DEX: Scaling Applications Beyond Machine Boundaries

Sang-Hoon Kim

FISS'19 @ KAIST

May 10, 2019

AJOU UNIV. 1973

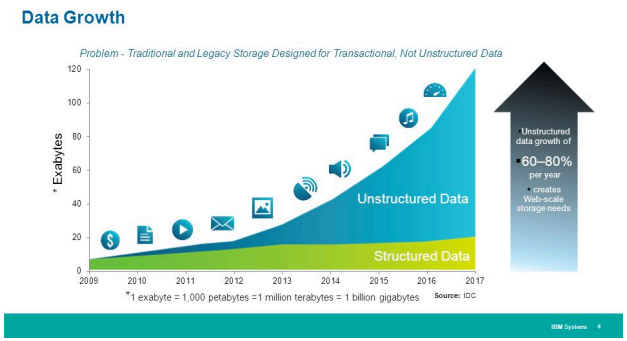
In collaboration with VIRGINIA TECH

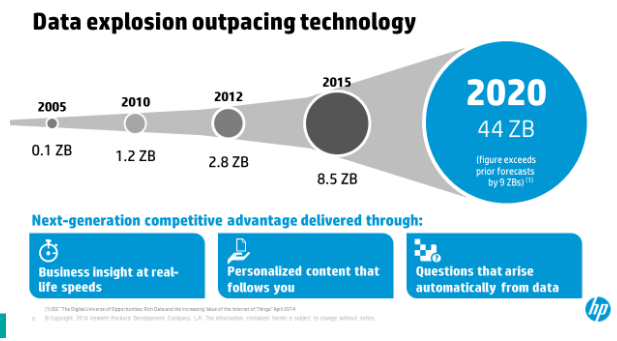
Trends in Data Centers and Business

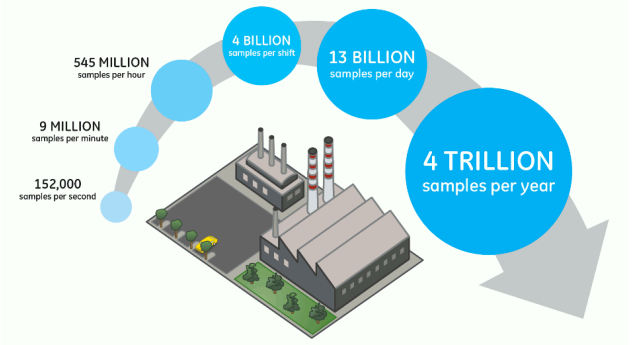
- Data volume keeps growing exponentially
- Machine learning and data analytics demand results within a short latency

➔ Increasing demand for high-performance ***scale-up*** machines

  – With extreme processing power and memory capacity in a single machine

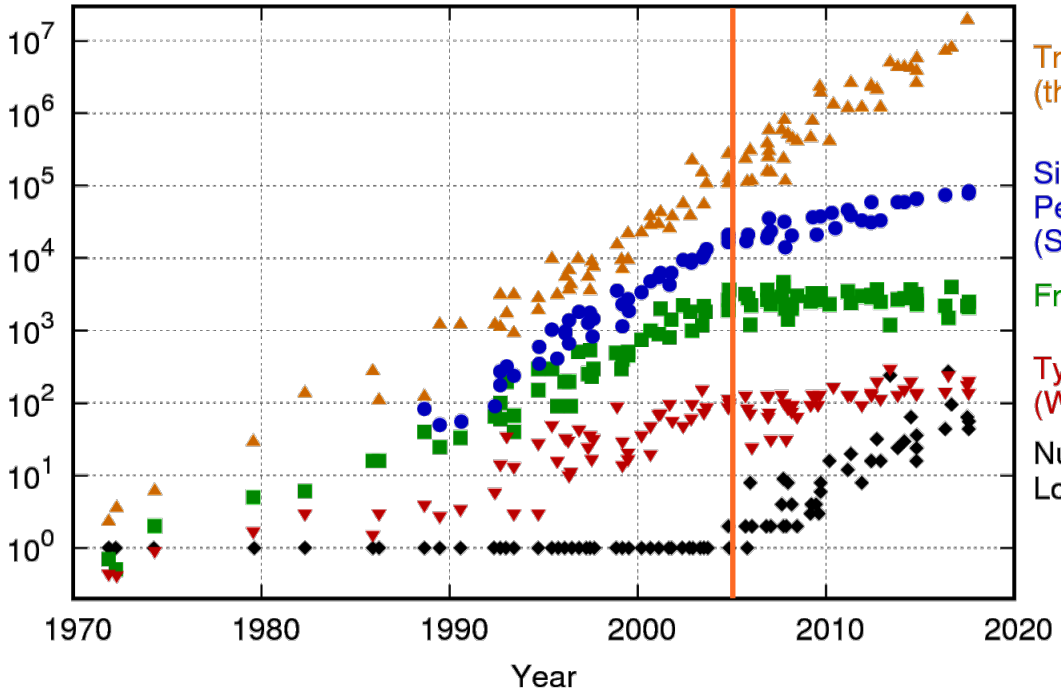






Microprocessor Trend

- 42 years of microprocessor trend data



Limited single thread performance
- Thermal and power budget
- Dark silicon effect

Fast increase in core counts

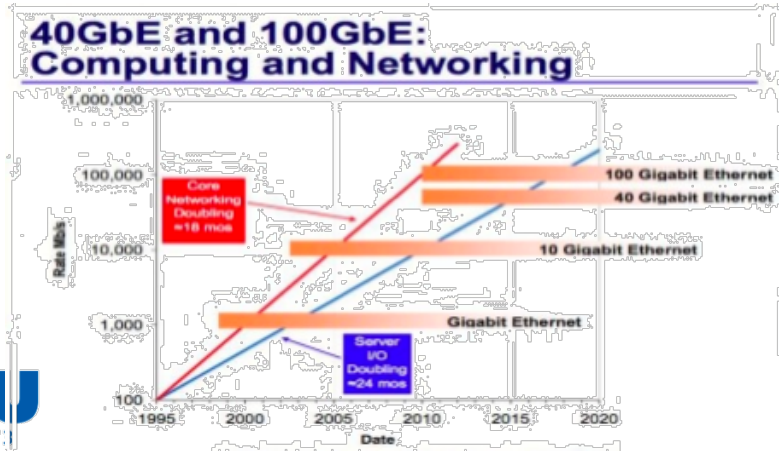
[https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data]

Increased complexities in scaling interconnects and coherence protocols
➔ Limit # of cores and processors in a single machine

Evolving of Network Technology

- Core network bandwidth is doubled in every 18 months
- InfiniBand, RoCE, Omni-Path, Gen-Z, …
  - High bandwidth (~600Gbps), low latency (~300ns)
    - Approaching to those of processor interconnect
  - Access remote memory directly through RDMA

➔ ***Blur*** the boundaries between local and remote machines

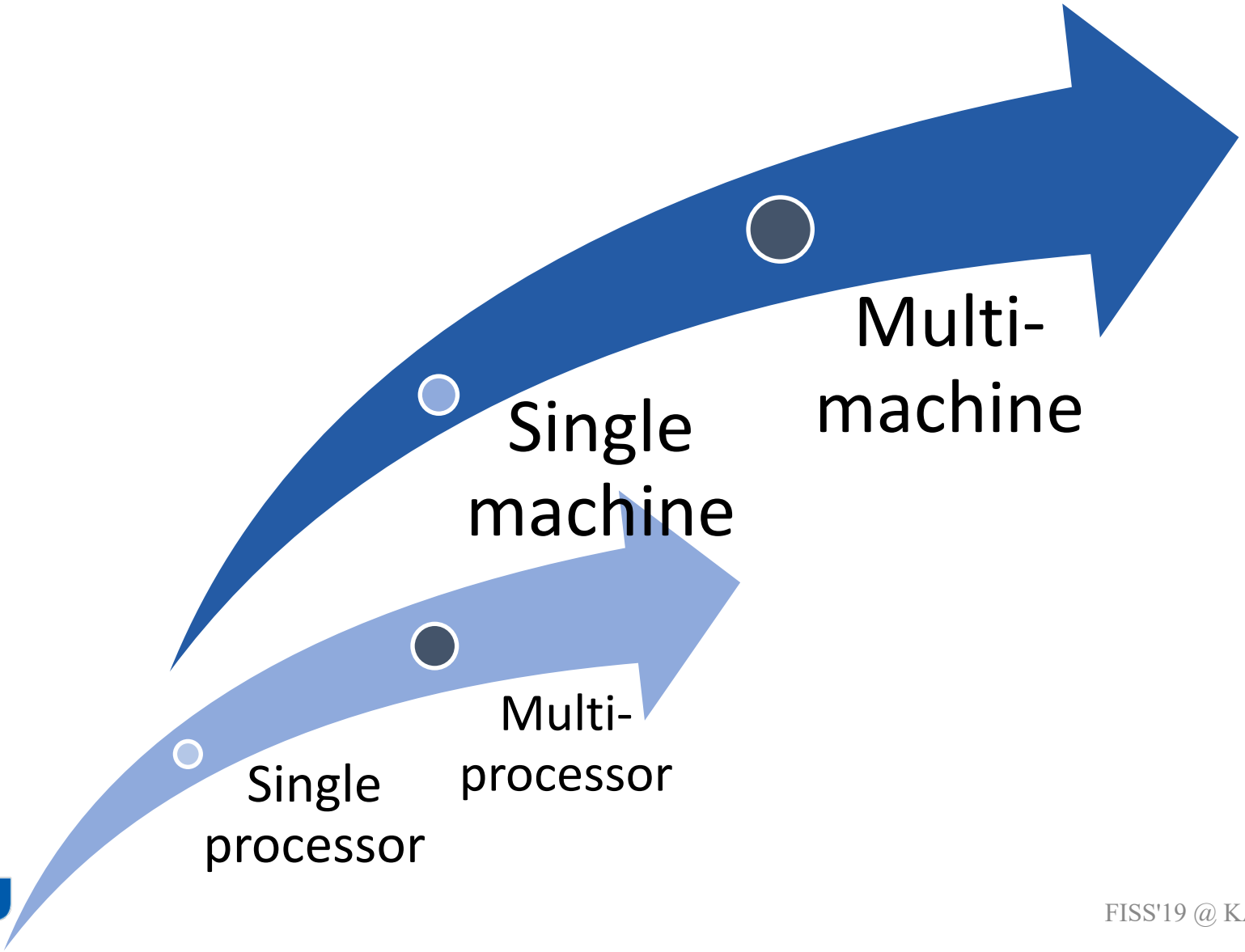


RoCE™

GEN Z

INFINIBAND™ TRADE ASSOCIATION

Towards Multi-Machine Era



Towards Multi-Machine Era

- Have been discussed for decades
  - Single system image (SSI)
  - Distributed shared memory (DSM) systems
- But, failed to win popularity
  - Bad performance due to slow network
  - Complicated programming models and memory semantics
  - Oftentimes require complete rewrite using custom APIs
  - Lack of flexibility in resource utilization

Lessons Learnt from the Past

- Programming model should be **intuitive** to understand
- **Should be applicable to existing applications with a minimal modification**
  - Cost(developers) >>>>>> cost(hardware)
- Many applications are (probably) already **scale-ready**
  - Multithreaded applications for multicore systems are common
  - Performance critical applications already implement data sharing and placement in mind
    - E.g., NUMA studies have provided great insights and techniques

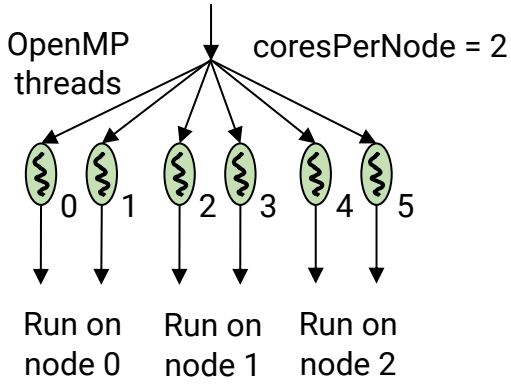
DEX

- **D**istributed Thread **Ex**ecution Environment
  - Better utilize scattered resource in a rack-scale setup
- Allow to **migrate/relocate threads to any node at anytime**
  - Intuitive way to scale application performance
- Support programming models *as-is*
  - Guarantee the sequential data consistency
  - Fully support native thread synchronization primitives
    - glibc POSIX thread(pthread) objects/operations and Linux futex
  - Eliminate application's burden

Using DEX

- To migrate a thread, just call a function

```
1  #pragma omp parallel default(shared)
2  {
3  #ifdef _DEX
4      int my_tid = omp_get_thread_num();
5      dex_migrate(my_tid / coresPerNode);
6  #endif
7      for (i = 0; i < NQ; i++) {
8          qq[i] = 0.0;
9      }
10     ...
```

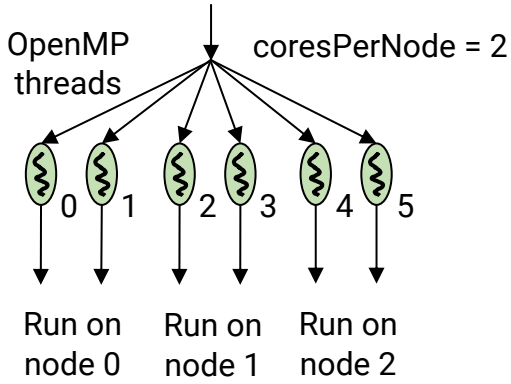


Application		Multithread	Mod. LoC	Application		Multithread	Mod. LoC
Simple	Grep	pthread		PARSEC	Blackscholes	pthread	
	Kmeans	pthread					
NPB	Common			Polymer	Common		
	BT	OpenMP (15)			BFS	pthread	
	EP	OpenMP (1)			BP	pthread	
	FT	OpenMP (7)			PageRank	pthread	

Using DEX

- To migrate a thread, just call a function

```
1  #pragma omp parallel default(shared)
2  {
3  #ifdef _DEX
4      int my_tid = omp_get_thread_num();
5      dex_migrate(my_tid / coresPerNode);
6  #endif
7      for (i = 0; i < NQ; i++) {
8          qq[i] = 0.0;
9      }
10     ...
```



Application		Multithread	Mod.	LoC	Application		Multithread	Mod.	LoC
Simple	Grep	pthread	+2		PARSEC	Blackscholes	pthread	+2	
	Kmeans	pthread	+2						
NPB	Common				Polymer	Common		+18	-18
	BT	OpenMP (15)	+38	-4		BFS	pthread	+6	-2
	EP	OpenMP (1)	+2			BP	pthread	+12	-11
	FT	OpenMP (7)	+28	-7		ageRank	pthread	+7	-5

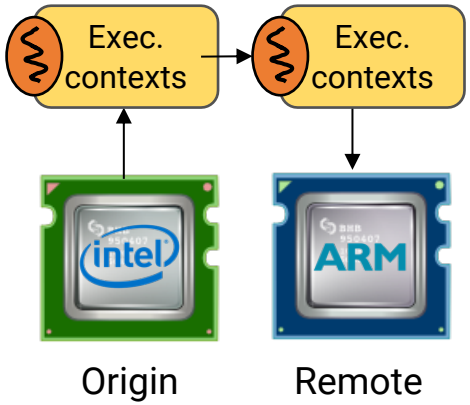
For converting multiple parallel regions

Mostly for replacing NUMA APIs to their general counterparts

Took ~3 days for a Ph.D. student

Migrating Execution

- Essentially performing context switch across machines
- At **origin**: Save the execution context
  - Leverage in-kernel thread/memory information saved during system call entry
  - A user-level runtime collects the register set



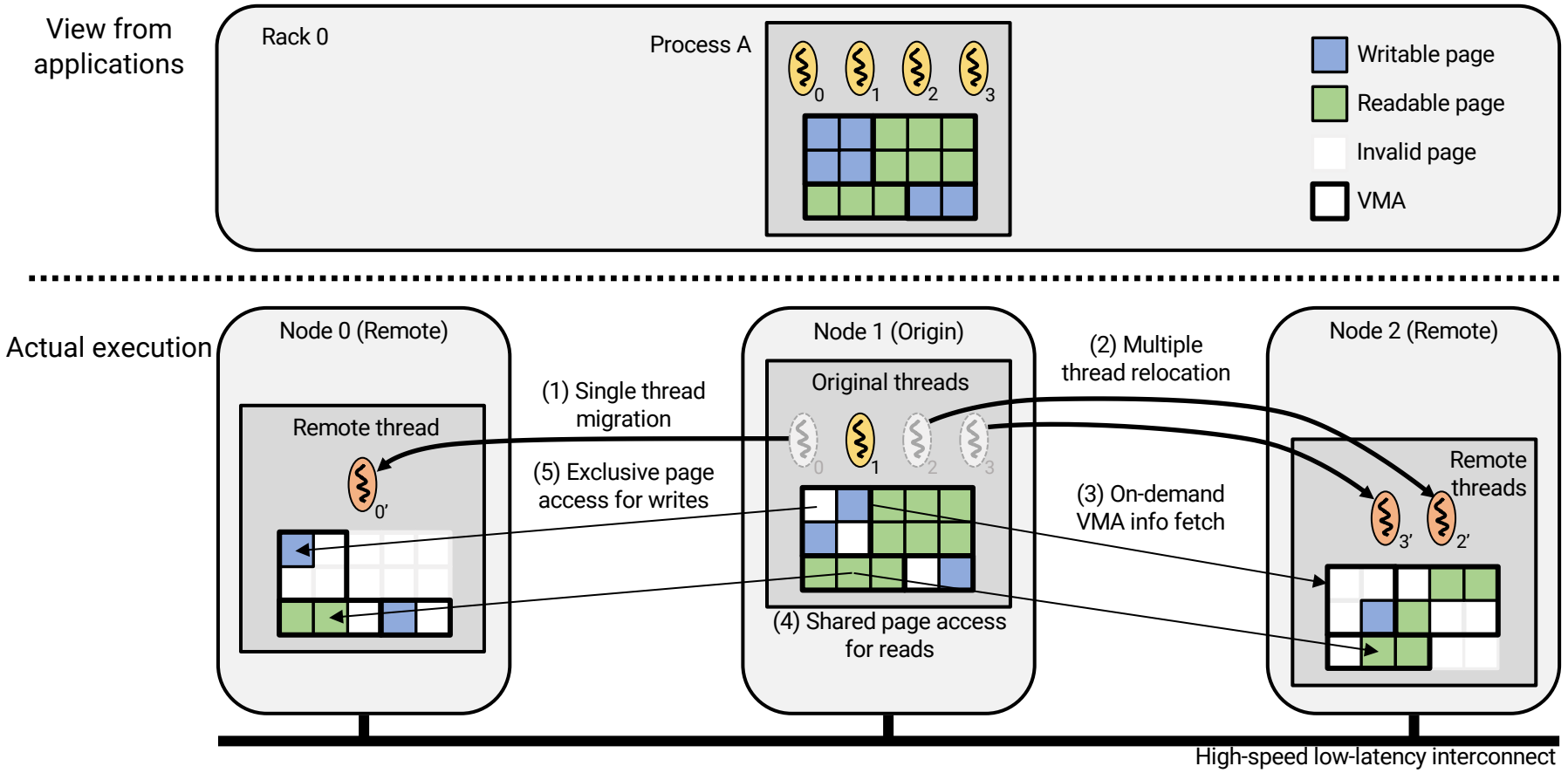
- At **remote**: Restore the context on a thread
  - Create a user thread and setup in-kernel data structures
  - Inject the execution context
  - Return from the kernel space

  ➔ Resume execution as if returned from system call

Migrating Execution

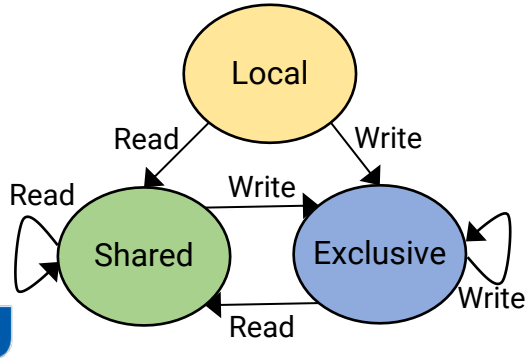
- Challenge 1: Callee-save registers and FP/XMM registers
  - Not saved during system call entry
  - User-level runtime collects those registers at the origin
  - Restore at remotes before resuming the original code
- Challenge 2: Frequent migration
  - Create a remote worker and fork remote threads from it
    - 812 us → 236 us
  - Leave memory data behind after back-migration

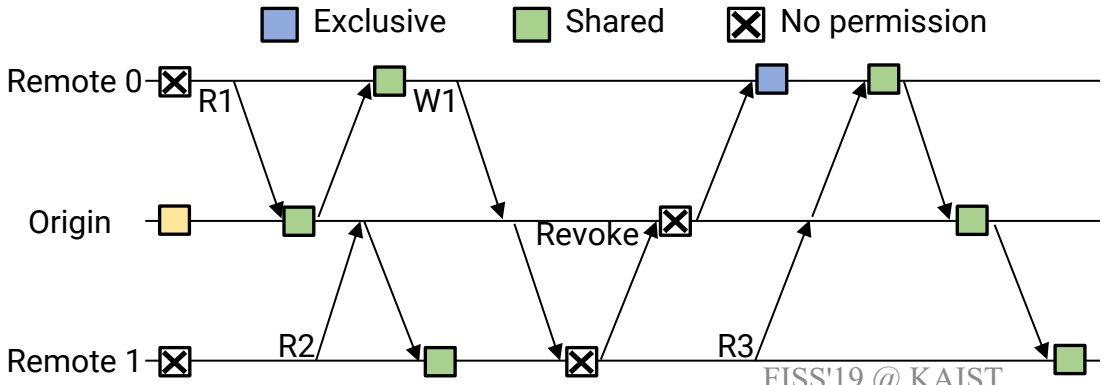
Distributed Thread Execution in DEX



Providing Consistent Memory View

- The origin controls the ownership and data
  - Origin owns all pages in the beginning
  - Contact origin to get ownership and data for pages
- Read-replicate, write-invalidate protocol at page granularity
  - To exploit the common cases in memory-intensive workloads
- Implemented in the virtual memory system in operating system
  - Transparent to the application's perspective





Taming Concurrent Page Faults

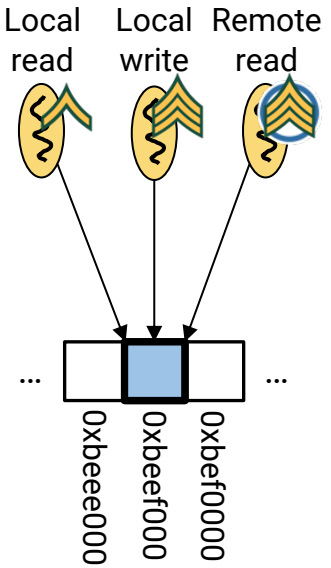
- Optimistic fault handling in Linux necessitates fault handling rollback

- Coalesce multiple faults and handle with a single operation

- **Leader** : First thread initiating a page fault handling for a page at a moment
  - Actually execute the fault handling operation for the page
    - E.g., bring the page from remotes, fix up page table, flush TLB, …

- **Followers**: Utilize the leader's outcome
  - Wait for the completion of the leader's fault handling

- Otherwise, wait or retry



Providing Consistent Memory View

- Challenge 1: Per-page metadata management
  - Owner(s), permission, copy-on-write status (for data section)
    - Index with MSBs of virtual addresses with a radix tree
  - The faults that are being processed
    - Hashmap
- Challenge 2: VMA synchronization
  - malloc and free (essentially mmap and munmap) manipulate VMAs
  - Threads tend to have localized VMA accesses
  - Lazy VMA Synchronization
    - Perform VMA operations at the origin
    - Only the result of munmap that shrinks VMA is broadcasted

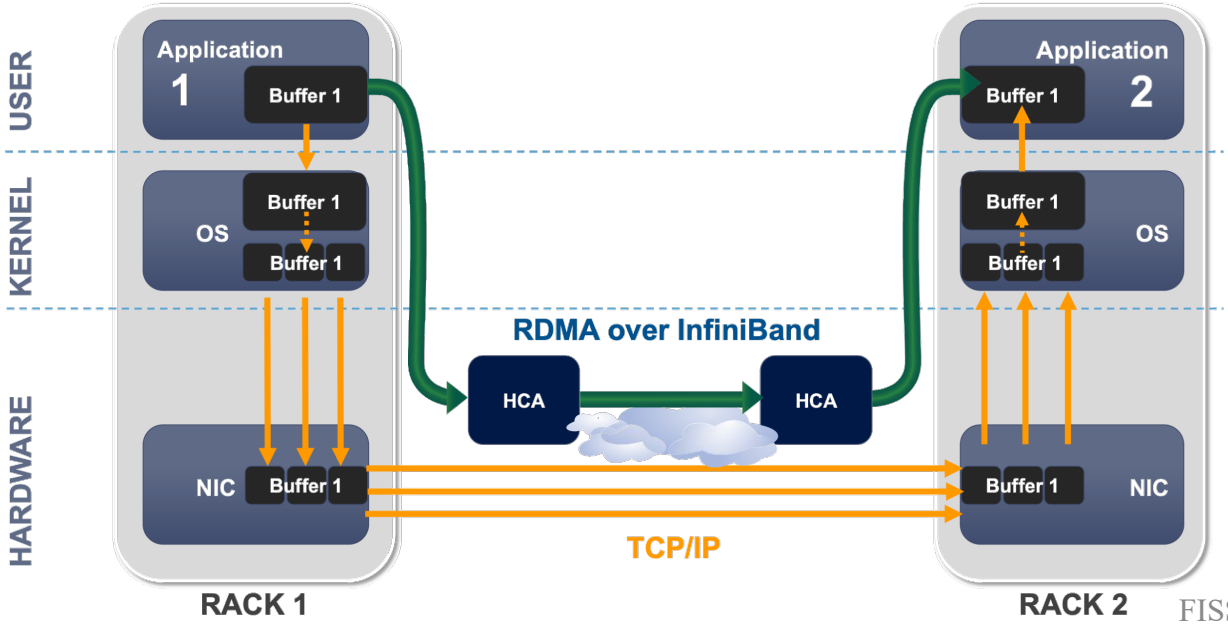
Kernel Features Across Machines

- Practically infeasible to reimplement every kernel feature distributed

- Adopt **work delegation** pattern
  - Leverage the original threads at the origin that are put to sleep
  - Remote threads forward stateful operations to their original threads
    - Futex, socket/file descriptors, …
  - The operations are performed at the origin serialized
  - Result is forwarded to the remote
    - One InfiniBand roundtrip incurs only negligible overhead
  - Can systematically implement as a wrapper layer

Leveraging RDMA

- RDMA over InfiniBand is different from traditional communication over TCP/IP
  - I/O buffers should be ***mapped*** to an DMA-able address space
  - Should be also ***registered*** to a memory region for remote access
  - Mapping and association take long, incurring a high overhead on latency-sensitive memory traffic



[http://www.hpcadvisorycouncil.com/events/2014/swiss-workshop/presos/Day_1/1_Mellanox.pdf]

Leveraging RDMA

- One approach does not fit all
  - Large messages (e.g., page data) are mapped/registered on demand
    - Memory copy >>> DMA map/RDMA region registration
  - Small messages (e.g., control messages) use a pre-allocated, pre-DMA-mapped, and pre-registered circular buffer
    - Memory copy <<< DMA map/RDMA region registration
- Asynchronous messaging
  - Increase parallelism in handling requests

Reducing False Page Sharing

- Behavior analysis tool helps to identify false page sharing
  - Analyze page fault events collected in profiling mode
  - Pinpoint to the location in code
    - # of faults, type of faults, type of program objects

Program Object	Number of Accesses	R	W	I
stack/mmap	28068523	12589437	4053812	11425274
heap	2198224	1265049	464711	468464
_dlfcn_hooks	13184	2261	5646	5277

Location	Number of faults	R	W	I
numa-PageRank.C:81	14010634	10664750	0	3345884
numa-PageRank.C:145	11661205	0	3850227	7810978
numa-PageRank.C:212	2140582	1450619	0	689963
graph.h:52	1574677	1572864	0	1813
polymer.h:2395	536313	0	507966	28347
utils.h:301	124023	117731	0	6292
utils.h:256	123871	0	117731	6140
graph.h:46	36732	36622	0	110

Reducing False Page Sharing

- Behavior analysis tool helps to identify false page sharing
  - Analyze page fault events collected in profiling mode
  - Pinpoint to the location in code
    - # of faults, type of faults, type of program objects

Application		Multithread	Mod. LoC		Application		Multithread	Mod. LoC	
Simple	Grep	pthread	+2		PARSEC	Blackscholes	pthread	+2	
	Kmeans	pthread	+2						
NPB	Common				Polymer	Common		+18	-18
	BT	OpenMP (15)	+38	-4		BFS	pthread	+6	-2
	EP	OpenMP (1)	+2			BP	pthread	+12	-11
	FT	OpenMP (7)	+28	-7		PageRank	pthread	+7	-5

Took 3 days for a Ph.D. student

Reducing False Page Sharing

- Behavior analysis tool helps to identify false page sharing
  - Analyze page fault events collected in profiling mode
  - Pinpoint to the location in code
    - # of faults, type of faults, type of program objects

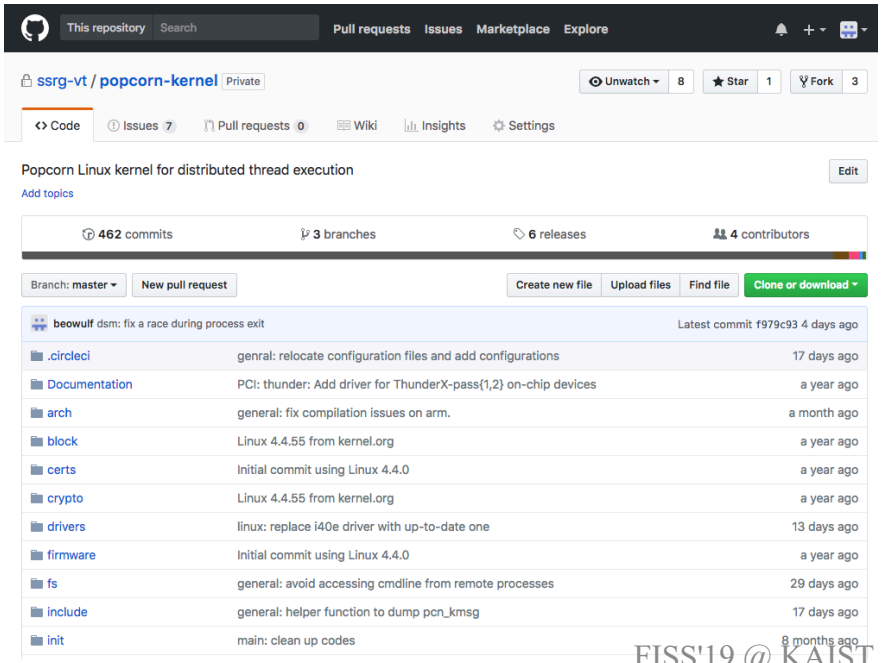
Application		Multithread	Mod.	LoC	Application		Multithread	Mod.	LoC
Simple	Grep	pthread	+21	-12	PARSEC	Blackscholes	pthread		
	Kmeans	pthread	+6	-3					
NPB	Common		+1	-1	Polymer	Common		+86	-67
	BT	OpenMP (15)	+5	-2		BFS	pthread	+10	-4
	EP	OpenMP (1)	+2	-1		BP	pthread	+13	-10
	FT	OpenMP (7)	+1	-1		PageRank	pthread	+32	-30

Took 4 days for the Ph.D. student

Implementation

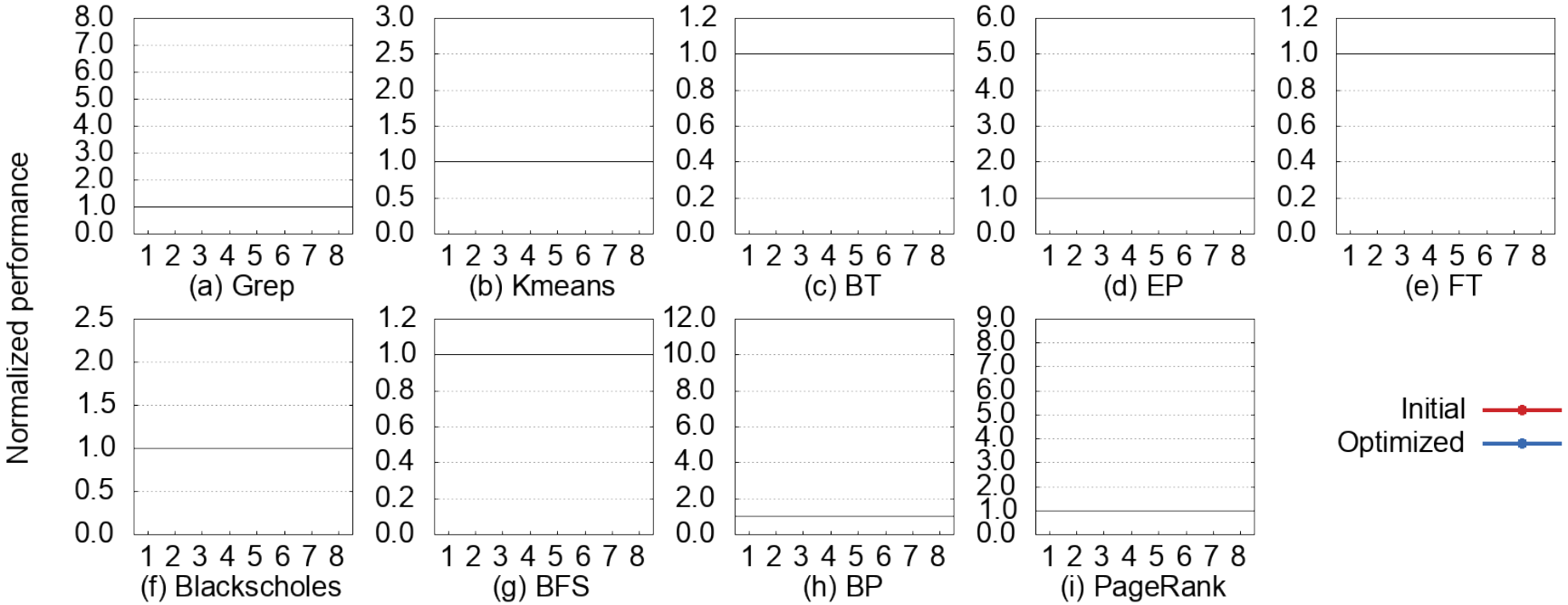
- Based on Linux kernel v4.4.137
  - Working on x86-64 and arm64
  - Use InfiniBand VERB for control messages and RDMA for pages
- Available at https://github.com/ssrg-vt/popcorn-kernel

Component	Lines of code
Kernel	10,114
Common libraries	1,202
Thread migration	2,058
Page migration	2,561
VMA handling	1,077
Communication layer	3,028
User-space library	370
Total	**10,484**



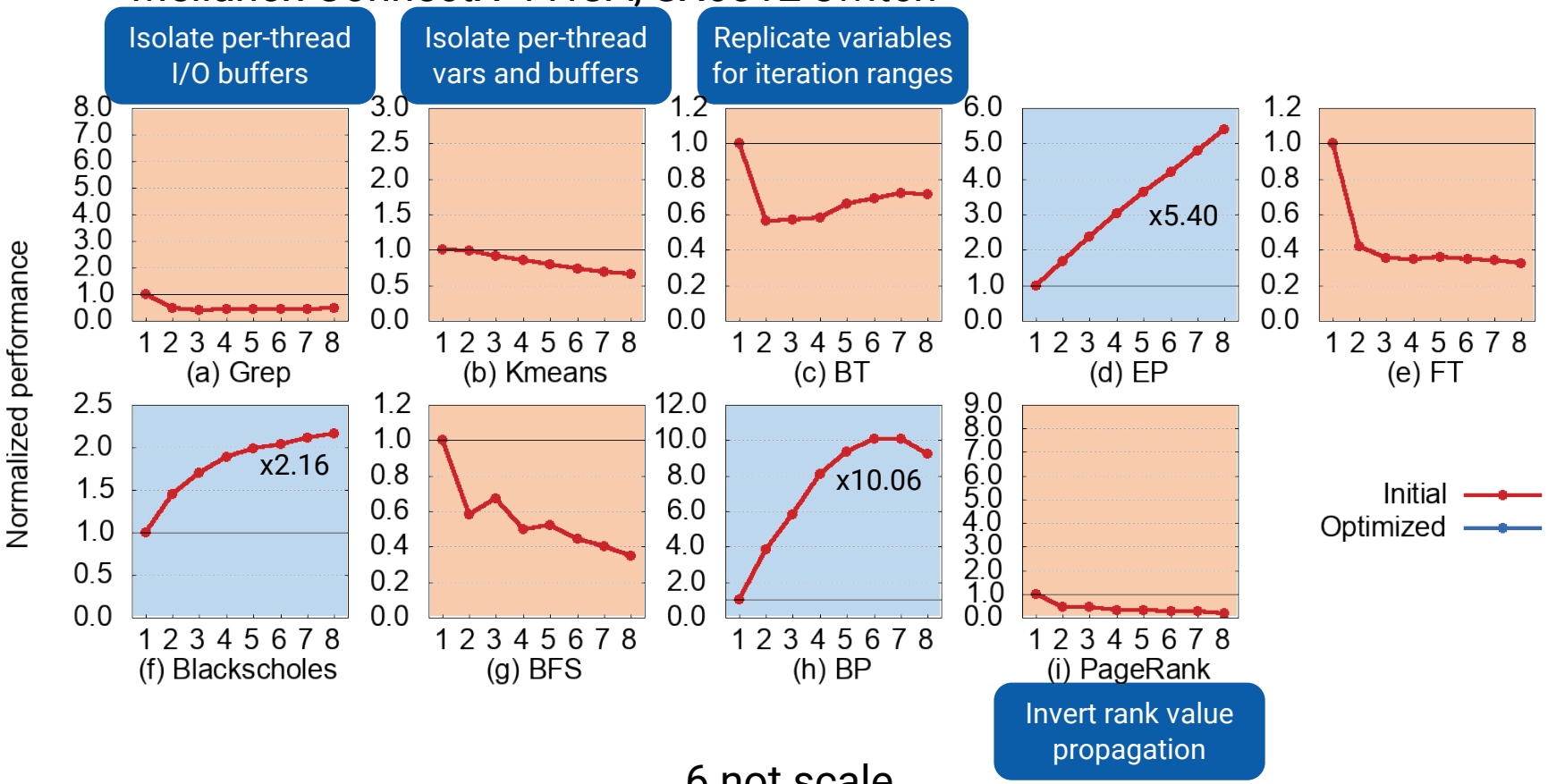
Evaluation

- 8x Silicon Mechanics R353.v6 (Intel Xeon Silver 4110 (8 cores), 48GB)
- Mellanox ConnectX-4 HCA, SX6012 switch



Evaluation

- 8x Silicon Mechanics R353.v6 (Intel Xeon Silver 4110 (8 cores), 48GB)
- Mellanox ConnectX-4 HCA, SX6012 switch

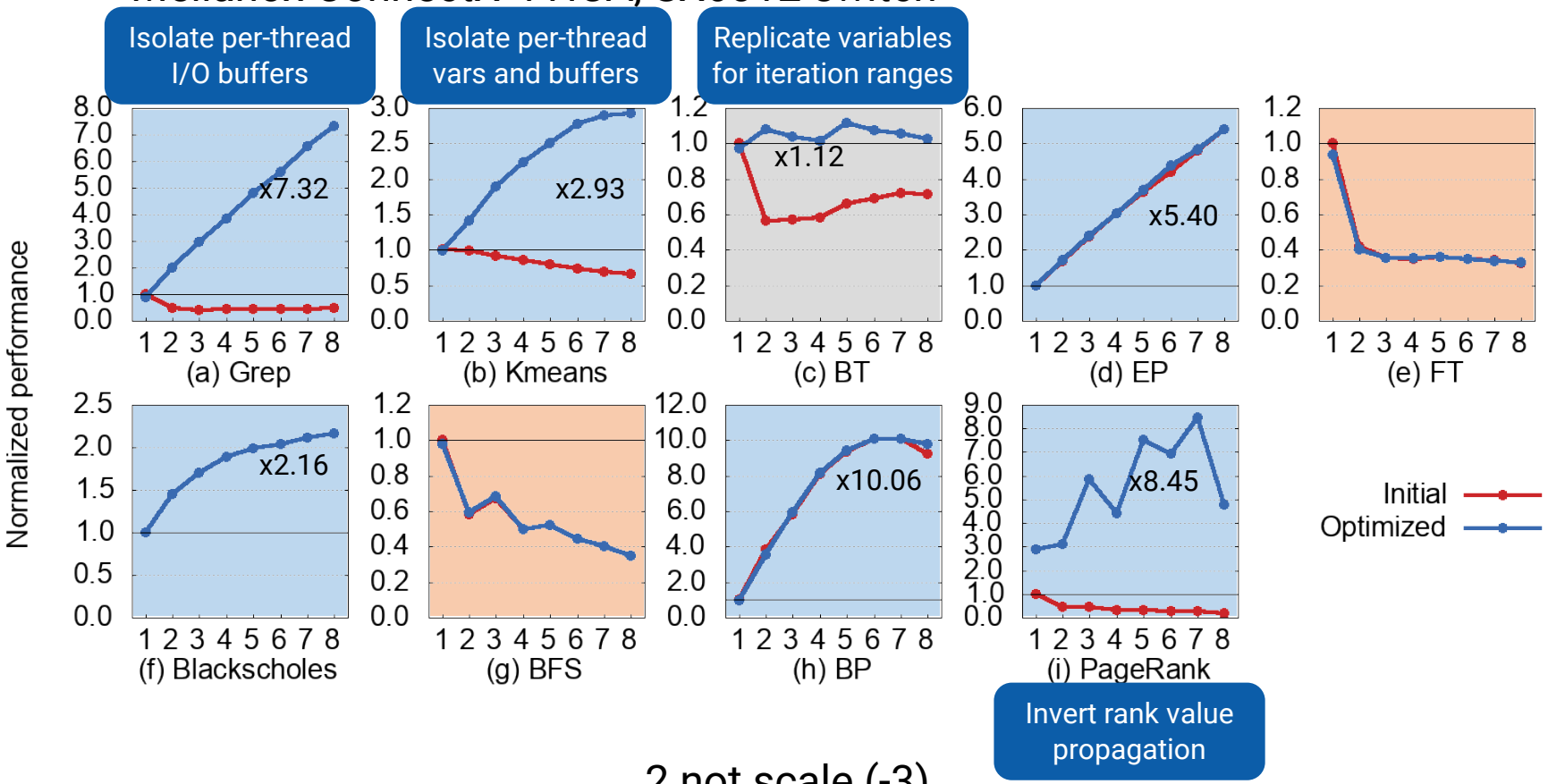


6 not scale

3 scale-ready

Evaluation

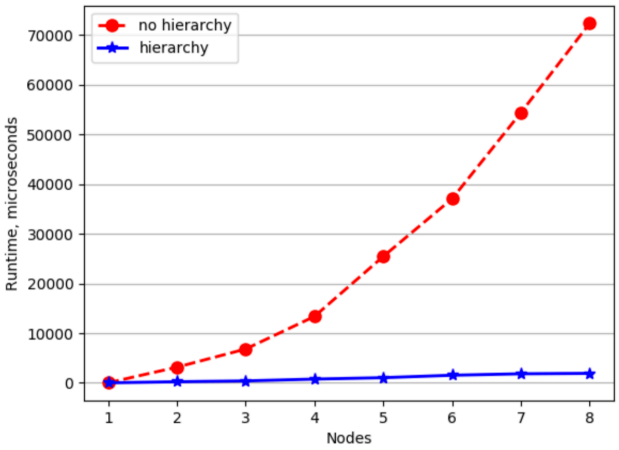
- 8x Silicon Mechanics R353.v6 (Intel Xeon Silver 4110 (8 cores), 48GB)
- Mellanox ConnectX-4 HCA, SX6012 switch



2 not scale (-3)
1 perform better (+1)
6 scale linearly (+2)

Can Further Improve Runtime?

- **libMPNode** to accelerate OpenMP applications on DEX
  - PMAM co-located with PPoPP'19
- Hierarchical synchronization and thread organization
  - Per-node leader performs global operations whereas others perform local operations
  - Process reduction in the same way



Barrier performance

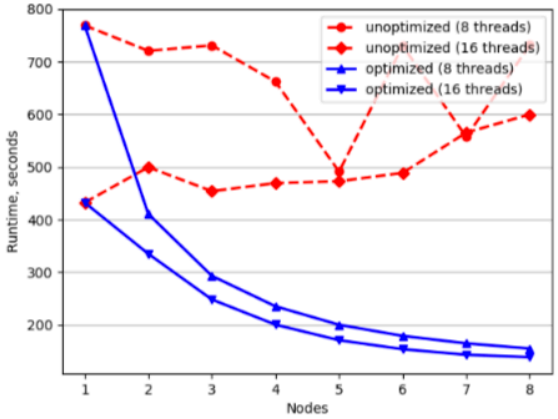


FIGURE 10 kmeans

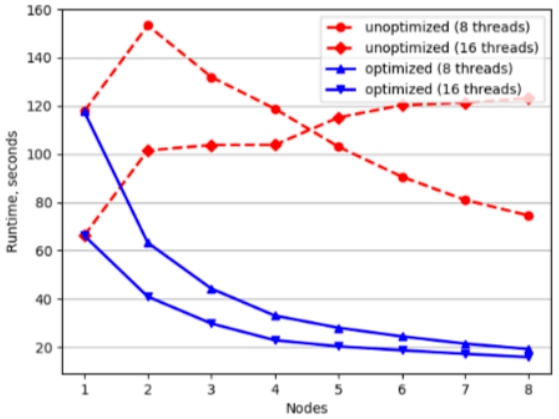


FIGURE 11 lavaMD

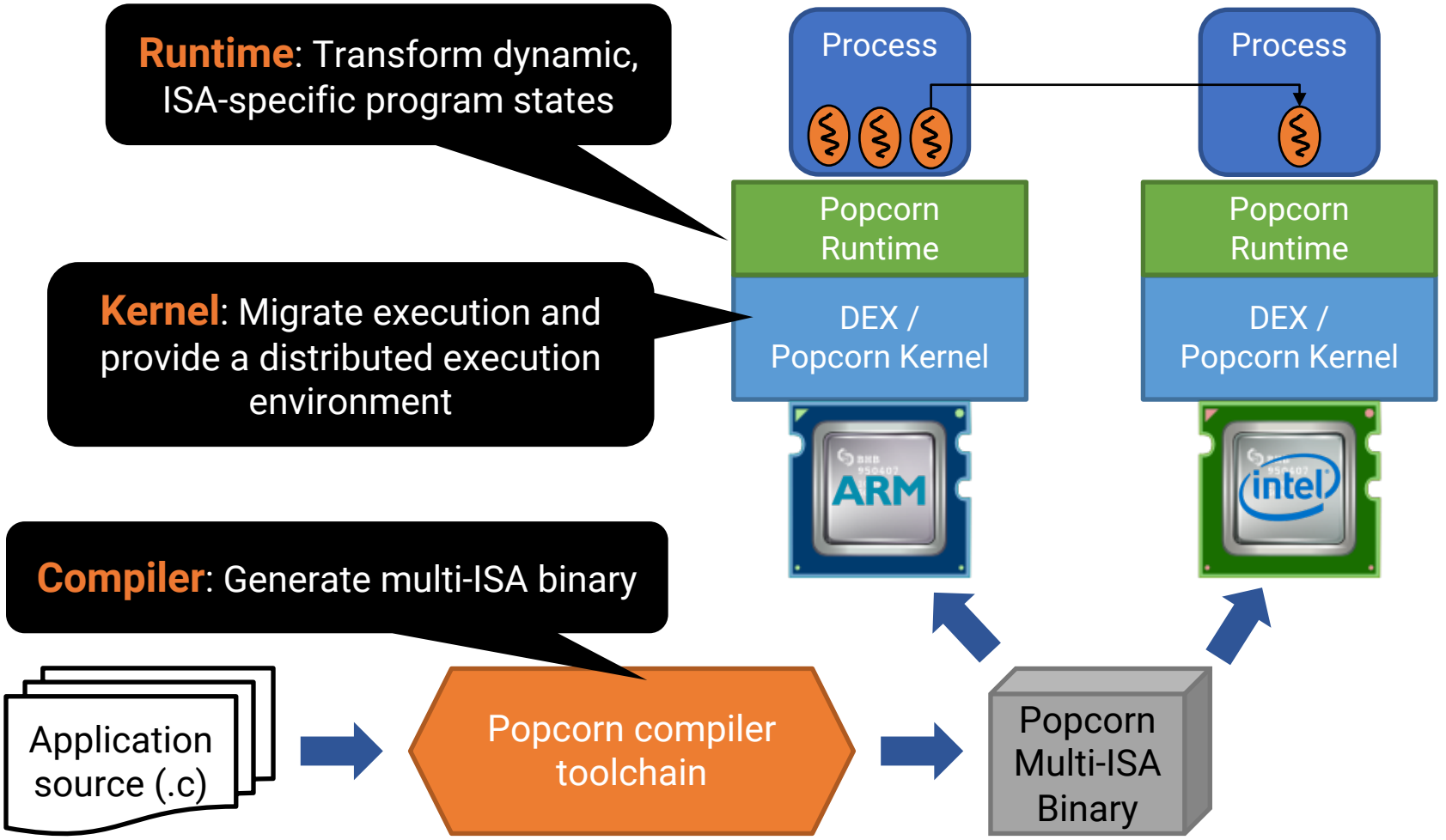
Application Performance

Exploit ISA Affinity

- Key ideas
  - Different ISA, different execution profile (ISCA'14)
  - Exploit the ISA affinity to optimize performance and/or energy

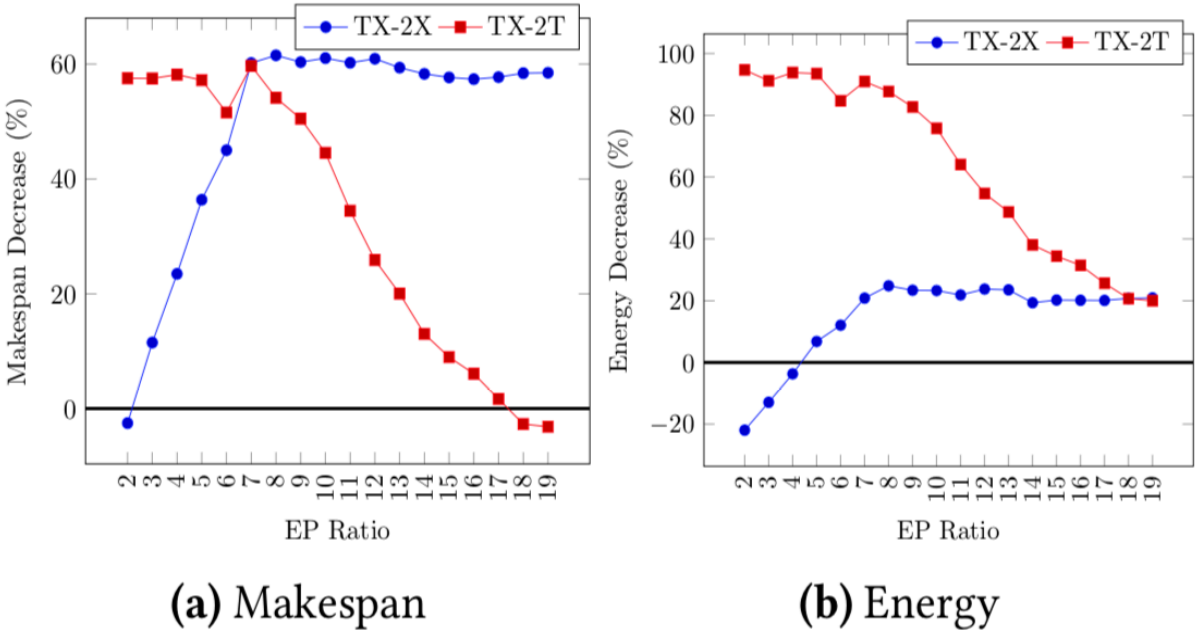
- **Popcorn Linux** for heterogeneous-ISA datacenters (ASPLOS'17, HotOS'17)
  - Mostly compiler and runtime work to support cross-ISA migration
  - Available at https://github.com/ssrg-vt/popcorn-compiler.git

Popcorn Linux Overview



Exploit ISA Affinity

- Scheduling HPC Workloads on Heterogeneous-ISA Architectures (Poster on PPoPP'19)



Leverage Popcorn Linux Further

- Counter attacks exploiting architecture-specific vulnerabilities
  - SFMA co-located with EuroSys'19
- Cross-ISA execution for optimizing SIMD execution
  - SYSTOR'19
- Office of Naval Research, SBIR Solicitation N192-095
  - Multi-Instruction Set Architecture (ISA) Processing with a Peripheral Component Interconnect express (PCIe)
  - Acquisition for PEO IWS 1.0 AEGIS Integrated Combat System

Ongoing Work

- What will be the best for I/O in DEX?
- Make the memory consistency protocol scalable
- Thread placement and scheduling
- Improve RDMA over InfiniBand
  - Keep DMA mapping for frequently migrated pages
  - Leverage high bandwidth as well as low latency
- Provide a programming model that allow (slightly) more modification and optimization
  - Too simple → little chance to optimize
- Incorporate non-volatile memory devices
  - E.g., Intel Optane DC Persistent Memory

Thank you!
고맙습니다

Sang-Hoon Kim

sanghoonkim@ajou.ac.kr

http://sslab.ajou.ac.kr