I/O, today, is Remote (block) Load/Store, and must not be slower than Compute, any more

Manolis Katevenis

FORTH, Heraklion, Crete, Greece (in collab. with Univ. of Crete)
http://www.ics.forth.gr/carv/
Acknowledgements

• Vassilis Papaefstathiou
• Manolis Marazakis
• Nikolaos Chrysos
• Iakovos Mavroidis
• John Goodacre (ARM, KALEAO)
• and many many more!...
The curse of being perceived as “Slow”

• Tell a computer engineer that something is slow and s/he will design-in a myriad other reasons to ensure that it will always... remain slow!
  – “Optimize for the critical path and the common case”

• Input/Output (I/O) hardware _was_ slow
  ⇒ Architecture (virtualization) and Software (OS, Languages, Apps) adjusted assuming that I/O is slow

• I/O hardware is no longer slow – but...
  – Architecture & Software Inertia “ensure” that it remains

• Shared-Memory Parallel Programs work “over Memory”
• Message-Passing Parallel Programs work “over I/O”...
I/O, nowadays, is NVM’s and fast Networks

• Slow peripherals only indirectly connected, to few Proc’s
• Most hi-speed processors are in Datacenters, HPC clusters
• Storage becomes Non-Volatile Memories – like DRAM
  – magnetic disks indirectly, through network & controllers
• High-speed network throughput approaches that of DRAM
• Short-range (e.g. on-board) network latency has no reason to be longer than DRAM latency
“Memory” vs. “I/O” – once upon a time vs. now

• “Memory is fast” ⇒ Virtualized in hardware, at ns scale
  – user process believes it owns entire address space
  – ld/st instructions, caches, TLB ensure access @ ns scale

• “I/O is slow” ⇒ don’t bother to optimize
  – rarely virtualized at user level (=process owns I/O devices)
  – L1-miss L2-hit (on-chip) latency on the order of 10 ns
  – on-chip PL (≈“I/O”’) register read in Zynq UltraScale+ FPGA on the order of 100 ns (~500 ns for PCIe)

Why??

• because people didn’t think it was worth optimizing...
Make I/O (interproc. commun.) a first-class citizen!

- Parallel (& distributed) systems are ubiquitous nowadays
- Interprocessor Communication is as important as Computation
- Shared Memory communication limited to ~100’s threads
- We need communication across hw-coherence islands (“I/O” to nearby chips) to be as fast as DRAM access (which is also “nearby chips”)
- There is *no reason* for a (16-64 By) message send-receive to be any slower than a cache-line invalidate to a nearby socket followed by a cache-line miss from that socket
Inefficiencies in Cluster Communication, to be overcome

- Five (5) copyings of the data, instead of just one (1)!
  - Data copying consumes time and energy
- Start by eliminating (large) NIC buffers:
  - DMA across the network – RDMA
**DMA Initiation: System Call, or directly from User-Level?**

- System Call to initiate a DMA (with physical addr.) \(\approx 3\) to \(4\) \(\mu s\)
  - on 64-bit ARM in Xilinx Zynq UltraScale+ under Linux
- For the virtualized DMA engine (8 channels), that accepts virtual address arguments, Initiation (writing 3-4 registers) \(\approx <0.5\) \(\mu s\)
- Next Questions: Who translates addresses & where? Why copy User to/from Kernel/Library?
for Scalability: not all nodes can be required to know all other nodes’ translation tables –see “Progressive Address Translation” in the M. Katevenis paper in: Stamatis Vassiliadis Symposium 2007
⇒ Destination Addresses in network packets must be Virtual

for Scalability: 64-bit GVA + (global) protection domain identifier
• Address translation architecture of A53 in Xilinx Zynq UltraScale+
• DMA engine is virtualized, but truncates virtual addresses to <64 b
• no mechanism to send the *untranslated* VA to the network
• (remote) load/store instructions suffer compulsory addr. translation
Mbox – Remote Enqueue: Asynchronous Event Multiplexing

- The basic synchronization mechanism for parallel programs, I believe
- Event Multiplexor – requests, notifications (like Select system call)
- Atomic enqueue into shared space, unlike dedicated space with RDMA
- Space reserved only for number of actual senders – not potential senders

longer messages: first composed locally...

P1

store

single-word messages sent via single store instruction

memory 2

m2

... then sent

memory 3

Remote Queue

"Mbox"

Shared Space

P3

Single Receiver

wait on empty

Multiple Senders

P2

m1

Single Receiver

P3

m1m2
Store-multiple instruction for atomic-packet send (remEnq)?

- Need a fast way to send a single, guaranteed-atomic network packet
- Store-multiple (registers) instruction exists on ARMv8, but...
- its current implementation does not guarantee that all data words will be transferred in a single AXI burst to the “I/O” hardware...

M. Katevenis: I/O is no longer slow - Per'18 Workshop, Goteborg May 2018
Per-Task Mbox Q’s & Scheduler as Interrupt Generalization

- Tasks block and wait when synchronously reading from an empty queue
- Scheduler selects a non-empty queue of highest priority and runs its task
- Time quantum switches Q’s and Tasks inside top non-empty priority class
- Enqueues into higher priority Q’s interrupt lower-priority running tasks

RDMA is the Datapath – Queues are the Control
Why copy data between User and Kernel/Library buffers?

- “Pinned” buffers: traditional DMA does not tolerate page faults
- “Registered” buffer addresses, in the lack of “System” (I/O) MMU
- Non-cacheable buffers, in the lack of cache-coherent DMA
- Send buffer reuse immediately after send initiation
- Transfer occurs before receive-buffer ready, and receive Application unwilling to accept reception at library-specified address
MPI: Send-Rcv Match at Receiver, User-Specified Rcv Addr

- cost = one extra network round-trip time for sender to learn the address
Snd-Rcv Match at Sender, User-Specified Receive Address

- minimal latency achieved; early receive (rb pre-alloc by user) still needed
- does not work with MPI_ANY_SOURCE

M. Katevenis: I/O is no longer slow - Per'18 Workshop, Goteborg May 2018
Receiver willing to accept Data at *any* Address (and not copy)

- Eager delivery at preallocated, library-managed buffer, then given to user

Producer (user process): write into sb

"allocate buffer, and... here come the data" or "use rb[i] that had been preallocated for me"

tell me when *and where* my data will be ready

Consumer (user process): read from rb

(done)
Conclusions

• “I/O” is Interprocessor Communication...
• ... and it should be made as fast as Compute
• Short messages: (remote) load/store-multiple instructions
• Long messages: virtualized DMA engine – no system calls
• Global Virtual Address Space (GVAS)
• RDMA is the Datapath – Queues (mbox) are the Control
Back-up Slides
Two Slides from my *Stamatis Vassiliadis 2007 Symposium* talk: 
Network Routing as Generalization of Address Decoding

- Physical Address Decoding in a uniprocessor

- Geographical Address Routing in a multiprocessor

Progressive Address Translation: Localize Migration Updates

- Packets carry global virtual addresses
- Tables provide physical route (address) for the next few steps
- When page 9 migrates within D, only tables in that domain need updating
- Variable-size-page translation tables look like internet routing tables (longest-prefix matches if we want small-page-within-big-region migration)
- Tables that partition the system, for protection against untrusted operating systems, look like internet firewalls