Comparing Memory Systems for Chip Multiprocessors

Jacob Leverich

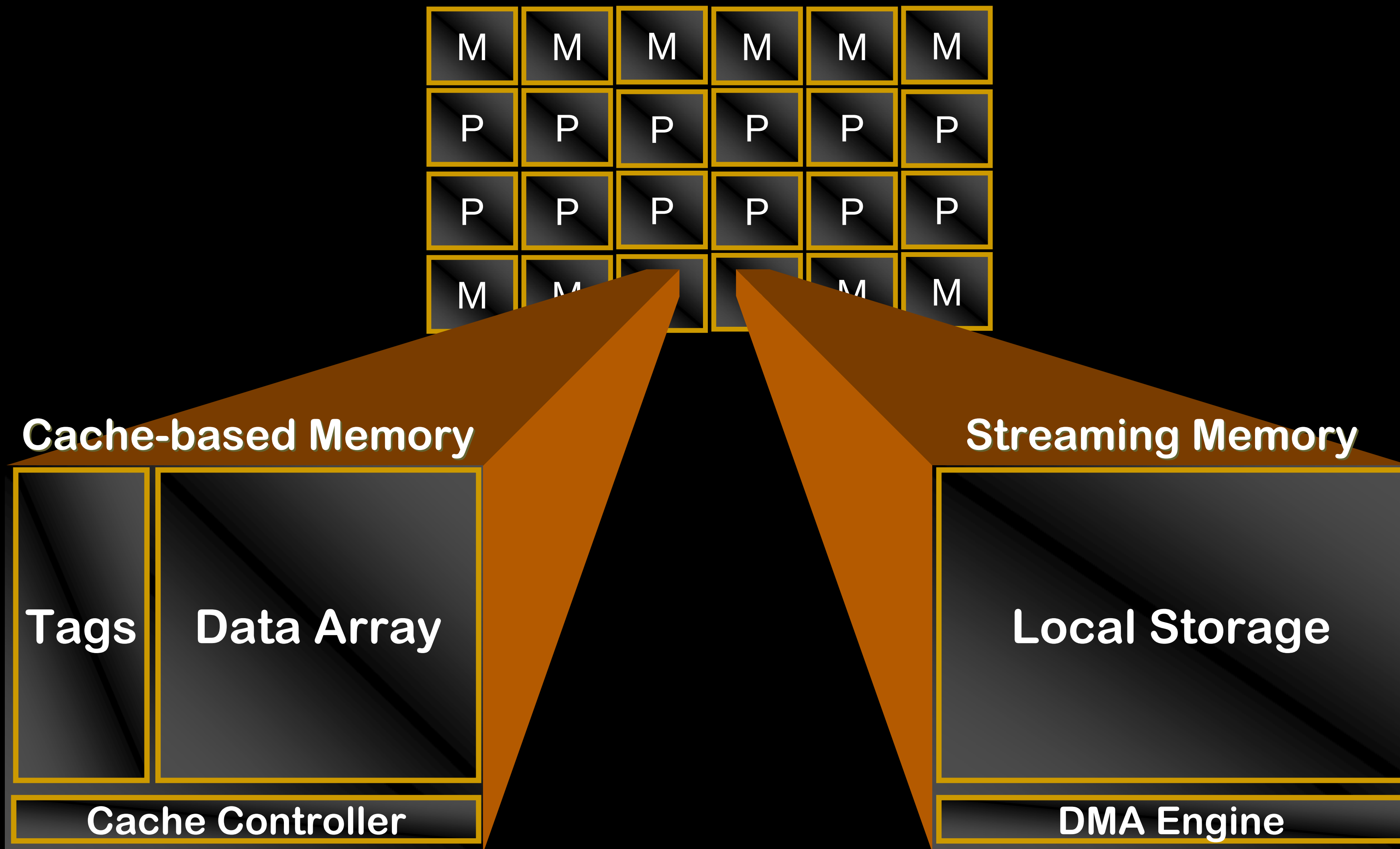
Hideho Arakida, Alex Solomatnikov,
Amin Firoozshahian, Mark Horowitz,
Christos Kozyrakis

Computer Systems Laboratory
Stanford University

Cores are the New GHz

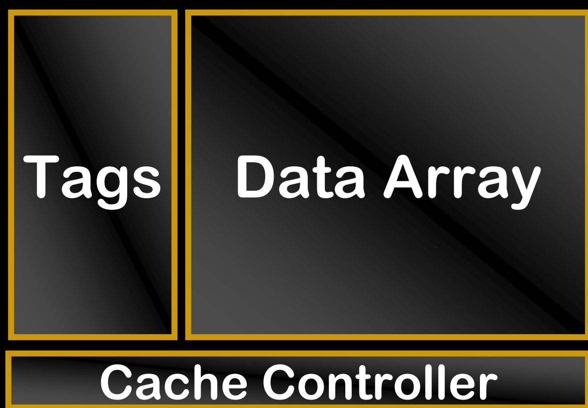
- 90s: ↑GHz & ↑ILP
  - Problems: power, complexity, ILP limits
- 00s: ↑cores
  - Multicore, manycore, …

What is the New Memory System?



The Role of Local Memory

Cache-based Memory



Streaming Memory

- Exploit spatial & temporal locality
- Reduce average memory access time
  - Enable data re-use
  - Amortize latency over several accesses
- Minimize off-chip bandwidth
  - Keep useful data local

Who Manages Local Memory?

	Cache-based	Streaming
Locality		
Data Fetch	Reactive	Proactive
Placement	Limited mapping	Arbitrary
Replacement	Fixed-policy	Arbitrary
Granularity	Cache block	Arbitrary
Communication		
Coherence	Hardware	Software

Cache-based: Hardware-managed

Streaming: Software-managed

Potential Advantages of Streaming Memory

- Better latency hiding
  - Overlap DMA transfers with computation
  - Double buffering is macroscopic prefetching
- Lower off-chip bandwidth requirements
  - Avoid conflict misses
  - Avoid superfluous refills for output data
  - Avoid write-back of dead data
  - Avoid fetching whole lines for sparse accesses
- Better energy and area efficiency
  - No tag & associativity overhead
  - Fewer off-chip accesses

How *Much* Advantage over Caching?

- How do they differ in Performance?
- How do they differ in Scaling?
- How do they differ in Energy Efficiency?
- How do they differ in Programmability?

Our Contribution: A Head to Head Comparison

Cache-based Memory vs. Streaming Memory

- Unified set of constraints
  - Same processor core
  - Same capacity of local storage per core
  - Same on-chip interconnect
  - Same off-chip memory channel
- Justification
  - VLSI constraints (e.g., local storage capacity)
  - No fundamental differences (e.g., core type)

Our Conclusions

- Caching performs & scales as well as Streaming
  - Well-known cache enhancements eliminate differences
- Stream Programming benefits Caching Memory
  - Enhances locality patterns
  - Improves bandwidth and efficiency of caches
- Stream Programming easier with Caches
  - Makes memory system amenable to irregular & unpredictable workloads
- Streaming Memory likely to be replaced or at least augmented by Caching Memory

Simulation Parameters

- 1 – 16 cores: Tensilica LX, 3-way VLIW, 2 FPUs
  - Clock frequency: 800 MHz – 3.2 GHz
- On-chip data memory
  - Cache-based: 32kB cache, 32B block, 2-way, MESI
  - Streaming: 24kB scratch pad
    DMA engine
    8kB cache, 32B block, 2-way
  - Both: 512kB L2 cache, 32B block, 16-way
- System
  - Hierarchical on-chip interconnect
  - Simple main memory model (3.2 GB/s – 12.8 GB/s)

Benchmark Applications

- No “SPEC Streaming” ☹
  - Few available apps with streaming & caching versions
- Selected 10 “streaming” applications
  - Some used to motivate or evaluate Streaming Memory
- Co-developed apps for both systems
  - Caching: C, threads
  - Streaming: C, threads, DMA library
- Optimized both versions as best we could

Benchmark Applications

- Video processing
  - Stereo Depth Extraction
  - H.264 Encoding
  - MPEG-2 Encoding
- Image processing
  - JPEG Encode/Decode
  - KD-tree Raytracer
  - 179.art
- Scientific and data-intensive
  - 2D Finite Element Method
  - 1D Finite Impulse Response
  - Merge Sort
  - Bitonic Sort

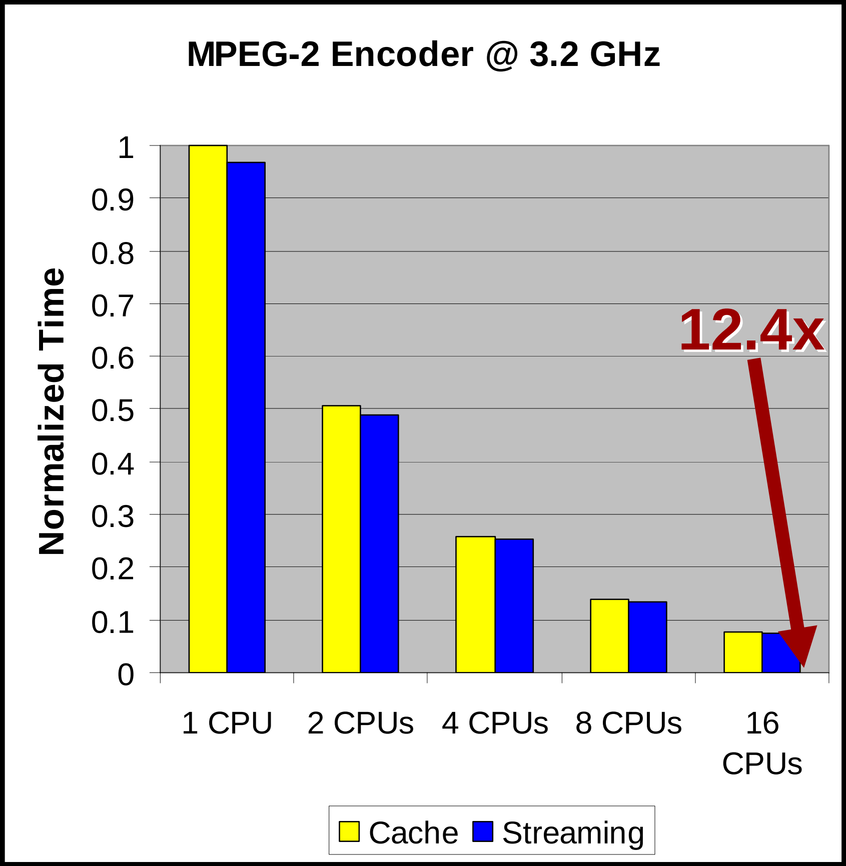
Irregular

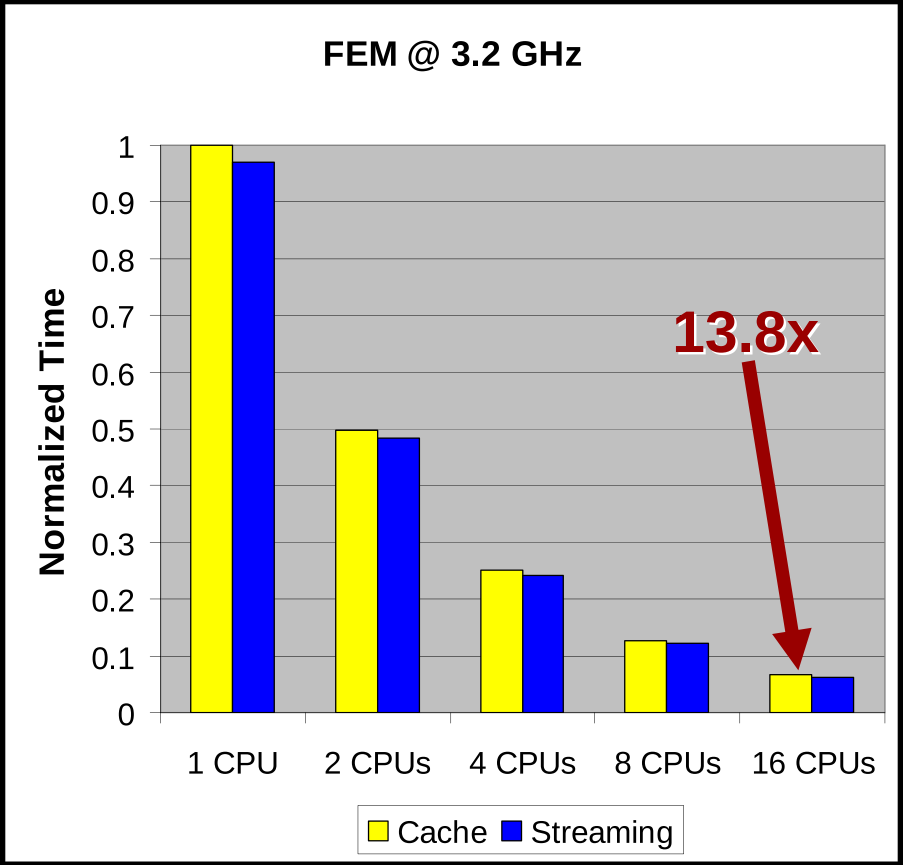
Unpredictable

Our Conclusions

- Caching performs & scales as well as Streaming
  - Well-known cache enhancements eliminate differences
- Stream Programming benefits Caching Memory
  - Enhances locality patterns
  - Improves bandwidth and efficiency of caches
- Stream Programming easier with Caches
  - Makes memory system amenable to irregular & unpredictable workloads
- Streaming Memory likely to be replaced or at least augmented by Caching Memory

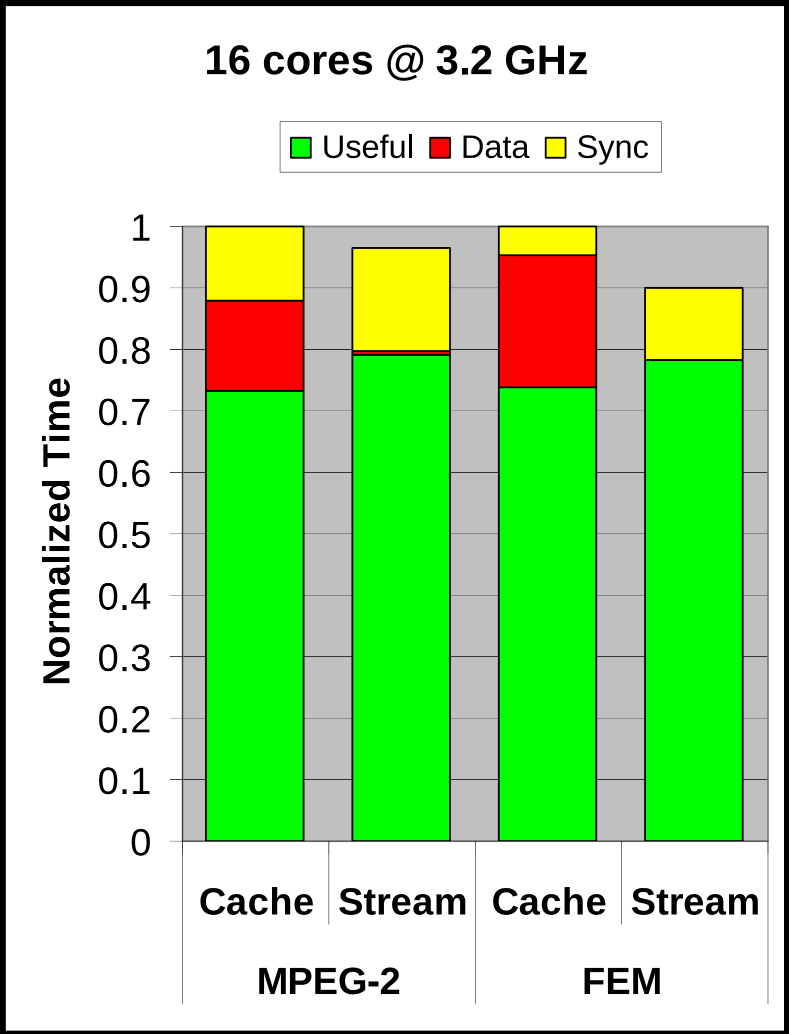
Parallelism Independent of Memory System





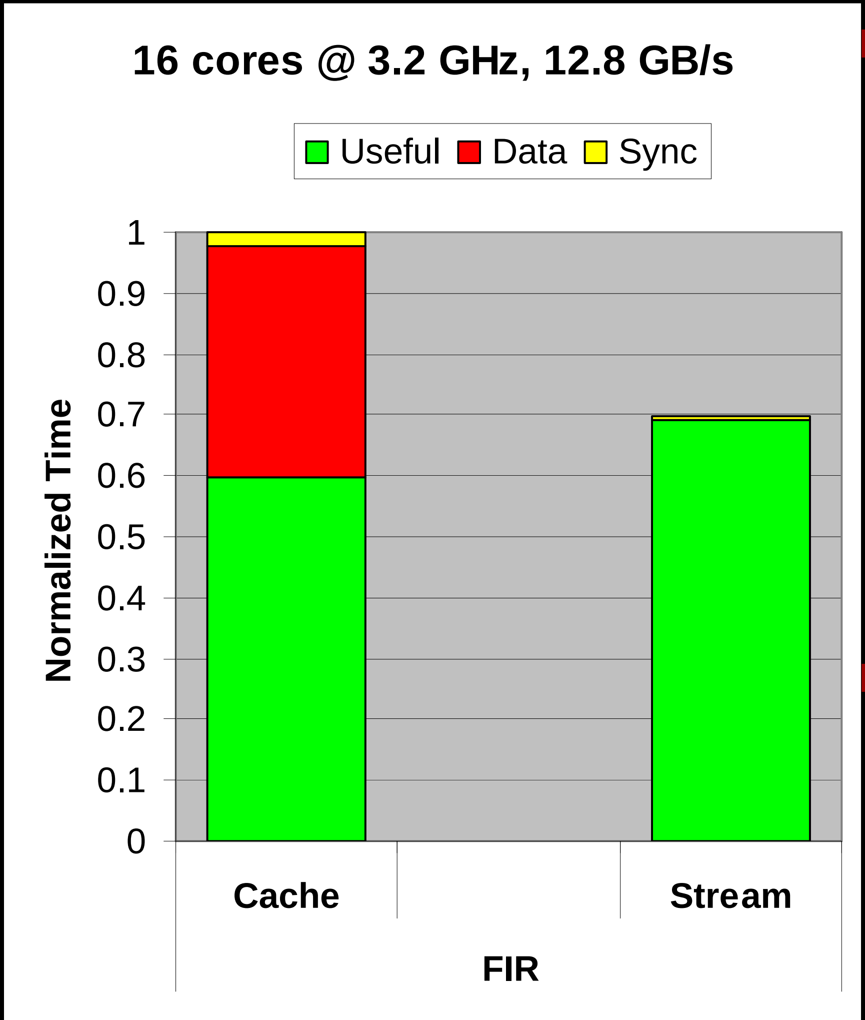
- 6/10 apps little affected by local memory choice

Local Memory Not Critical For Compute-Intensive Applications



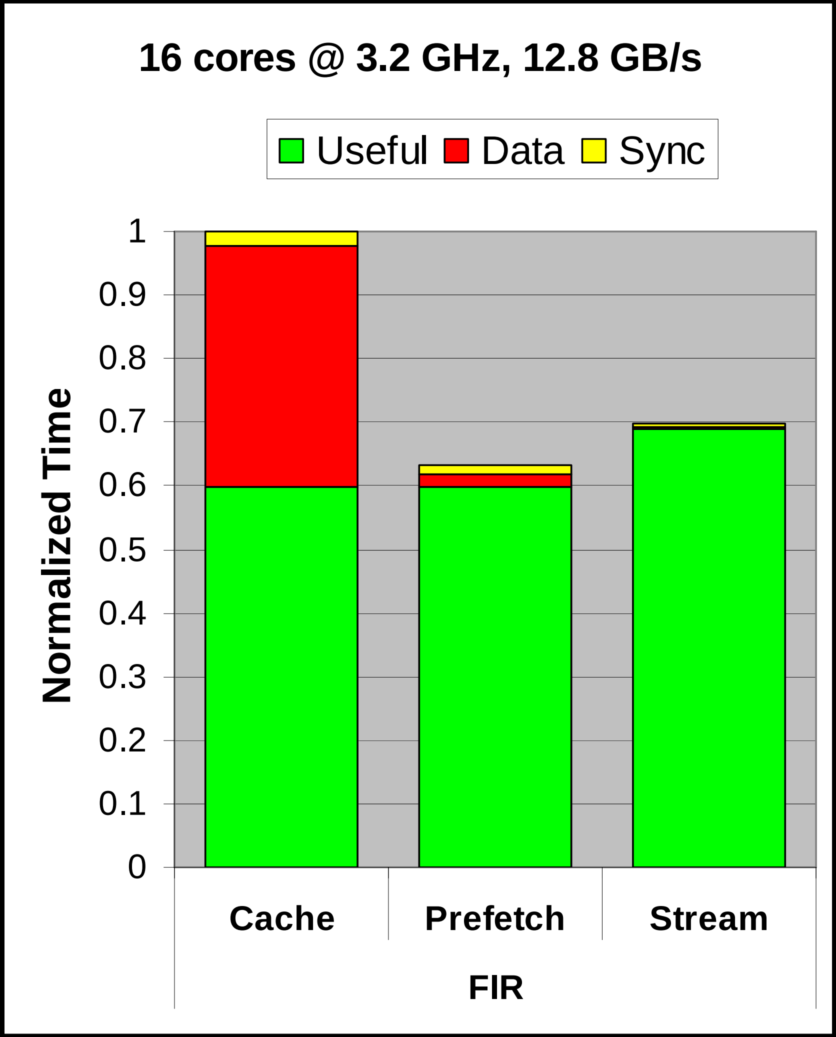
- Intuition
  - Apps limited by compute
  - Good data reuse, even with large datasets
  - Low misses/instruction
- Note:
  - “Sync” includes Barriers and DMA wait

Double-Buffering Hides Latency For Streaming Memory Systems



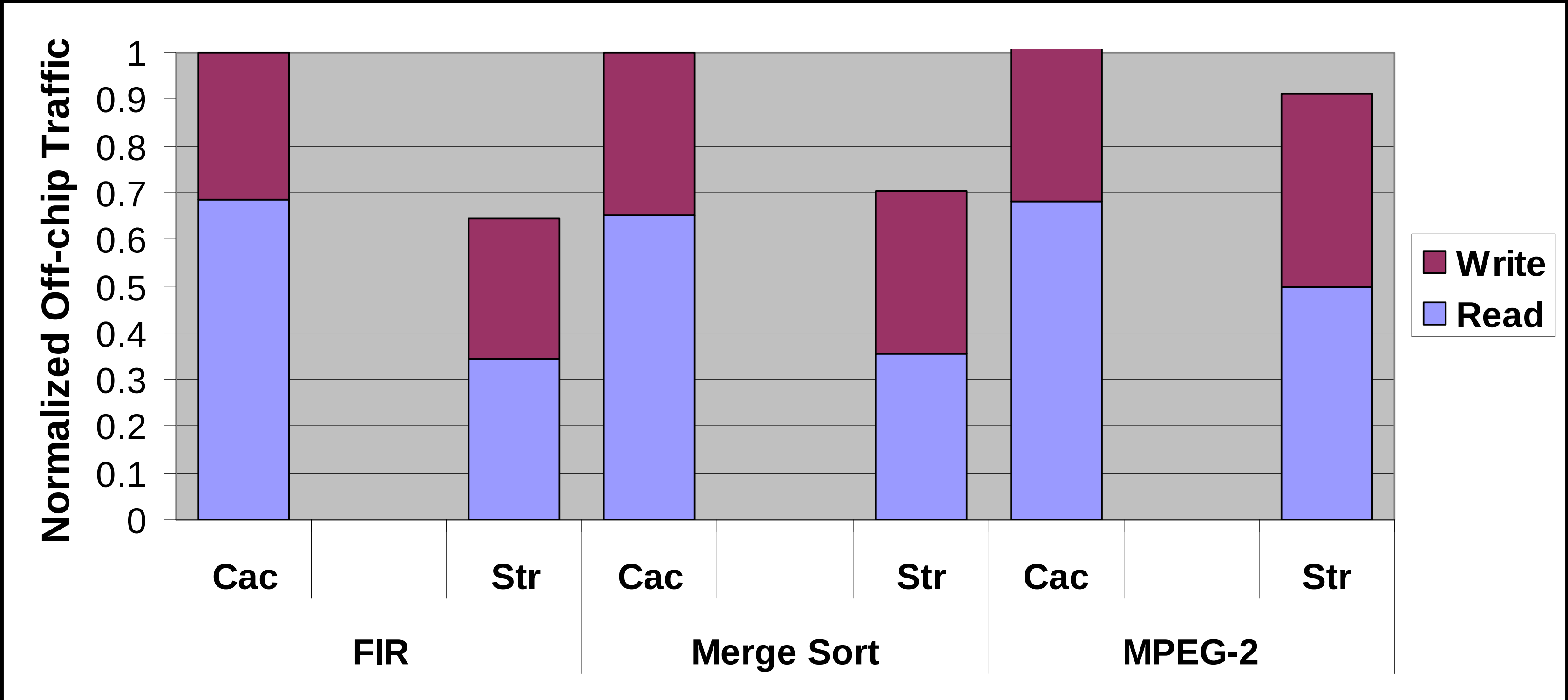
- Intuition
  - Non-local accesses entirely overlapped with computation
  - DMAs perform efficient SW prefetching
- Note
  - The case for memory-intensive apps not bound by memory BW
    - 179.art, Merge Sort

Prefetching Hides Latency For Cache-Based Memory Systems



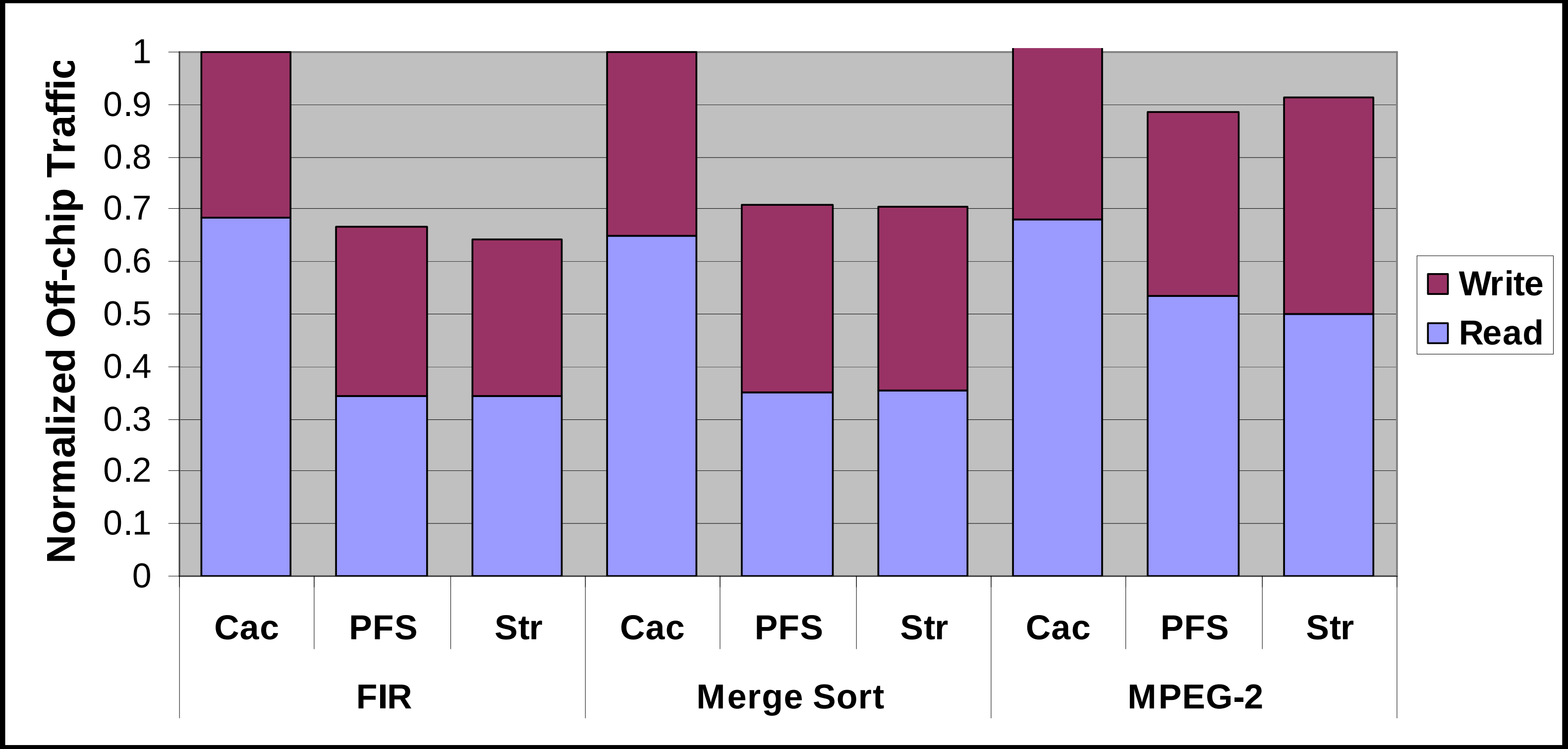
- Intuition
  - HW stream prefetcher overlaps misses with computation as well
  - Predictable & regular access patterns

Streaming Memory Often Incurs Less Off-Chip Traffic



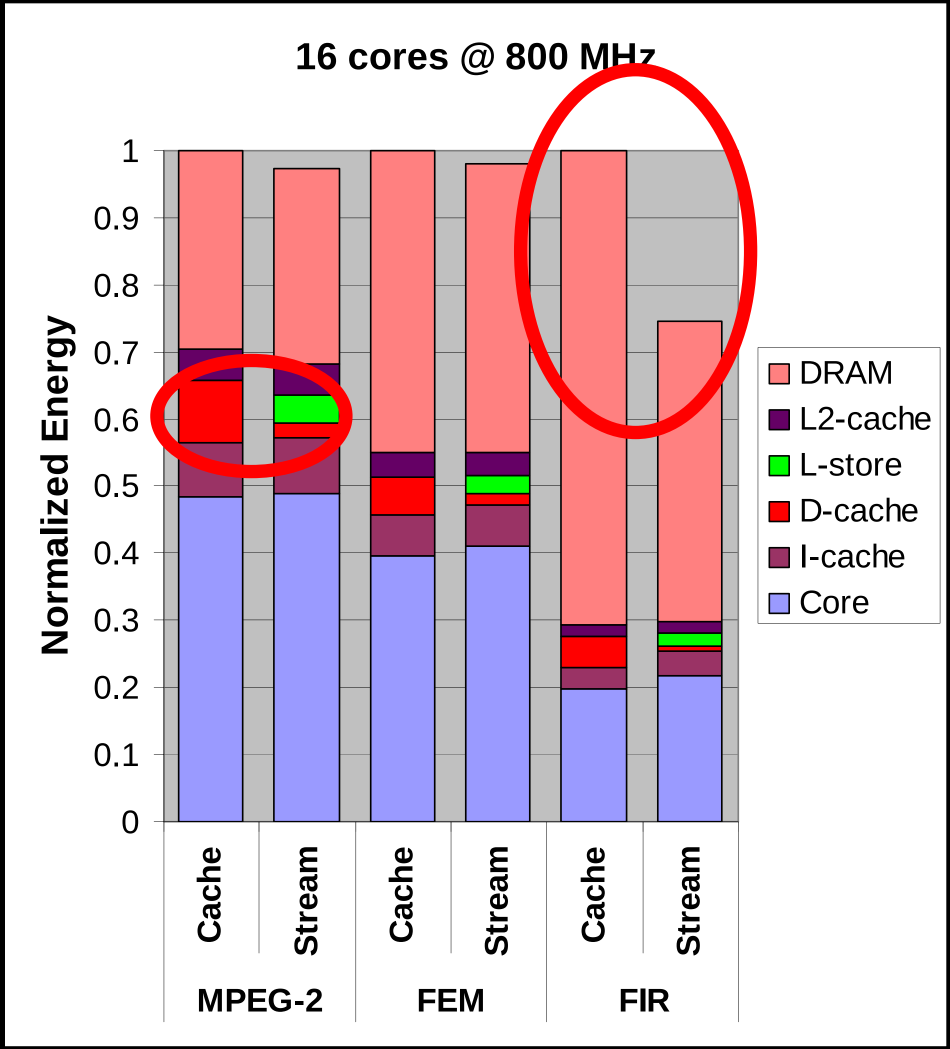
- The case for apps with large output streams
  - Avoids superfluous refills for output streams
  - Not the case for write-allocate, fetch-on-miss caches

SW-Guided Cache Policies Improve Bandwidth Efficiency



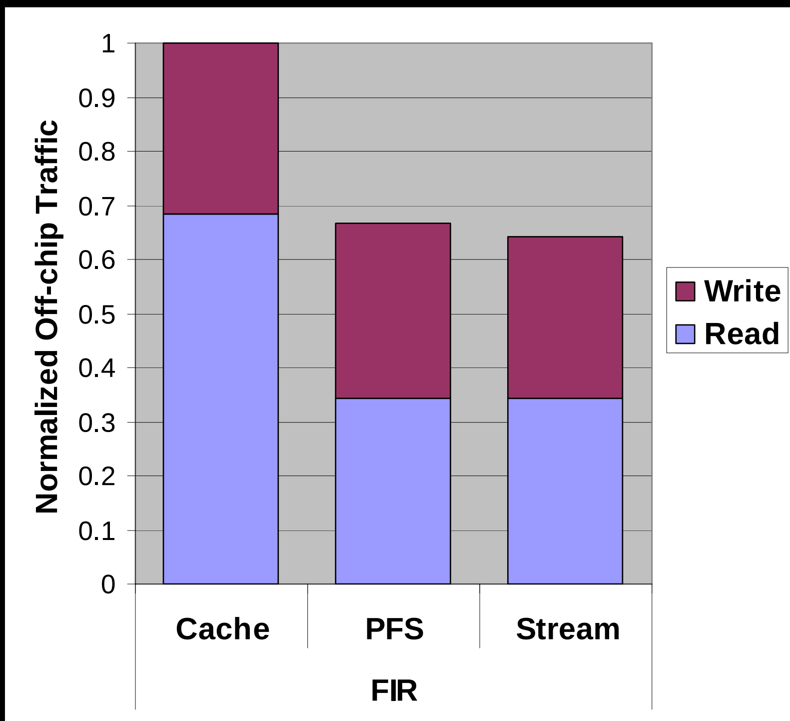
- Our system: “Prepare For Store” cache hint
  - Allocates cache line but avoid refill of old data
- Xbox360: write-buffer for non allocating writes

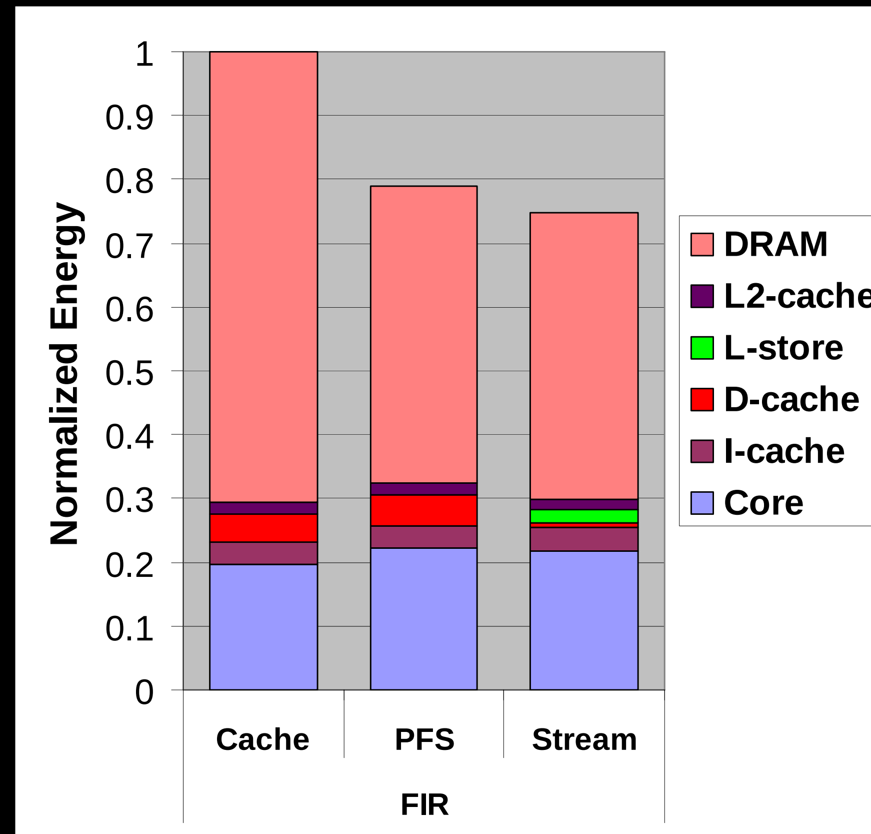
Energy Efficiency Does not Depend on Local Memory



- Intuition
  - Energy dominated by DRAM accesses and processor core
  - Local store ~2x energy-efficiency of cache, but small portion of total energy
- Note
  - The case for compute-intensive applications

Optimized Bandwidth Yields Optimized Energy Efficiency



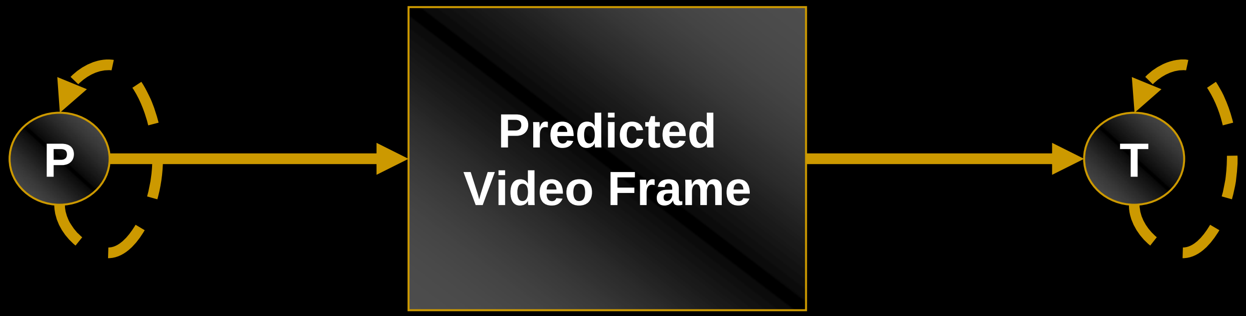


- Superfluous off-chip accesses are expensive!
- Streaming & SW-guided caching reduce them

Our Conclusions

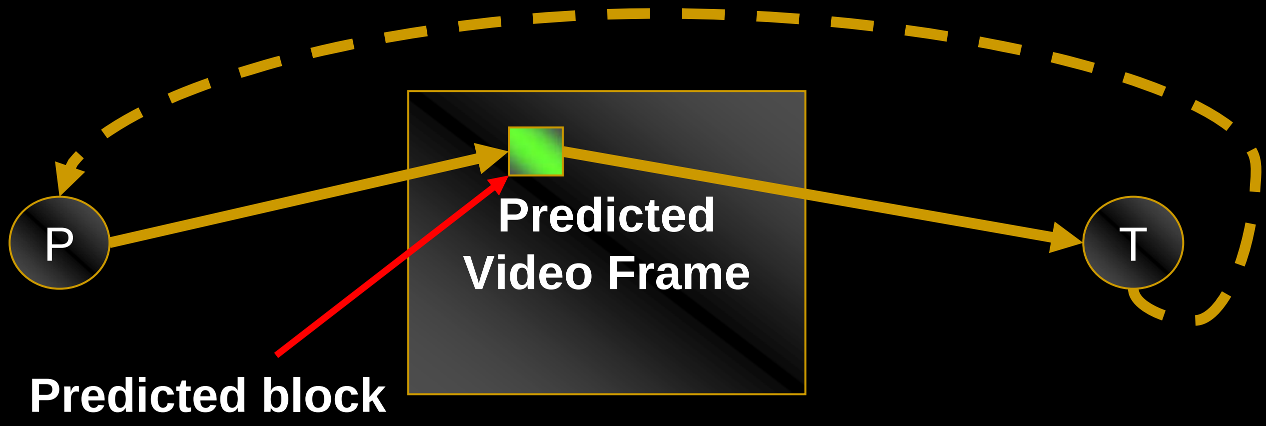
- Caching performs & scales as well as Streaming
  - Well-known cache enhancements eliminate differences
- Stream Programming benefits Caching Memory
  - Enhances locality patterns
  - Improves bandwidth and efficiency of caches
- Stream Programming easier with Caches
  - Makes memory system amenable to irregular & unpredictable workloads
- Streaming Memory likely to be replaced or at least augmented by Caching Memory

Stream Programming for Caches: MPEG-2 Example



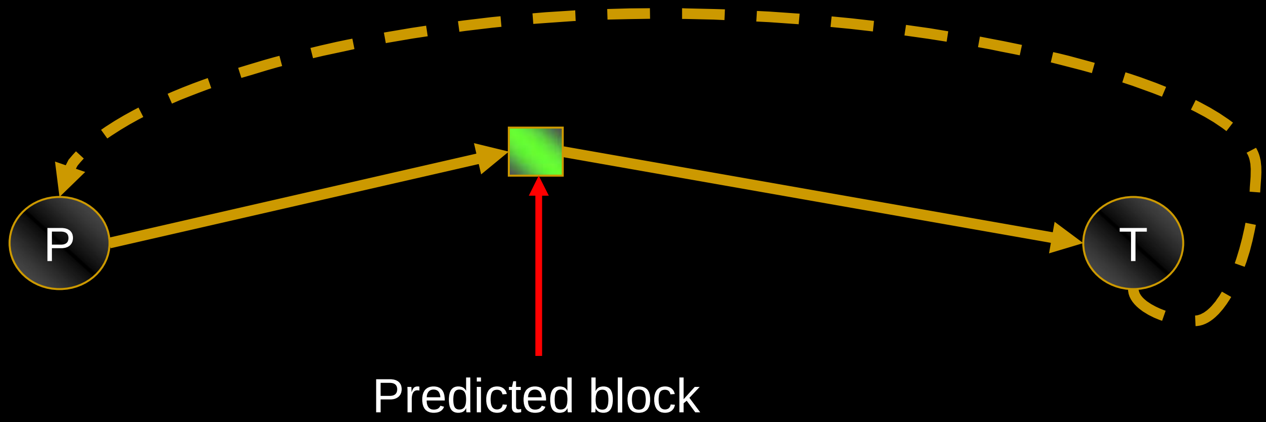
- MPEG-2 example
  - P() generates a video frame later consumed by T()
  - Whole frame is too large to fit in local memory
  - No temporal locality
- Opportunity
  - Computation on frame blocks are independent

Stream Programming for Caches: MPEG-2 Example



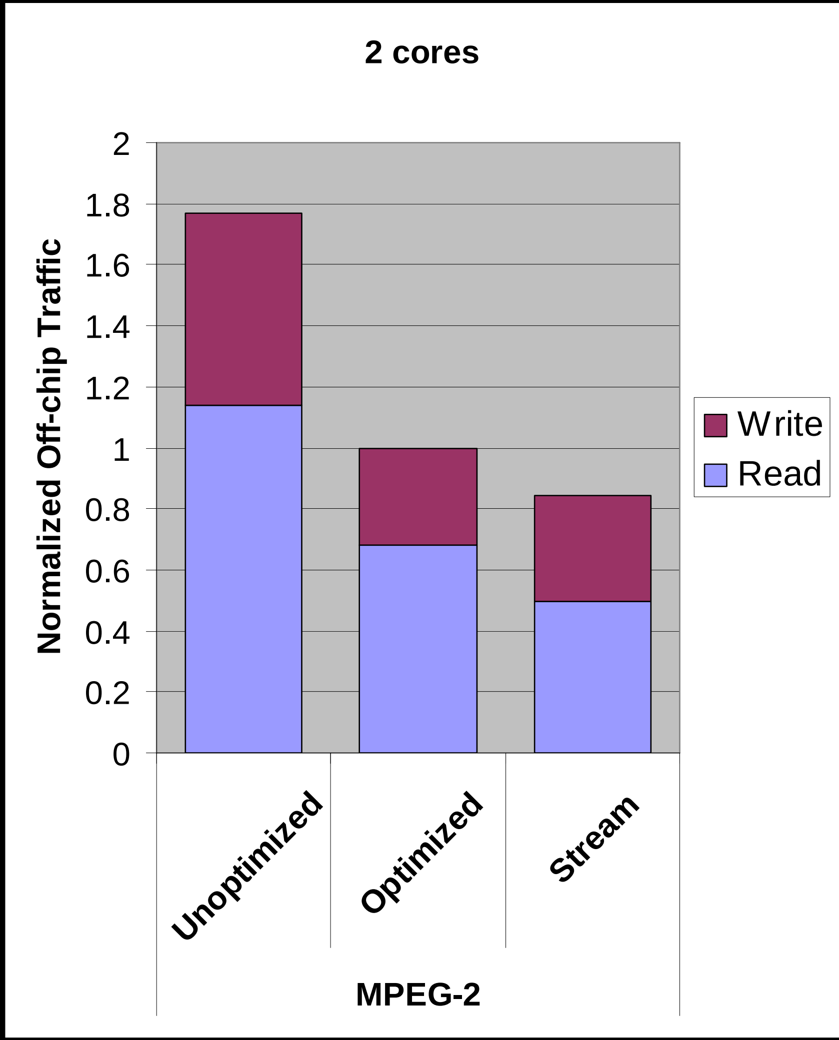
- Introducing temporal locality
  - Loop fusion for P() and T() at block level
  - Intermediate data are dead once T() done

Stream Programming for Caches: MPEG-2 Example



- Exploiting producer-consumer locality
  - Re-use the predicted block buffer
  - Dynamic working set reduced
  - Fits in local memory; no off-chip traffic

Stream Programming for Caches: MPEG-2 Example



- Stream programming beneficial for any Memory System
  - Exposes locality that improves bandwidth and energy efficiency of local memory
- Stream programming toolchains helpful

Our Conclusions

- Caching performs & scales as well as Streaming
  - Well-known cache enhancements eliminate differences
- Stream Programming benefits Caching Memory
  - Enhances locality patterns
  - Improves bandwidth and efficiency of caches
- Stream Programming easier with Caches
  - Makes memory system amenable to irregular & unpredictable workloads
- Streaming Memory likely to be replaced or at least augmented by Caching Memory

Stream Programming is Easier with Caches

- Stream programming necessary for correctness on Streaming Memory
  - Must refactor all dataflow
- Caches can use Stream programming <u>for *performance*</u>, not *correctness*
  - Incremental tuning
  - Doesn’t require up-front holistic analysis
- Why is this important?
  - Many “streaming apps” include some unpredictable patterns

Specific Examples

- Raytracing
  - Unpredictable tree accesses
  - Software caching on Cell (Benthin ’06)
    - Emulation overhead, DMA latency for refills
  - Tree accesses have good locality on HW caches
- 3-D shading
  - Unpredictable texture accesses
  - Texture accesses have good locality on HW caches
  - Caches are ubiquitous on GPUs

Our Conclusions

- Caching performs & scales as well as Streaming
  - Well-known cache enhancements eliminate differences
- Stream Programming benefits Caching Memory
  - Enhances locality patterns
  - Improves bandwidth and efficiency of caches
- Stream Programming easier with Caches
  - Makes memory system amenable to irregular & unpredictable workloads
- Streaming Memory likely to be replaced or at least augmented by Caching Memory

Limitations of This Work

- Did not scale beyond 16 cores
  - Does cache coherence scale?
- Application scope
  - May not generalize to other domains
  - General-purpose != application-specific
- Sensitivity to local storage capacity
  - Intractable without language/compiler support

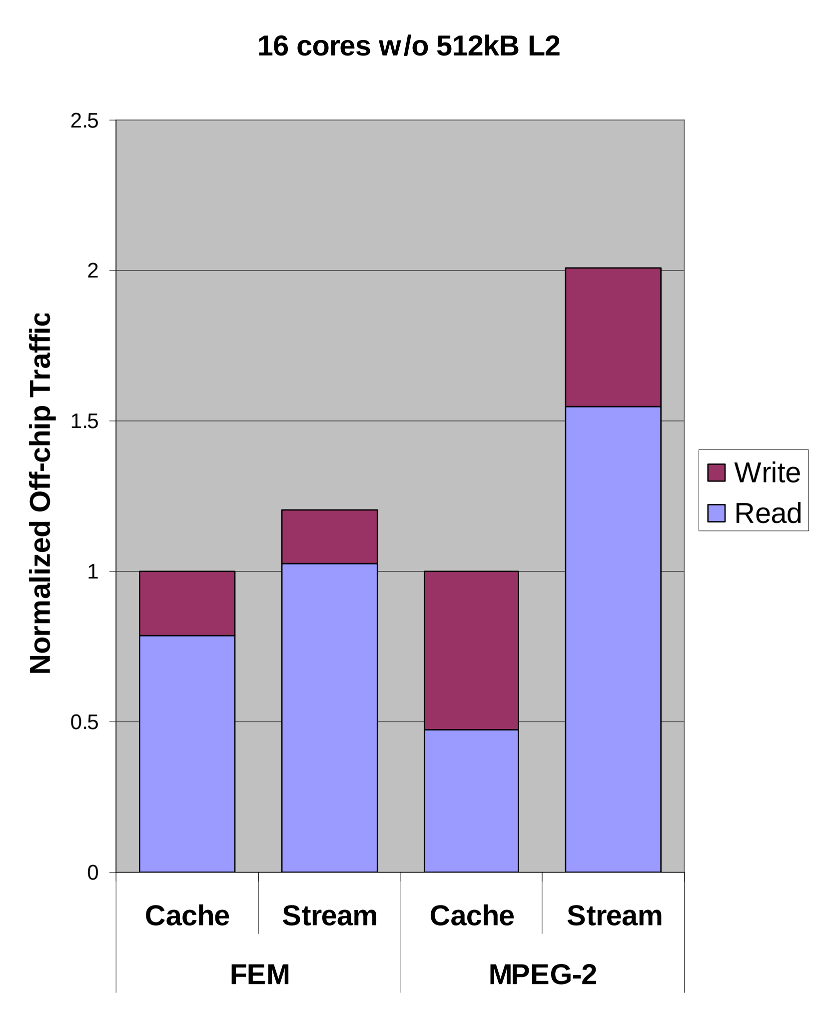
Future Work

- Scale beyond 16 cores
  - Exploit streaming SW to assist HW coherence
- Extend application scope
  - Generalize to other domains
  - Consider further optimizations
- Study sensitivity to local storage capacity
  - Introduce language/compiler support

Thank you!

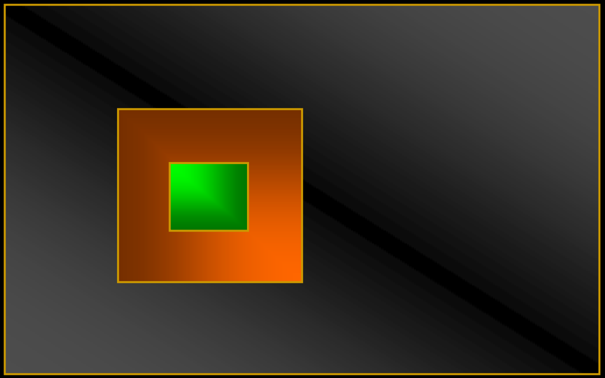
Questions?

Streaming Memory and L2 Caches

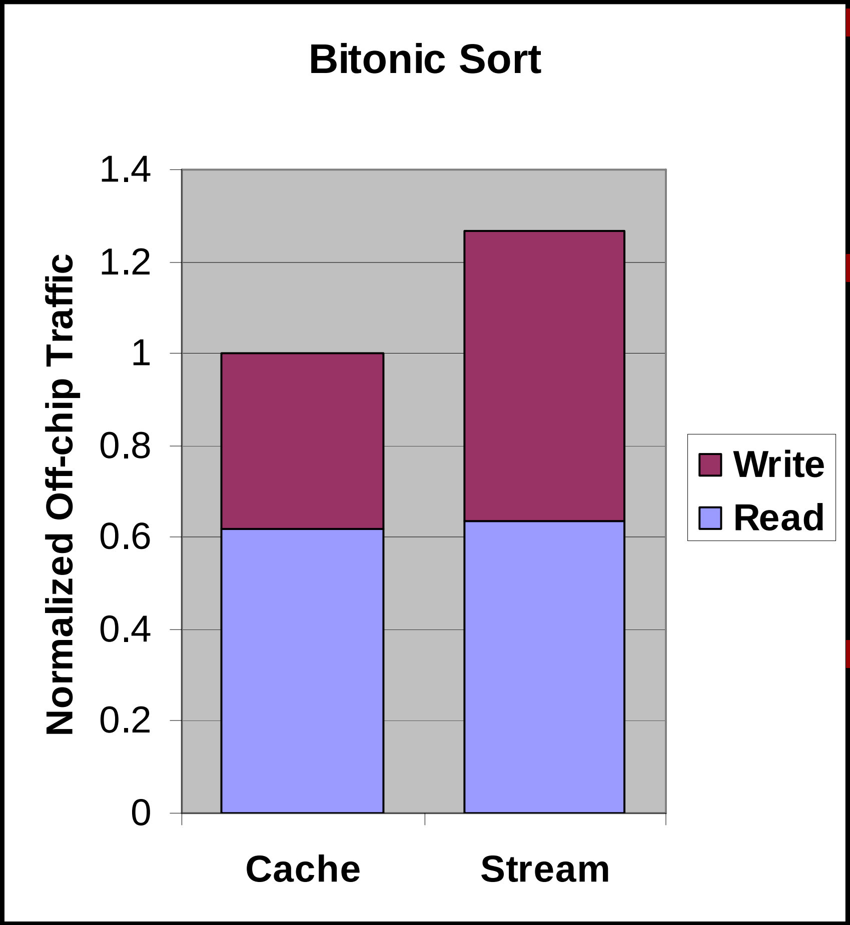


- L2 caches mitigate overfetch in Streaming apps
  - Unstructured meshes
  - Motion estimation
    - search window
    - reference frames

Motion Estimation



Streaming Memory Occasionally Consumes More Bandwidth



- The problem
  - Data-dependent write pattern
- Caching
  - Automatically track modified state
  - Write back only dirty data
- Streaming
  - Writes back everything
  - Programming burden & overhead to track modified state