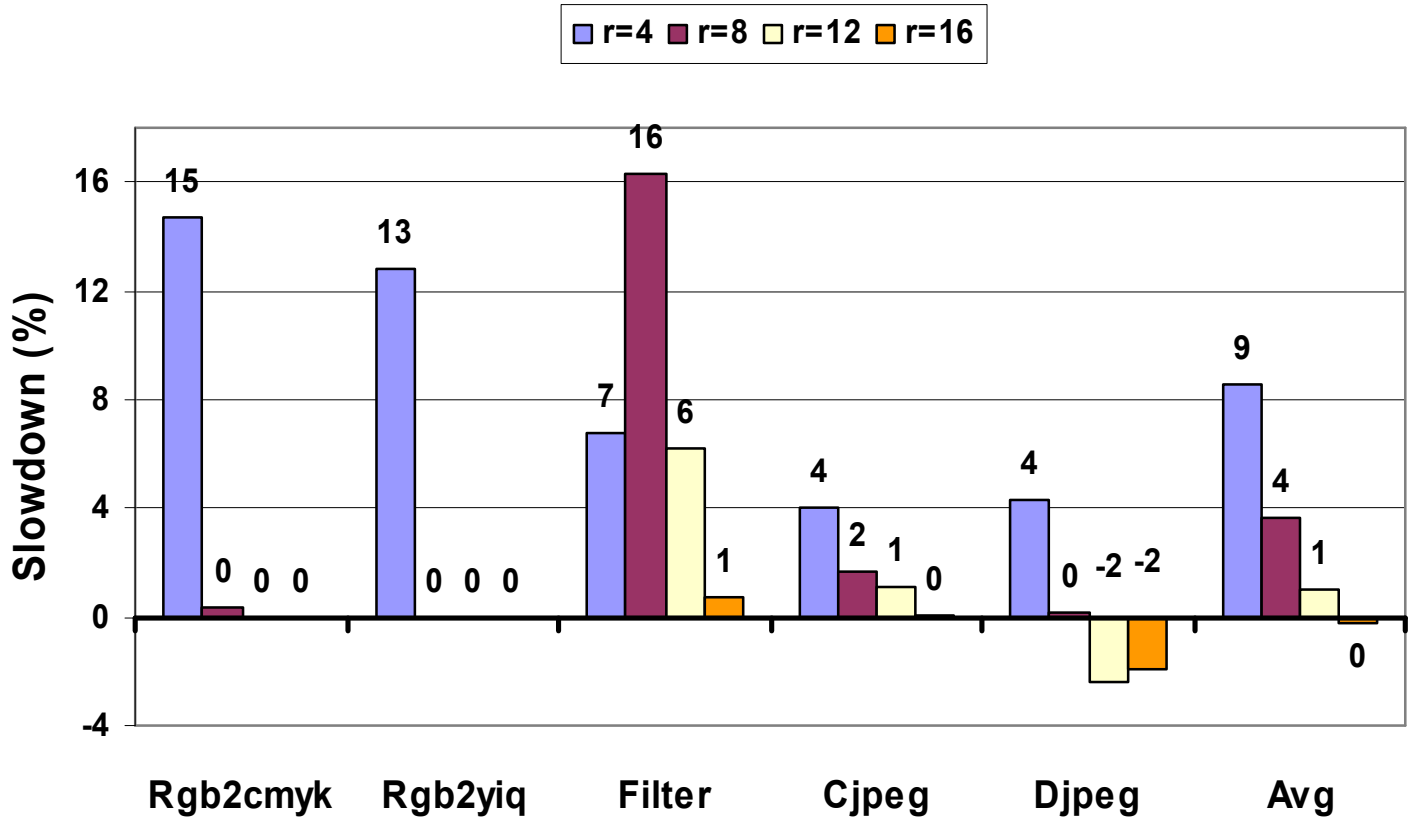
- Same area, same clock frequency (200 MHz), same memory system

CODE: Latency Tolerance



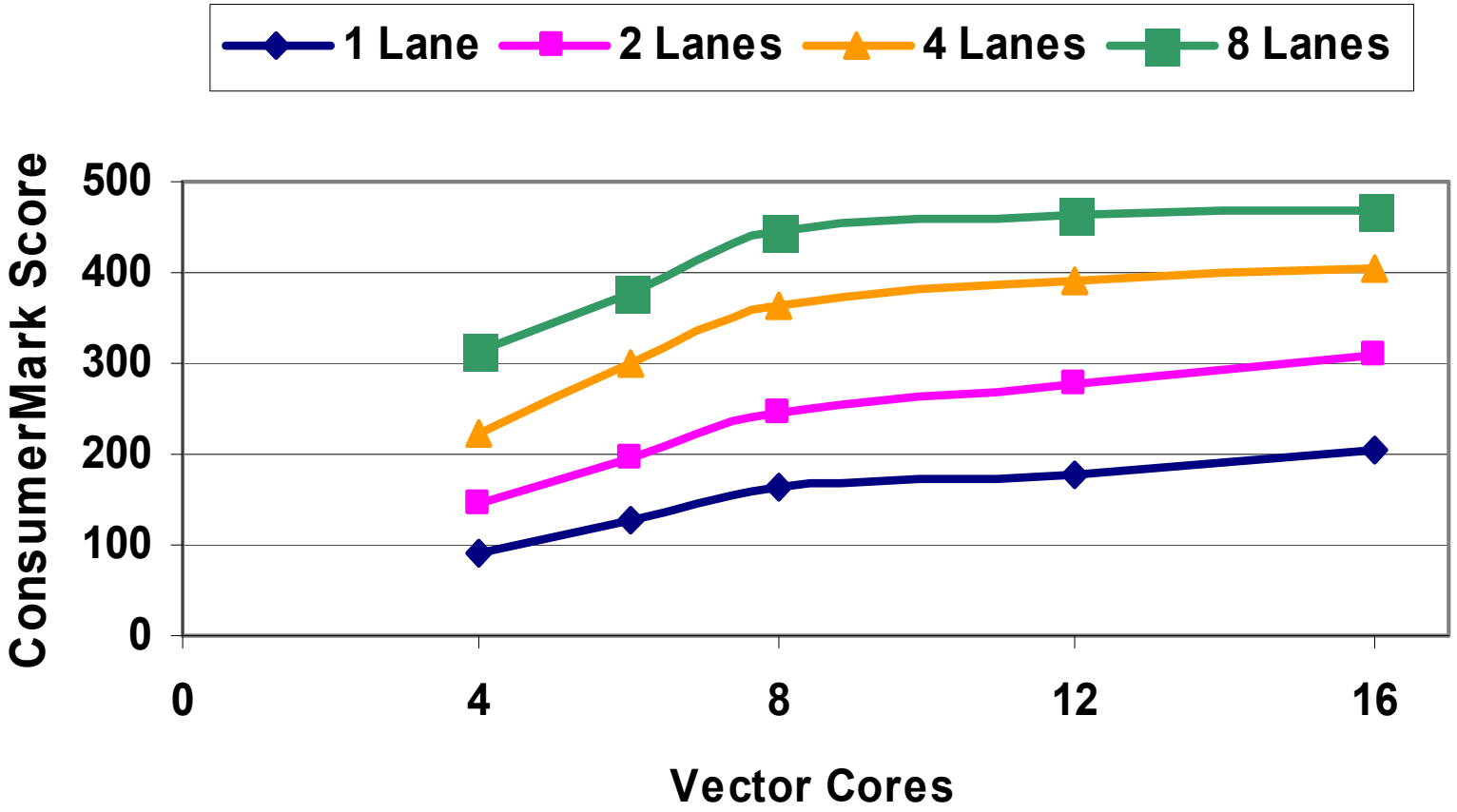
- Slowdown over memory system with1 cycle memory latency
- Memory latency in processor cycles

CODE: Precise Virtual Memory Exceptions



- Approach: use unallocated registers to maintain old register values until exception behavior is known
- It requires changes to the issue logic only
  - History file for updates to the renaming table

CODE: Scalability



- 200 MHz clock frequency

Conclusions

- It is possible to design efficient microprocessors for embedded multimedia systems with
  - High performance
  - Low energy/power consumption
  - Low design complexity
  - High scalability
- Thesis contributions
  - Demonstrated the efficiency of the VIRAM architecture with multimedia tasks
  - Presented & analyzed the VIRAM-1 media-processor
  - Presented & analyzed the CODE vector microarchitecture
  - Demonstrated that embedded DRAM is an appropriate technology for high bandwidth memory systems

Future Work

- Further application development
- Languages & compilers for multimedia processing
- Improved memory systems for CODE
- Architectures for data-level & thread-level parallelism
- Specialized hardware engines for complicated tasks
- Exploit modularity for yield & reliability improvements

Acknowledgements

- The U.C. Berkeley IRAM Group
  - Advisors: David Patterson, Kathy Yelick
  - Hardware: Sam Williams, Joe Gebis, Hiro Hamasaki, Iakovos Mavroidis, Ioannis Mavroidis
  - Software: Dave Martin, Dave Judd, Rich Fromm, Brian Gaeke, Mani Narayanan
  - Others: Krste Asanovic, Jim Beck, John Wawrzynek
- Help from:
  - IBM, MIPS Technologies, Cray, Avanti
- Funding from:
  - DARPA, California State, DoE
  - IBM Research Fellowship