Perspectives on Streaming

Christos Kozyrakis

Computer Systems Laboratory

Stanford University

christos@ee.stanford.edu

Motivation for Streaming

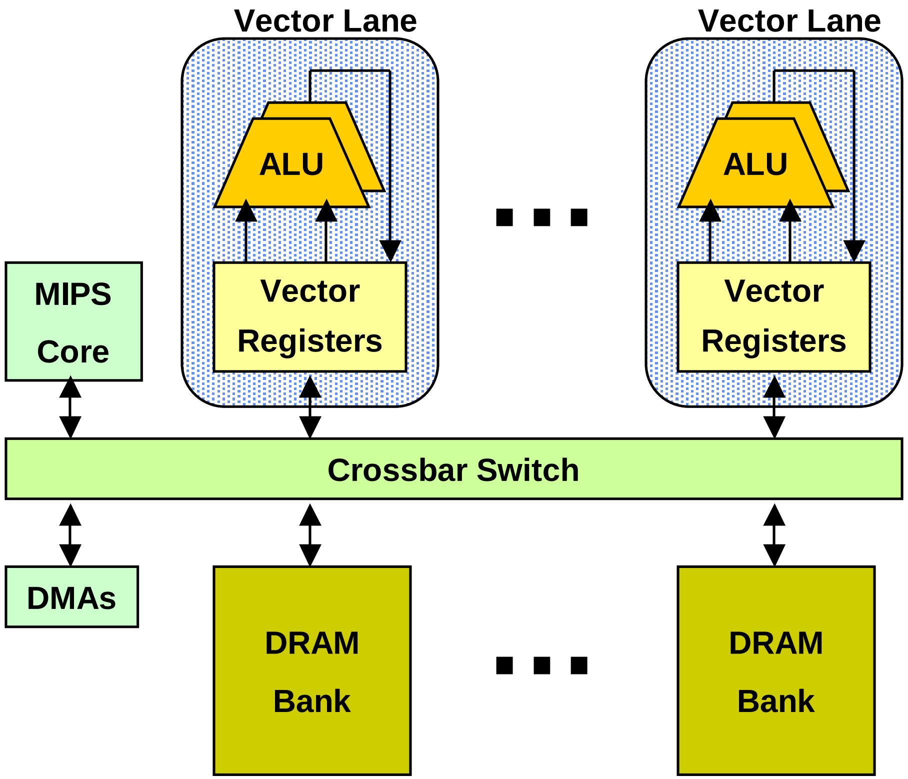
- **Hard problems for architects**
  - Identify independent tasks to execute in parallel
  - Memory access latency
  - Constraints of embedded & server applications
    - ► Power, cost, complexity, real-time guarantees, scalability

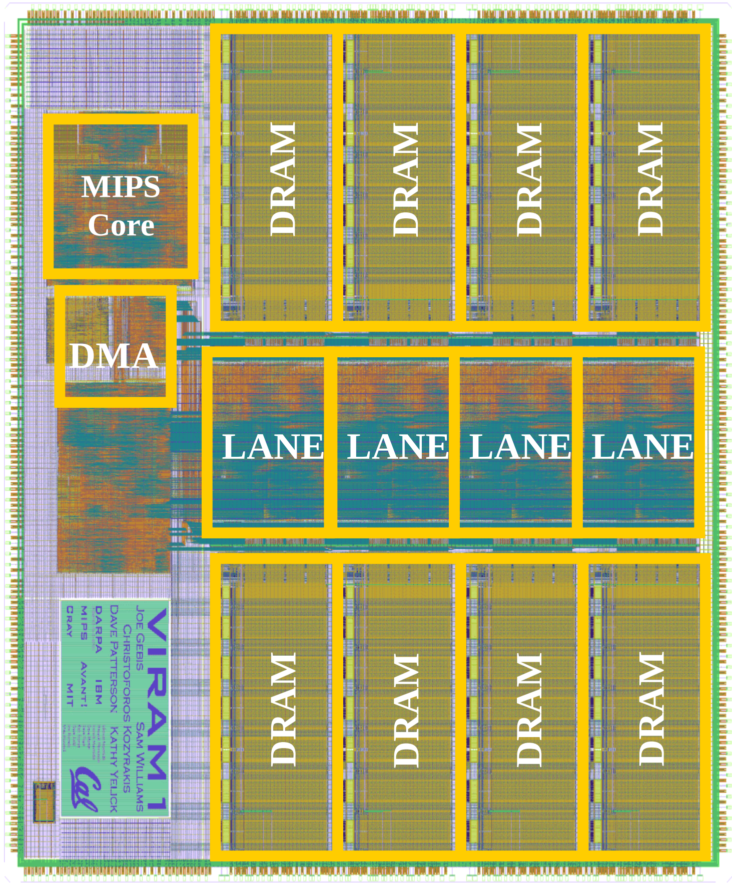
- **Opportunity: streaming nature of important data-intensive applications**
    - ► Video, audio, graphics, telecom, physical simulation, …
  - Data-level and task-level parallelism
  - Structured data access and communication patterns

Streaming from Architect's Perspective

- **A programming model that allows**
  - Separation of computation from data accesses
  - To express the parallelism & patterns in applications
  - Compiler can analyze and schedule the computation and data accesses
- **Benefits for hardware architecture**
  - Can use parallel architecture (10s to 100s of PEs)
  - Can use clustering and deep memory hierarchies
  - Can use high bandwidth to tolerate latency
  - Can minimize control overhead in PEs and mem. hierarchy
- **High performance with hardware that is scalable, power efficient**

The VIRAM “Stream” Processor





The VIRAM “Stream” Processor

- **Vector hardware**
  - 125M transistors, 13MB DRAM, 0.18um CMOS
  - Single issue, in order, no hardware caches
  - 4.8 Gops (32b), 200MHz, 2W
- **Performance**
  - Evaluated for multimedia, telecom, and scientific apps
  - 10x over superscalar and VLIW
  - Even better with a clustered, decoupled vector processor
    - See MICRO’02, IPDPS’02, ISCA’03
- **“Streaming” software**
  - C with pragmas for data-level parallelism
    - Sufficient for many array-based and SDF computations
  - Vectorizing compiler (based on Cray compiler)

Challenges for Future Work

- **Convergence of streaming architectures**
  - Though polymorphic hardware
- **Convergence of streaming languages (?)**
  - Focus on wide acceptance
- **Voltage/frequency/architecture scaling based on stream analysis and scheduling**
  - Specification of performance/power constraints in language?
  - Need run-time system support?
- **Streaming beyond synchronous data-flow applications and array-based computations**
  - For apps that are not statically analyzable?
  - For apps with pointer-based data structures?

Streaming beyond Arrays

- **Many data-intensive apps use pointer-based structures**
  - Trees, hash tables, sets, lists, Markov chains
- **An initial approach to "streaming"**
  - Based on libraries with template-based data-structures
    - ► Standard templates library (STL) in C++, standard containers in Java
  - Library API describes algorithmic properties, invariants, and asymptotic complexities
  - Treat library members as "built-in" data types in compiler
    - ► To separate memory accesses from computation
    - ► To reason about dependencies, ordering, access pattern,…
  - Current work: an STL-aware C++ compiler
    - ► Target polymorphic multiprocessors

Implications for Hardware Architecture

- **Memory hierarchy for array-based streaming is simple**
  - DMA engines and software managed caches

- **Memory hierarchy for pointer-based structures?**
  - DMA engine → memory processor?
    - Executes the data-structure API
    - Prefetches (preevicts) data to (from) level of the hierarchy
    - Adapts prefetching rate and placement policy
  - Software managed caches → polymorphic caches?
    - Can be used for caches, buffers, scratchpad, …
    - Configured based on the properties of data-structures and computation

Summary

- **Benefits of streaming**
  - Separation of computation & data accesses
  - Data/task parallelism and data access patterns
- **Characteristics of streaming hardware**
  - Modular, parallel architecture
  - High bandwidth memory hierarchy
  - Simple control
- **Challenges for future streaming systems**
  - Convergence of architectures and languages
  - Streaming for non-static applications
  - Streaming for pointer-based applications