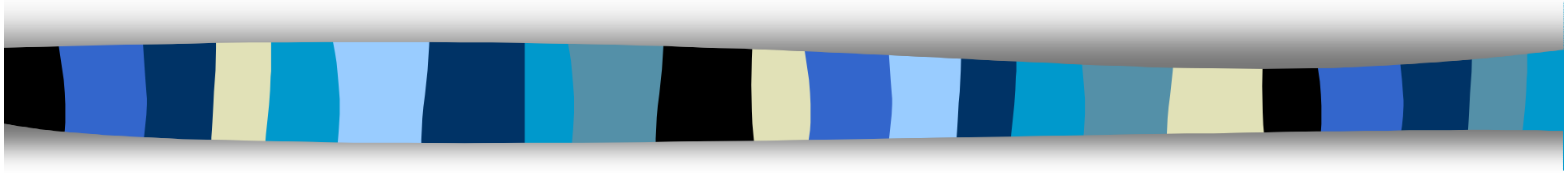
Explicitly Parallel Architectures for Memory Performance Enhancement



Christoforos E. Kozyrakis

Computer Science Division
University of California at Berkeley

kozyraki@cs.berkeley.edu
http://iram.cs.berkeley.edu

Thesis

- Memory performance enhancement requires the synergy of hardware and software techniques
- This talk: focus on hardware
  - Explicitly parallel instruction sets for efficient communication of parallelism between hardware and software
  - Exploit memory bandwidth to hide or tolerate high memory latency

Software-Hardware Strengths

- Software (compiler/run-time system)
  - Better view and understanding of both application requirements and system resources
  - Ability to apply complex optimizations over thousands of source-code lines
- Hardware
  - Massive resources for parallel execution of memory and arithmetic operations
  - Can use dynamic (run-time) information to apply aggressive optimizations within a small window of instructions

Problems with Current ISAs

- Strict sequential semantics
  - Sequential ordering even for independent ops
  - Any parallelism must be (re-)discovered by HW
- Lack of memory access pattern information
  - Any regular access pattern is broken into a series of accesses to a single memory location
  - Access pattern must be "(re-)discovered" by HW to apply optimizations (e.g. prefetching)
- Small number of registers
  - Exacerbate the memory performance problem
  - Make several compiler optimizations difficult

The Semantics Problem

- Starting with:

```
for (i=0; i<N; i++) *(A+10*i)++;
```

- You get (naive version):

```
Loop:   ld    t0, 0(A)
        addi  t0, t0, 1
        st    t0, 0(A)
        addi  A,  A, 10
        addi  i,  i, 1
        bne   i,  N, Loop
```

No stride info!

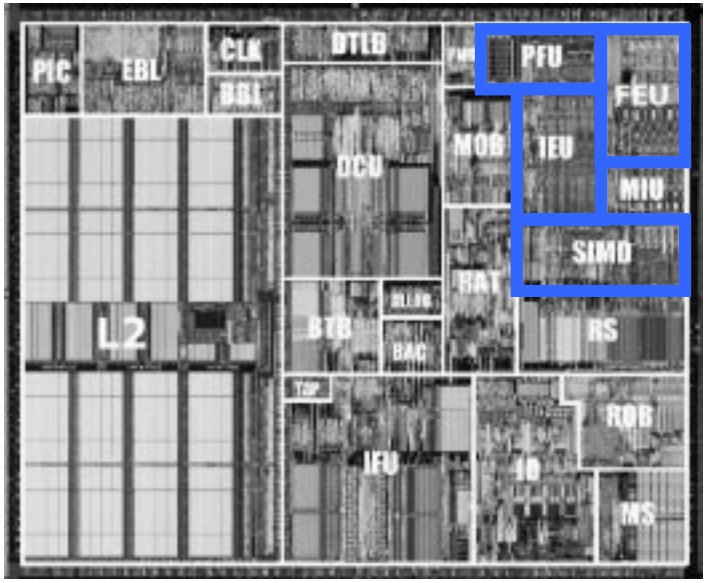
No locality info!

No mem dependence info

Dependent operations!

- To get high performance you need
  - Unrolling, SW pipelining, prefetching, scheduling (SW)
  - Prefetching, stride predictor, dynamic scheduling, speculation, dynamic memory disambiguation (HW)

Hardware for Current ISAs



Pentium® III

- ~80% of the die used to (re-)discover parallelism and address memory dependencies and latency
- Techniques like caching and speculation are becoming inefficient in resource utilization
- Poor match to media applications with real-time requirements and limited spatial locality of data

Explicitly Parallel Instruction Sets

Feature

- Specify independent ops to be executed in parallel
- Provide explicit memory access pattern info
- Provide large number of registers
- Support for predication, load/store speculation

Benefit

- Simpler instruction issue logic
- Simper (pre)fetching decisions; effective cache management
- Less memory accesses; enable SW prefetching; less memory dependencies
- More aggressive scheduling of load/store ops; simpler HW for memory dependencies and exceptions (faults)

Parallel Semantics

- Starting with:

```
for (i=0; i<N; i++) *(A+10*i)++;
```

- You want to get:

```
Loop:   vld.nc  v0, A, 10, L;  addi i,  i, L
        vaddi   v0, v0, 1
        vst.nc  v0, A, 10, L;  addi A, A, 10L
        bne i, N, Loop
```

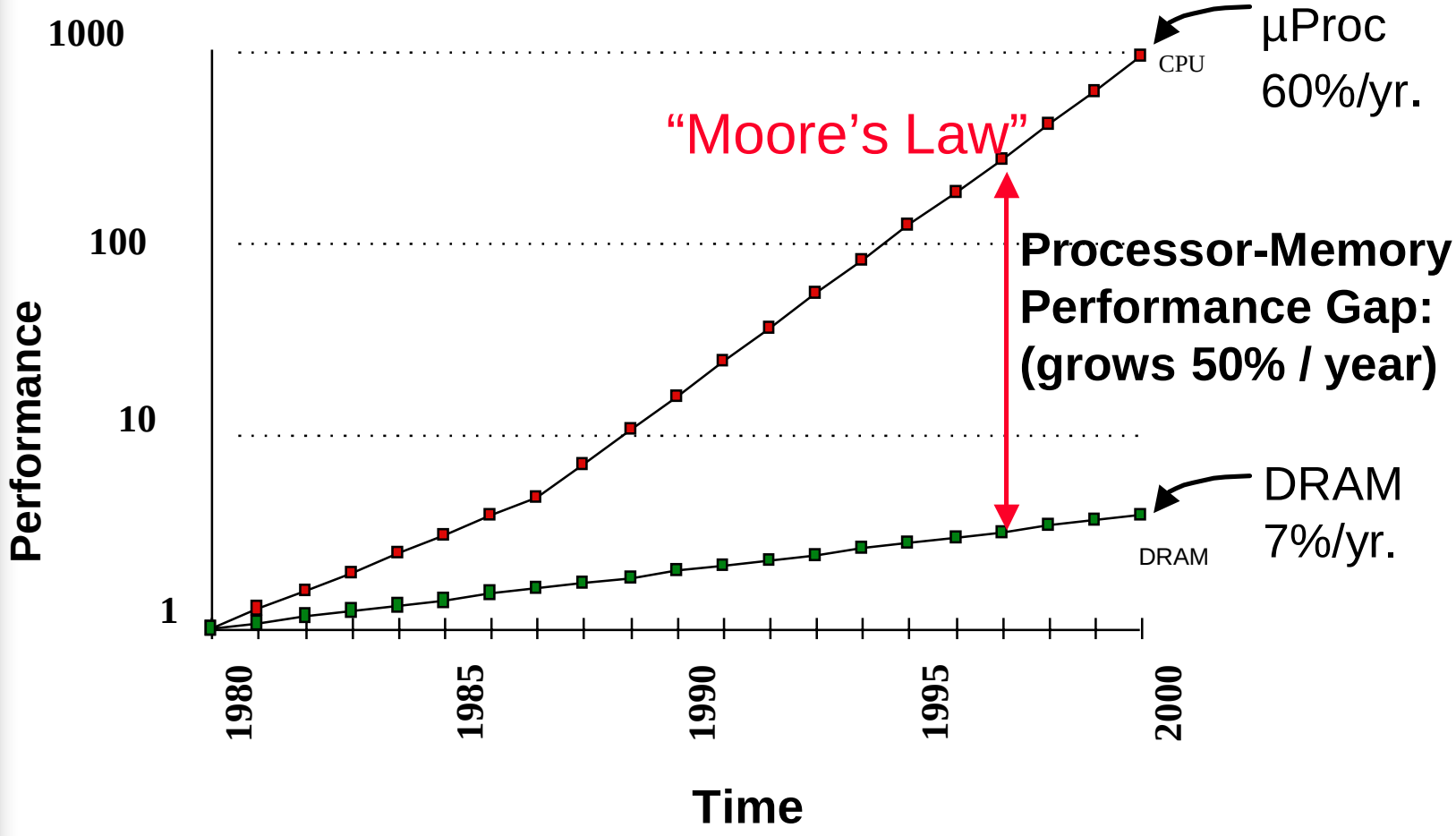
Vector load/store; stride 10; L elements; no-caching

Independent operations; execute in parallel

Explicit ISAs & Systems

Example	Features
IA-64 (EPIC)	■Intel & HP ■VLIW with 128 registers ■Memory accesses with locality hints ■Load/store speculation and predication
Vector IRAM	■UC Berkeley ■Vector ISA with 8 Kbyte register file ■Vector memory accesses (strided/indexed) ■Load speculation and predication
Impulse	■U. of Utah ■Programmable memory controller ■Used for optimized prefetching for regular patterns and remapping for better caching

The Memory Latency Problem



Looking into the Future

- DRAM latency unlikely to be dramatically reduced
- Memory bus latency scales slowly too
- Microprocessors keep improving fast
  - More instructions issued in parallel
  - More datapaths per die
  - More processor cores per die
- Applications are memor -intensive
  - Many media applications do not cache well due to their streaming nature
  - Database applications work on huge data-sets that do not fit in caches
- Memory latency will become even more critical

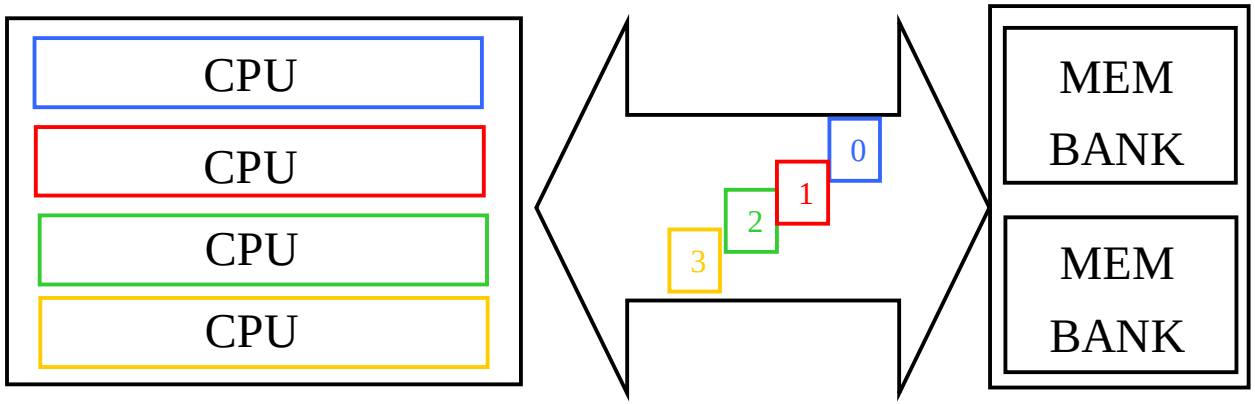
Latency Hiding Techniques

Technique	Problems
Larger caches	Area; clock cycle; power
Efficient caches	Clock cycle
Larger cache blocks	Redundant fetches; power; capacity misses
HW prefetching	Complexity; redundant fetches; power
Lock-up free caches	Complexity; requires efficient scheduling
Speculative loads (dynamic scheduling)	Complexity; redundant fetches; power

Utilizing Memory Bandwidth

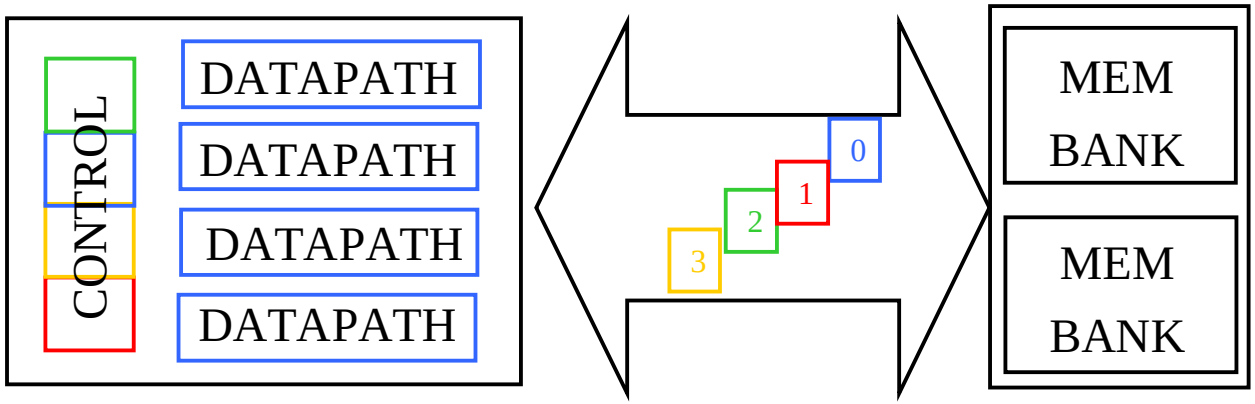
- High bandwidth available for future designs
  - multi-bank memory systems, embedded DRAM, cost-efficient MCMs, high-bandwidth interfaces (DDR-II, Rambus)
- How bandwidth is turned into performance
  - Issue early: issue memory accesses early
  - Fetch many: support a large number of pending accesses to the memory system
  - Fetch useful: avoid redundant fetches (less power, less interference)
  - Utilize access pattern info: to reduce memory conflicts and maximize caching efficiency

Single-chip Multiprocessors



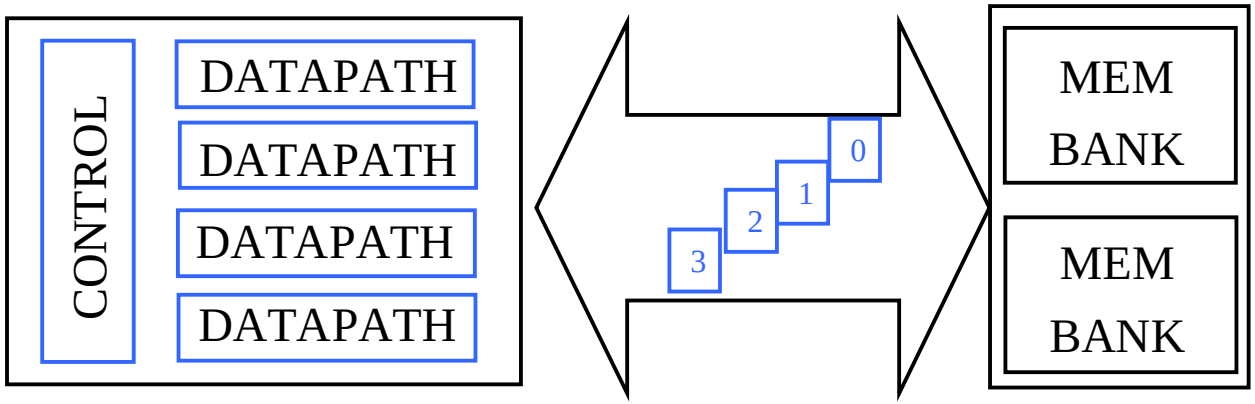
- Multiple programs/threads run in parallel on multiple CPUs
- Multiplexed access streams utilize the high memory bandwidth
- Each access stream observes the same (long) latency, but the overall system throughput is high
- Very efficient in the presence of coarse-grain parallelism

Multi-threaded Processors



- Multiple threads running on a single core
- When a thread stalls due to memory latency, another thread is executed
- Multiplexed access streams utilize the high memory bandwidth
- Each access stream observes the same (long) latency, but the overall system throughput is high

Vector Architectures



- Multiple memory accesses from one instruction
- Explicit access pattern information
- Decoupling used tolerate long latencies
- Deep pipelining to amortize the latency over a large number of elements
- Very efficient in the presence of fine-grain parallelism

Summary

- Memory performance enhancement requires the synergy of hardware and software techniques
- Explicitly parallel instruction sets
  - Provide parallel execution semantics
  - Allow simpler, efficient hardware
  - Enable aggressive optimization opportunities in software
- Exploiting memory bandwidth in hardware
  - The remedy to high memory latency
  - Required hardware features:
    - Issue accesses early
    - Support many pending accesses
    - Avoid redundant transfers
    - Utilize access pattern information