MEMORY MANAGEMENT FOR MANY-CORE PROCESSORS WITH SOFTWARE CONFIGURABLE LOCALITY POLICIES

Jin Zhou and Brian Demsky
University of California, Irvine

Yousun Ko

Aims of this presentation

- Be aware of how memory behaviors can affect to the performance
  - Garbage Collection for heap memory
  - Free to attempt, expensive to maintain
- Know about the origin of the memory problems
  - Histories of problems and ad-hocs…
- Thinking about fundamental solutions

Introduction to the paper

"We present a NUMA-aware approach to garbage collection that balances the competing concerns of data locality and heap utilization to improve performance. … Our collector turns off cache coherence to eliminate the overhead of inter-cache coherence traffic."

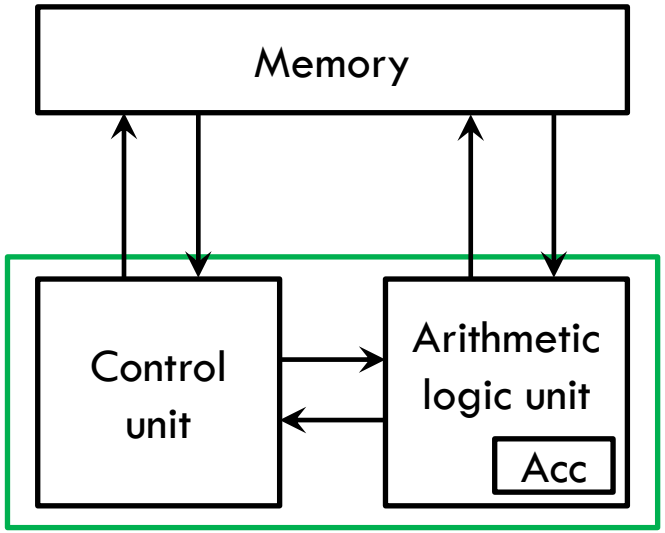
Introduction to the paper

"We present a NUMA-aware approach to garbage collection that balances the competing concerns of data locality and heap utilization to improve performance. … Our collector turns off cache coherence to eliminate the overhead of inter-cache coherence traffic."

??

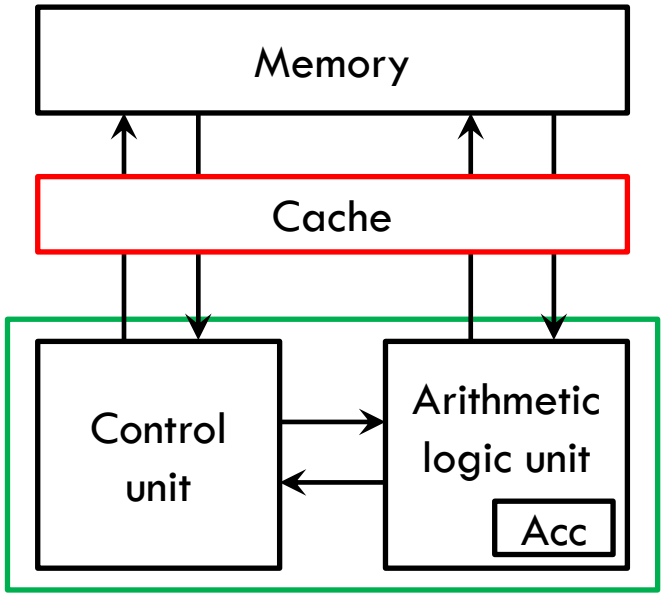
Von Neumann Architecture

- Implements a universal Turing machine and has a sequential architecture.
- Stored-program computer
- The word-at-a-time tube is built-in bottleneck
  - Most of tube traffic is due to names of data, operations etc.



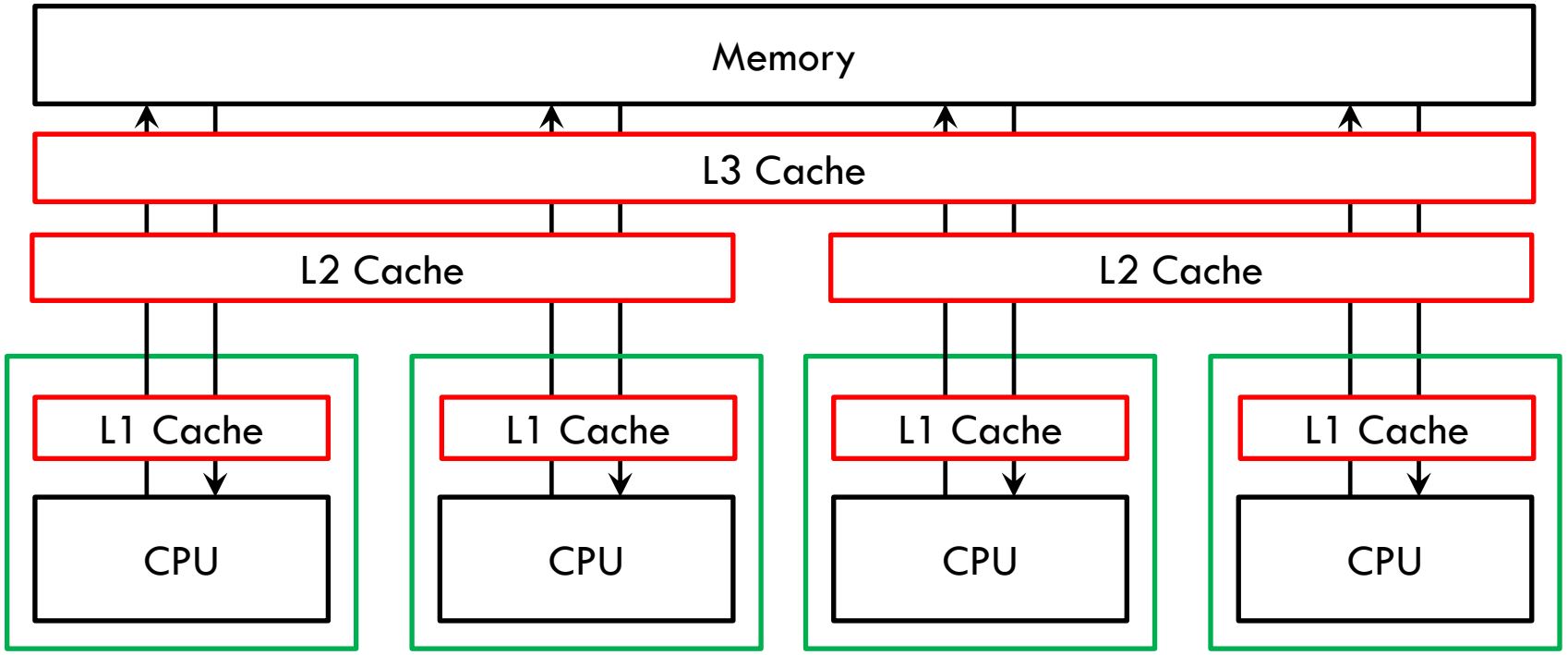
Cache

- Moore's Law: "Number of transistors on integrated circuits doubles approximately every 18 months"
- How about memory bandwidth?



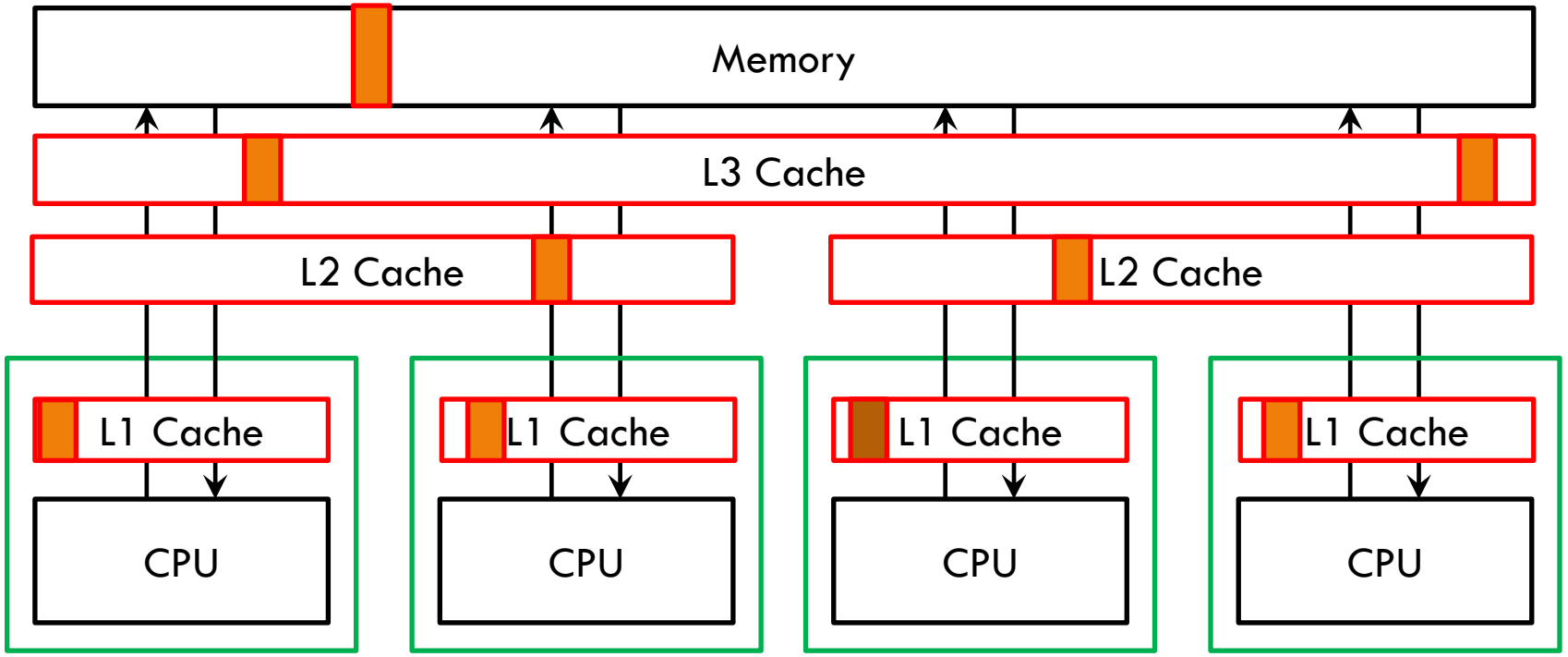
Caches

- Moore's Law: "Number of transistors on integrated circuits doubles approximately every 18 months"
- How about memory bandwidth?



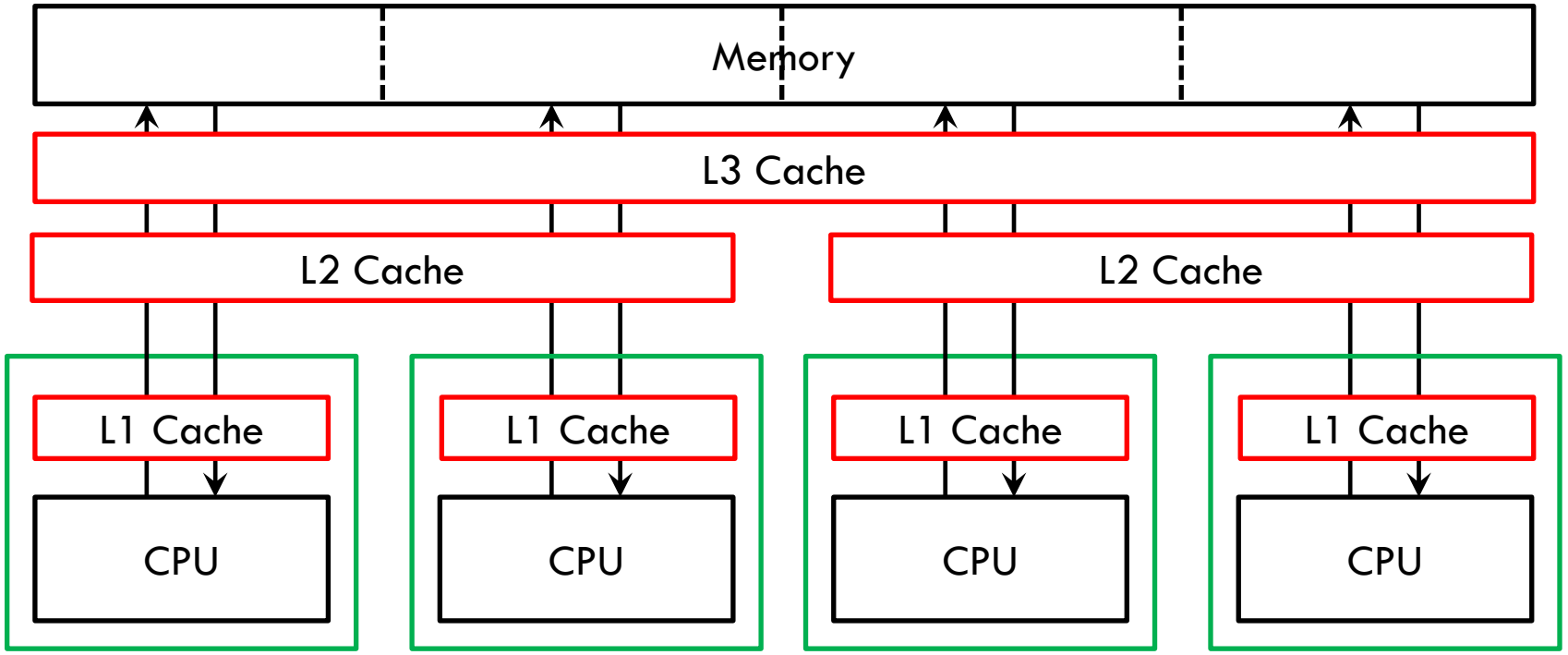
Cache Coherence

- Inconsistency due to duplication of data
- What if all processors are referencing a single data?

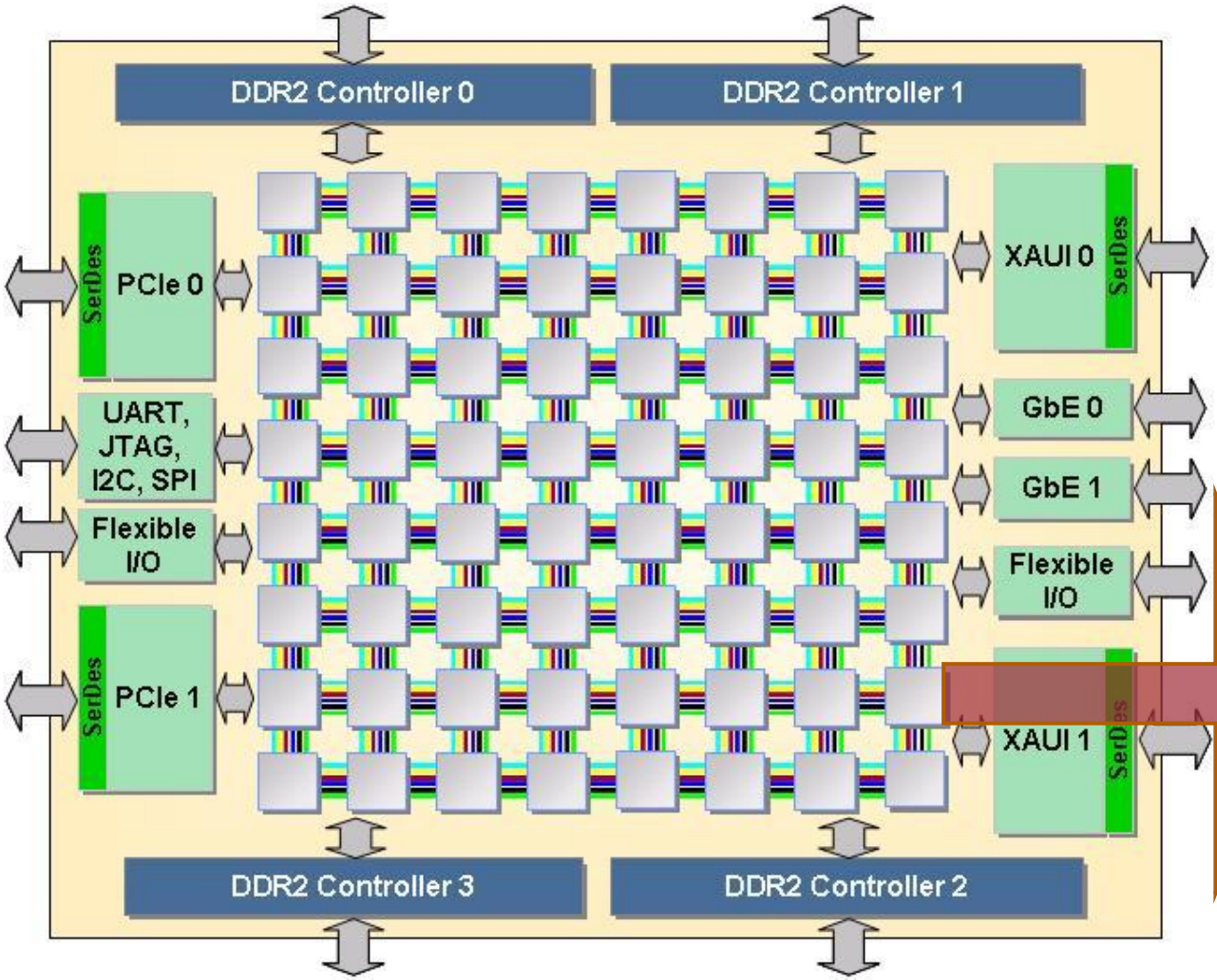


NUMA

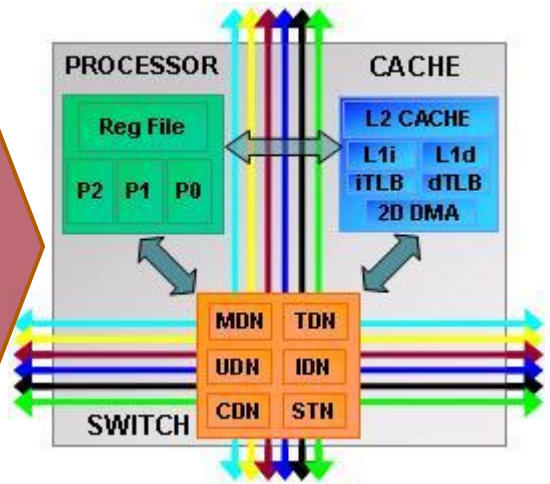
- NUMA (Non-Uniform Memory Architecture)
  - Memory access time depends on the memory location relative to a processor
  - Designed for multiprocessing



Evolving everyday…



64-core TILEPro64 (Tilera)

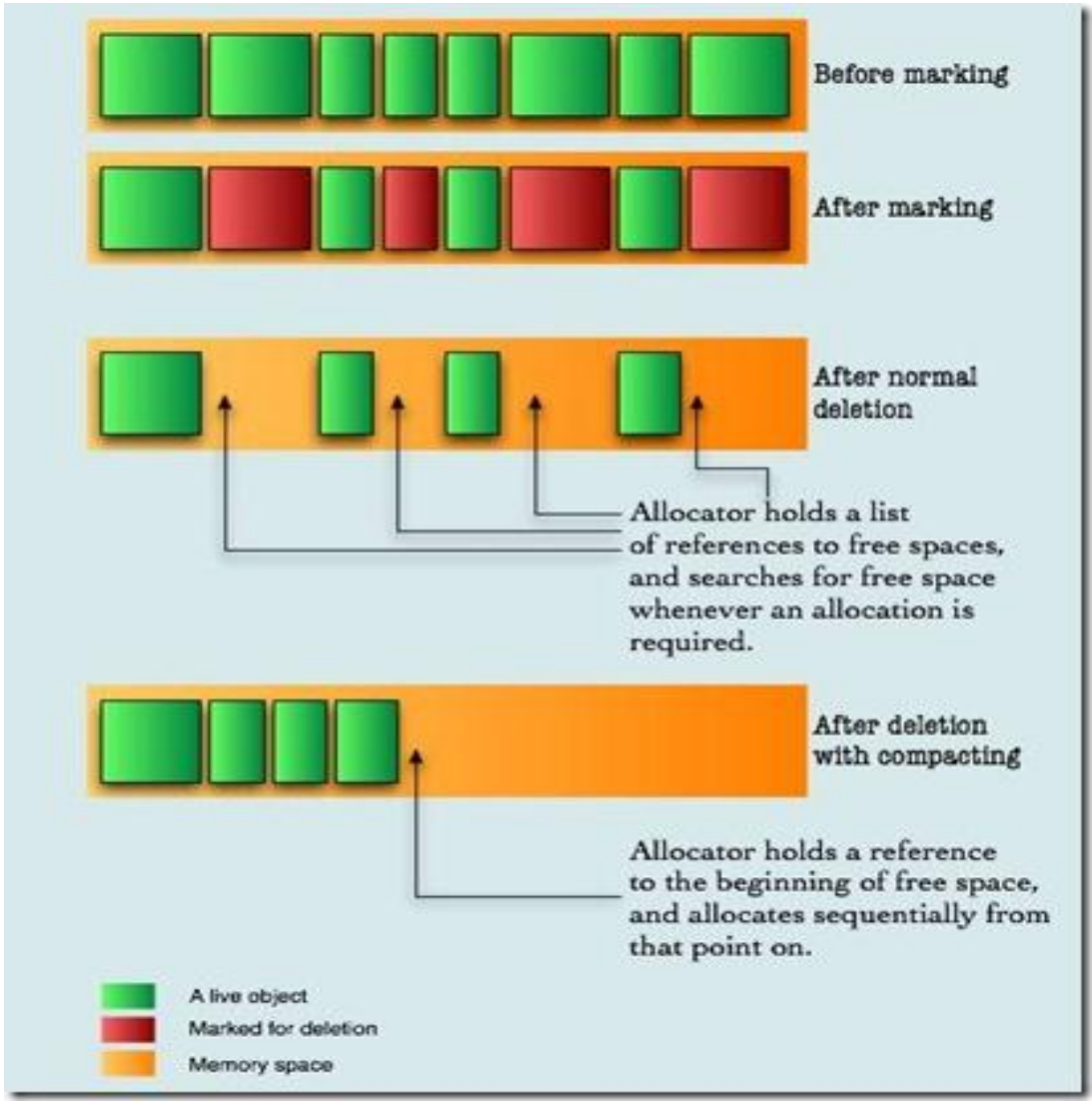


TILEPro64

- 64 700MHz cores arranged in a grid
- Each core is connected to a mesh network
- Each core has a local L1 and L2 cache
- Directory-based cache coherence
  - Keep track in a directory of which processors are caching a location and the state
  - Home core for a cache line maintains the directory for that line
- Software configurable supports:
  - Software can specify a core to home all of the cache lines of a given page
  - Home for cache lines on the page is computed by hashing the base address of the cache line
    - Distributes homes for a page's cache lines across many cores to avoid cache hoe spots by load balancing memory accesses across several caches.
  - Software can specify the page is not cached incoherently
  - Software can specify the page is not cached

Garbage Collection

- A form of automatic memory management
- Irregular memory behavior
- Consumes massive computing time
- Stalls program while garbage collection
- Still a hot issue on memory related academic venues



Contributions

- NUMA-aware Collector
  - Optimize both allocation and collection for NUMA
- Incoherent Garbage Collection
  - Turn off hardware cache coherence during collection
- Hybrid Heap Organization
  - Partition the heap to support independent collection
  - Maintain locality in a NUMA memory system
  - Defragmenting the entire heap to ensure free partitions
- Adaptive Cache Coherence Policies

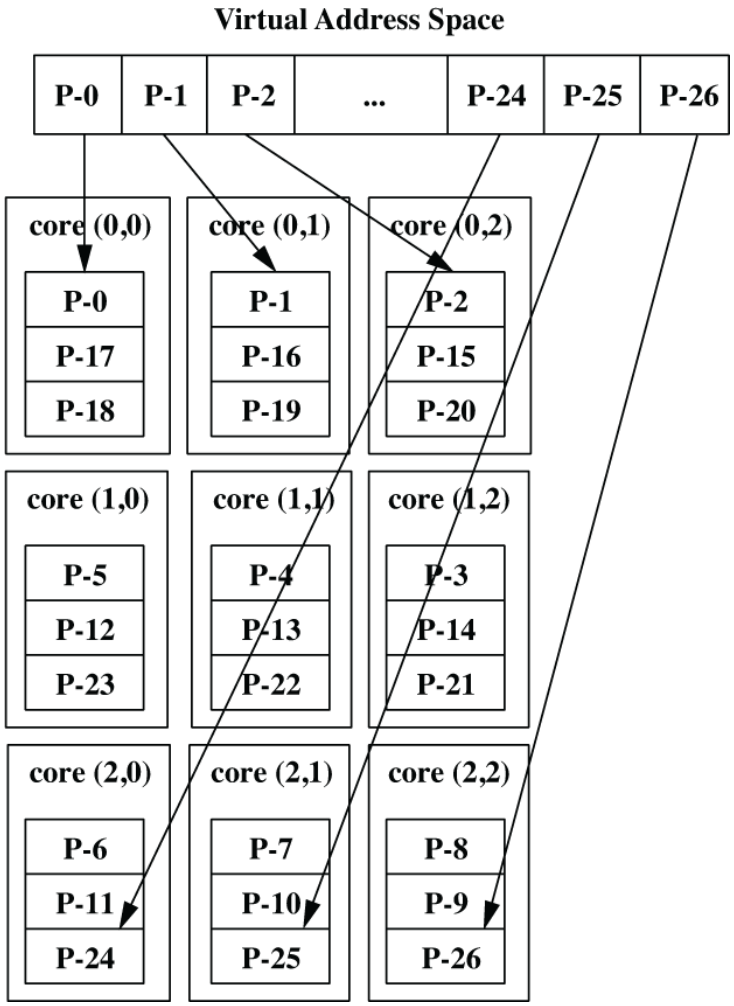
Caching policy considerations

- Effective reduction of cache size
  - Every cache line in the local cache that is remotely homed occupies two cache lines – one in the local core's L2 cache and one in the remote core's L2 cache. Performance benefit by shrinking the size of available cache.
- Home core capacity constraints
  - Size limitation of a home core due to a large, hot data structure. Hash-for-home may be required despite of performance degradation.
- Extra latency
  - Cache hit in a local L2: 8 cycles
  - Cache hit in a remote L2: 36 cycles
  - Cache miss in a local L2: 80 cycles
  - Cache miss in a remote L2: 123 cycles
- Hot spots
  - Limited capacity for servicing requests.

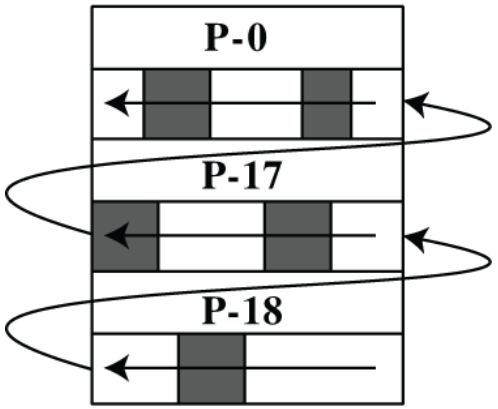
NUMA-Aware Collector

- Three phases of collector
  1) Marking Phase: mark live objects
  2) Planning Phase: plan to compact live objects
  3) Updating phase: update references through the heap
- Why NUMA bothers collector?
  - Most of collector are sequential
    - Parallel compaction?
  - Most of collectors are ill-suited for NUMA
    - Most of collectors regards the heap memory as global
  - Cannot simply partition a memory evenly to each processors
    - Fragmentation
- Then hybridize approaches!
  - Partition the global heap into contiguous core local heaps for locality while still compacting the global heap to recover large contiguous blocks

NUMA-Aware Collector



Heap Mapping



Garbage Collection Strategy on Core 0,0

Garbage Collection w/o Cache Coherence

- Cache coherence is disabled during garbage collection
- How to maintain memory consistency without cache coherence
  - No cache consistency issue for marking and planning phase
  - For updating phase, make a copy of the current memory status! (c.f., snapshot isolation)

Automatic Cache Tuning

- Measure usage patterns and tune homing policies to minimize hotspots and remote accesses
  - Similar to memory orchestration
- Continuous monitoring of memory accesses
  - To estimate how often each core accesses each page
- Compensate measurement for garbage collection
  - To pick best moves in the heap to reduce fragmentation
- Compute tuned caching policies
- Update caching policies
  - Update caching policies during collection

Caching Policies

- `All-hash` **policy**
  - Sets hash-for-home caching for all pages in the heap
  - Avoids hot spots enduring unnecessary inter-cache communication
- `Locally-homed` **policy**
  - Homes each partition on the core that collects the partition
  - Likely to have extremely hot page
- `Hottest` **policy**
  - Homes data on the core that accesses it most recently
  - Monitoring overhead
- `Adaptive` **policy**
  - Homes a page on a core if that core access the page the most and performs more than a quarter of the page's total access
  - Otherwise hash-for-home policy

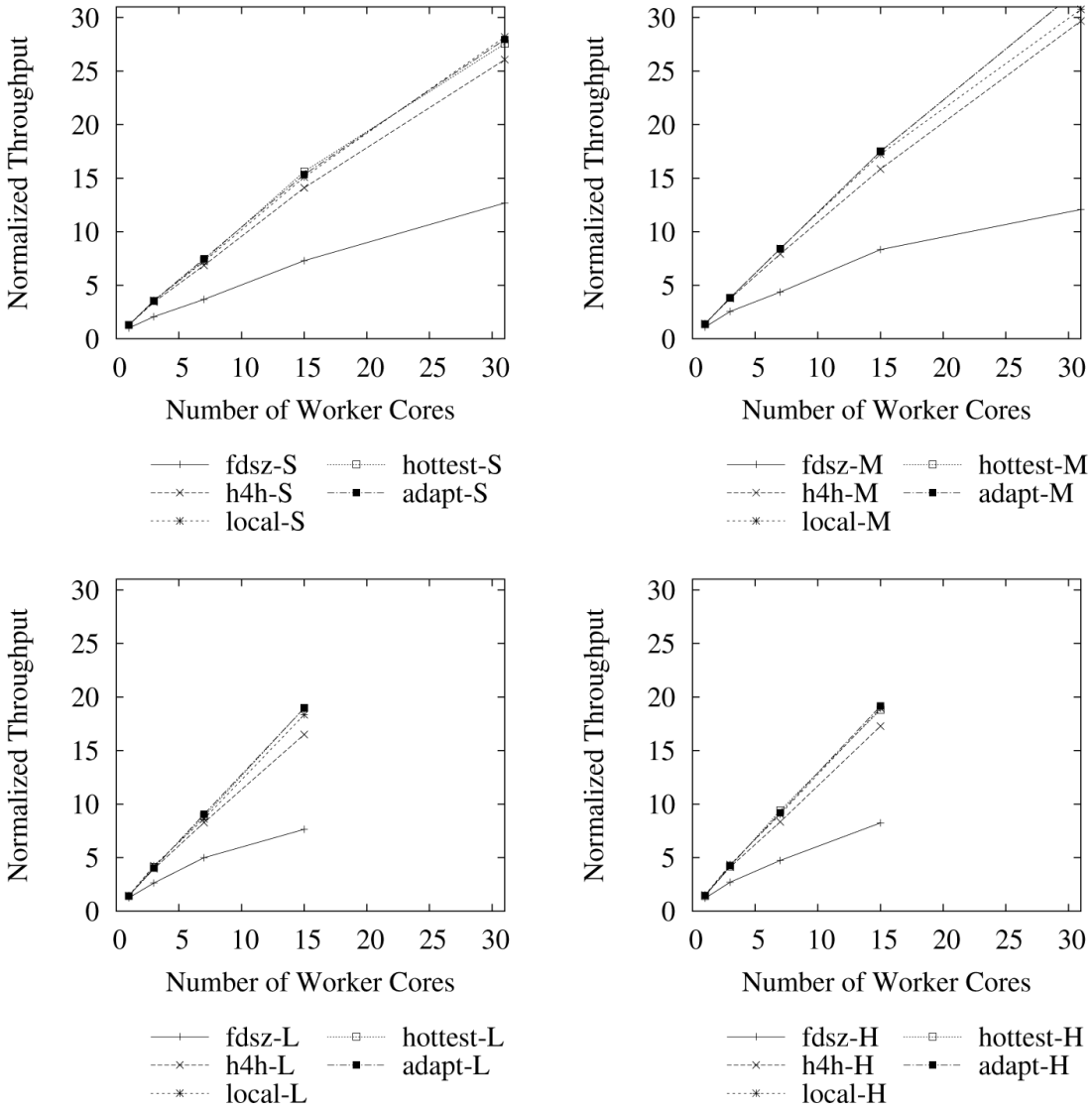
Evaluation

- Implemented own collector and adaptive cache tuning framework in own Java compiler and runtime system
- Compiler generates C code that runs on the TILEPro64 processor
  - Then what is the need of JVM?
- They say they had hard time to find proper benchmarks for this work because most of garbage collection benchmarks are sequential

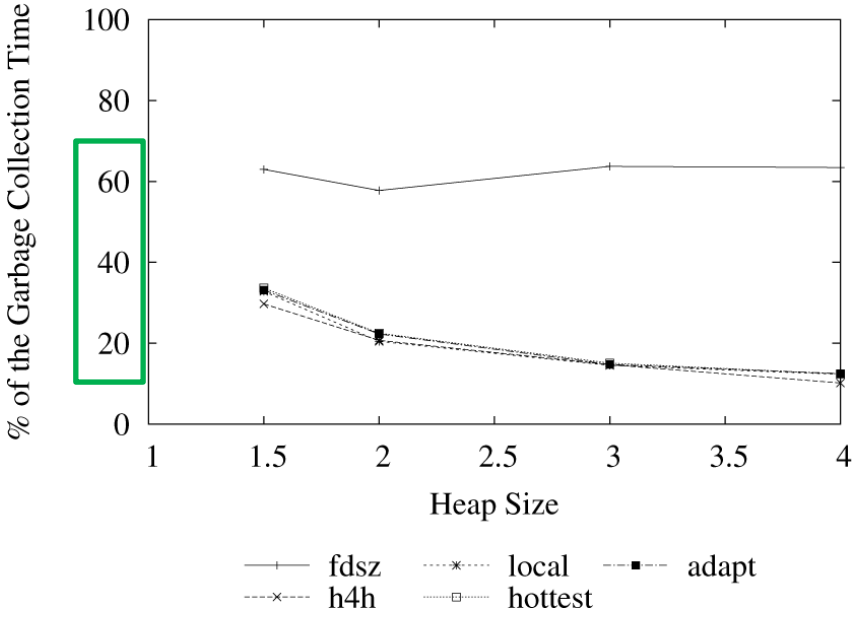
Versions of Experiment

- `fdsz`:
  - Mark compact collector of previous work
  - `hash-for-home` policy
- `h4h`:
  - `hash-for-home` policy
- `local`:
  - `locally-homed` policy
- `hottest`:
  - `Hottest` policy
- `adapt`:
  - `Adaptive` policy

Normalized Throughput



Gain from Disabling Cache Coherence



Percentage Time Spent in GC (15 worker cores versions)

Threads	h4h	local	hottest	adapt
1	12.5%	2.6%	11.0%	11.0%
3	15.6%	2.7%	15.0%	14.3%
7	20.3%	3.6%	30.8%	29.5%
15	21.4%	4.2%	42.4%	39.1%
31	24.1%	4.6%	47.4%	47.6%

Speedup from Turning Off Coherence During GC

Memory Access Count from Disabling Cache Coherence

		GC		APPLICATION	
		LOCAL	REMOTE	LOCAL	REMOTE
incoherent	h4h	151,113,750	135	154,434,824	71,517,748
	local	156,356,673	132	177,962,266	31,418,911
	hottest	157,920,761	283,812	223,777,594	19,795,376
	adapt	148,728,773	286,122	225,405,657	16,897,063
coherent	fdsz	17,665,795	97,889,718	65,902,083	32,575,816
	h4h	22,425,566	98,401,064	141,708,183	66,618,789
	local	136,820,278	9,993,122	176,170,786	29,276,631
	hottest	32,960,861	80,392,653	178,312,884	16,129,789
	adapt	31,298,317	75,769,027	168,107,310	16,134,823

Memory Access Counts

Speedup from Disabling Cache Coherence

Benchmark	Version	2	4	8	16	32	62
Mixed-Access	h4h	10.2%	16.9%	19.6%	20.8%	11.7%	18.0%
	local	0.3%	0.6%	0.7%	0.3%	-4.8%	-7.3%
	hottest	9.3%	9.1%	8.3%	7.2%	2.2%	3.3%
	adapt	7.4%	7.8%	8.7%	7.9%	2.1%	1.9%
GCBench	h4h	10.2%	16.0%	21.8%	22.5%	14.3%	12.2%
	local	-0.8%	2.3 %	3.9%	1.1%	-5.8%	-15.7%
FibHeaps	h4h	10.3%	18.5%	19.7%	22.1%	16.3%	20.3%
	local	5.5%	11.9%	8.5%	6.7%	2.9%	13.3%
BarnesHut	h4h	10.5%	14.1%	19.3%	22.3%	19.5%	14.6%
	local	-0.7%	-4.6%	5.5%	2.9%	-7.5%	-13.0%
LCSS	h4h	5.9%	10.8%	10.8%	12.1%	11.0%	9.0%
	local	0.0%	0.2%	0.6%	-0.4%	-4.2%	-6.6%
TSP	h4h	7.6%	13.2%	17.8%	17.3%	16.3%	14.0%
	local	-3.3%	0.0%	1.6%	0%	-2.9%	-7.6%
Voronoi	h4h	11.0%	17.2%	19.3%	23.1%	25.1%	14.1%
	local	0.7%	1.8%	2.8%	2.0%	1.5%	0.5%
RayTracer	h4h	6.3%	8.2%	3.6%	-1.8%	-14.1%	-12.6%
	local	-0.6%	-0.7%	-7.4%	-8.3%	-16.2%	-13.1%
	hottest	2.3%	6.7%	4.6%	1.3%	-2.2%	1.2%
	adapt	2.9%	6.8%	3.5%	1.2%	-5.0%	-0.5%

Speedup from Turning Off Coherence

Execution Times with difference benchmarks

Benchmark	Version	2	4	8	16	32	62
Mixed-Access	fdsz	32.43	68.89	79.22	251.68	232.47	439.46
	h4h	6.86	6.79	8.84	9.94	10.27	13.42
	local	5.55	5.62	7.43	13.30	23.47	45.61
	hottest	5.27	5.60	6.29	7.01	8.22	10.11
	adapt	5.30	5.60	6.40	7.33	8.15	9.01
GCBench	fdsz	5.26	9.14	16.07	30.35	63.11	108.47
	h4h	1.52	1.67	2.01	2.29	3.07	4.51
	local	1.40	1.41	1.52	1.55	1.78	2.27
FibHeaps	fdsz	0.75	0.88	1.01	1.08	1.24	1.44
	h4h	0.47	0.49	0.50	0.50	0.64	0.77
	local	0.41	0.43	0.41	0.41	0.53	0.55
BarnesHut	fdsz	9.41	11.32	13.22	14.54	16.17	19.10
	h4h	5.28	5.48	5.52	5.72	6.34	7.57
	local	5.07	5.37	5.25	5.24	5.65	6.51
LCSS	fdsz	2.58	3.00	3.47	3.83	4.37	5.20
	h4h	1.98	2.03	2.10	2.16	2.43	2.80
	local	1.98	1.95	1.94	1.94	2.03	2.26
TSP	fdsz	1.52	1.56	1.52	1.57	1.63	1.84
	h4h	1.36	1.42	1.52	1.38	1.40	1.49
	local	1.34	1.32	1.38	1.35	1.38	1.40
Voronoi	fdsz	1.20	1.39	1.75	2.04	2.90	4.78
	h4h	0.95	0.98	1.07	1.04	1.12	1.40
	local	0.90	0.94	0.91	0.92	0.95	1.12
RayTracer	fdsz	4732.28	4365.18	2646.02	1274.70	578.96	305.73
	h4h	3882.02	3674.44	2141.16	1073.34	554.46	308.73
	local	4313.14	3995.34	2023.85	1112.94	641.88	458.75
	hottest	3961.88	3828.35	2013.39	1041.31	524.44	309.27
	adapt	4298.56	3731.31	1925.55	992.21	523.69	284.71

Execution Times in 10^9 Clock Cycles. Lower is better.

Necessary-Evil: Memories

"Effectful programming (imperative programming) is known for employing side effects to make programs function. Effect-free programming (functional programming) in turn is known for its minimization of side effects."

-Wikipedia, "Side Effect"

- Use variables to imitate the computer's storage cells
  - control statements == jump and test instructions
  - assignment statements == fetching, storing, and arithmetic
    - Symbol := is linguistic von Neumann bottleneck!

Thank you!