

# A New DMA Registration Strategy for Pinning-Based High Performance Networks

Dan Bonachea & Christian Bell

U.C. Berkeley and LBNL

*{bonachea,csbell}@cs.berkeley.edu*

*<http://www.cs.berkeley.edu/~bonachea/gasnet>*

*This work is part of the UPC and Titanium projects,  
funded in part by the DOE, NSF and DOD*

## Problem Motivation: Client

- Global-address space (GAS) languages
  - UPC, Titanium, Co-Array Fortran, etc.
  - Large globally-shared memory areas w/language support for direct access to remote memory
    - Total remotely accessible memory size limited only by VM space
    - Working set of memory being touched likely to fit in physical mem
  - App performance tends to be sensitive to the latency & CPU overhead for small operations
- Implications for communication layer (GASNet)
  - Want low-latency and low-overhead for non-blocking small puts/gets (think ? 8 bytes)
  - Want high-bandwidth, zero-copy msgs for large transfers
    - zero-copy: get higher bandwidth AND avoid CPU overheads
  - Ideally all communication should be fully one-sided
    - one-sided: don't interrupt remote host CPU - hurts remote compute performance and increases round-trip latency

## Problem Motivation: Hardware

- Pinning-based NIC's (e.g. Myrinet, Infiniband)
  - Provide one-sided RDMA transfer support, but...
  - Memory must be explicitly registered ahead of time
    - Requires explicit action by the host CPU on **both** sides
    - Tell the OS to pin virtual memory page (kernel call)
    - Register fixed virtual/physical mapping w/NIC (PCI transaction)
  - Memory registration can be expensive!
    - Especially on Myrinet - average is 40 microsec to register one page, 6 **milliseconds** to deregister one page
    - Costs primarily due to preventing race conditions with pending messages that could compromise system memory protection
  - Want to reduce the frequency of registration operations and the need for two-sided synchronization
    - Reducing cost of a single registration operation is also important, but orthogonal to this research

## Memory Registration Approaches

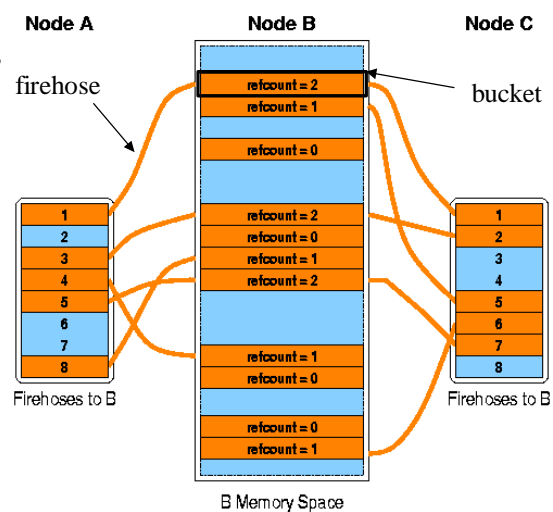
- Hardware-Based (e.g. Quadrics)
  - Zero-copy, One-sided, Full memory space accessible, No handshaking or bookkeeping in software
  - Hardware complexity and price, Kernel modifications
- Pin Everything - pin pages at startup or when allocated
  - Zero-copy, One-sided (no handshaking)
  - Total usage limited physical memory, may require a custom allocator
- Bounce Buffers - stream data through pre-pinned bufs
  - No registration cost at runtime, Full memory space accessible
  - Two-sided, mem copy costs (CPU consumption - increases CPU overhead, prevents comm. overlap), Messaging overhead (metadata and handshaking)
- Rendezvous - round-trip message to pin remote pages
  - Zero-copy, Full memory space accessible, Only handshaking synchronous
  - Two-sided, Registration costs paid on every operation (very bad on Myrinet)
- Firehose - our algorithm
  - Zero-copy, One-sided (common case), Full memory space accessible, Only handshaking is synchronous, Registration costs amortized
  - Messaging overhead (metadata and handshaking) on miss (uncommon case)

## Basic Idea: A Hybrid Approach

- Firehose - A distributed strategy for handling registration
  - Get the benefits of Pin-Everything in the common case
  - Revert to Rendezvous-like behavior for the uncommon case
- Allow remote nodes to control and cache registration ops
  - Each node sets aside M bytes of physical memory for registration purposes (some reasonable fraction of phys mem)
  - Guarantee  $F = \lfloor \frac{M}{\text{pagesize} * (\text{nodes} - 1)} \rfloor$  physical pages to every remote node, which has control over where they're mapped in virtual mem
  - When a remote page is already mapped, can freely use one-sided RDMA on it (a hit) - exploits temporal and physical locality
  - Send Rendezvous-like synchronization messages to change mappings when a needed remote page not mapped (a miss)
  - Also cache local memory registration to amortize pinning costs

## Firehose: Conceptual Diagram

- Runtime snapshot of two nodes (A and C) mapping their firehoses to a third node (B)
- A and C can freely "pour" data through their firehoses using RDMA to/from anywhere in the buckets they map on B
- Refcounts used to track number of attached firehoses (or local pins)
- Support lazy deregistration for buckets w/ refcount = 0 using a victim FIFO with a fixed max length ( $\text{MAXVICTIM}$ ) to avoid re-pinning costs



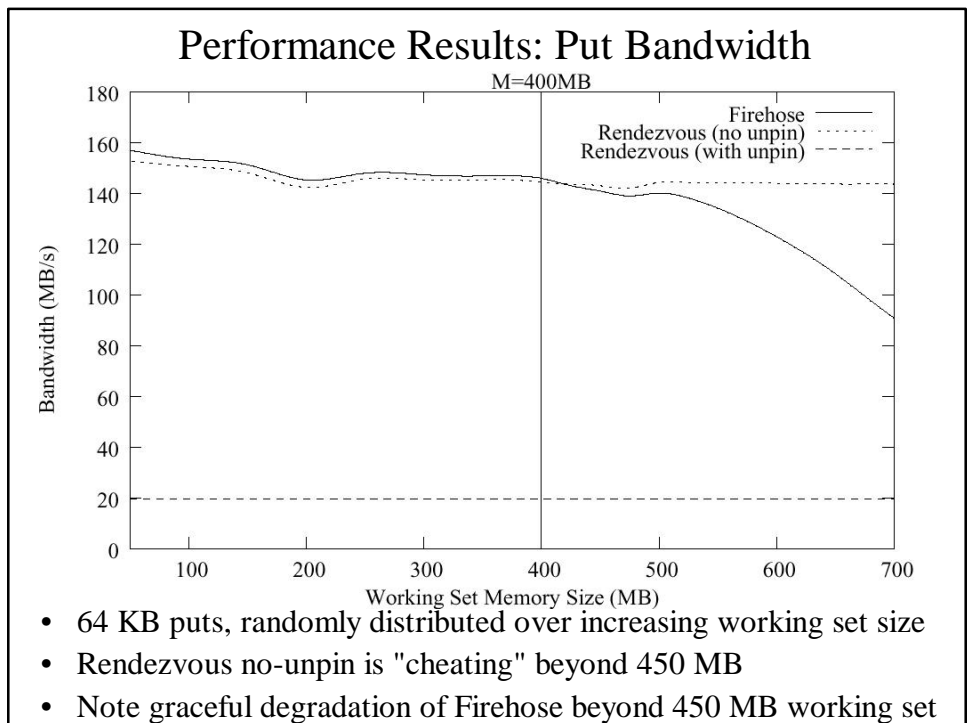
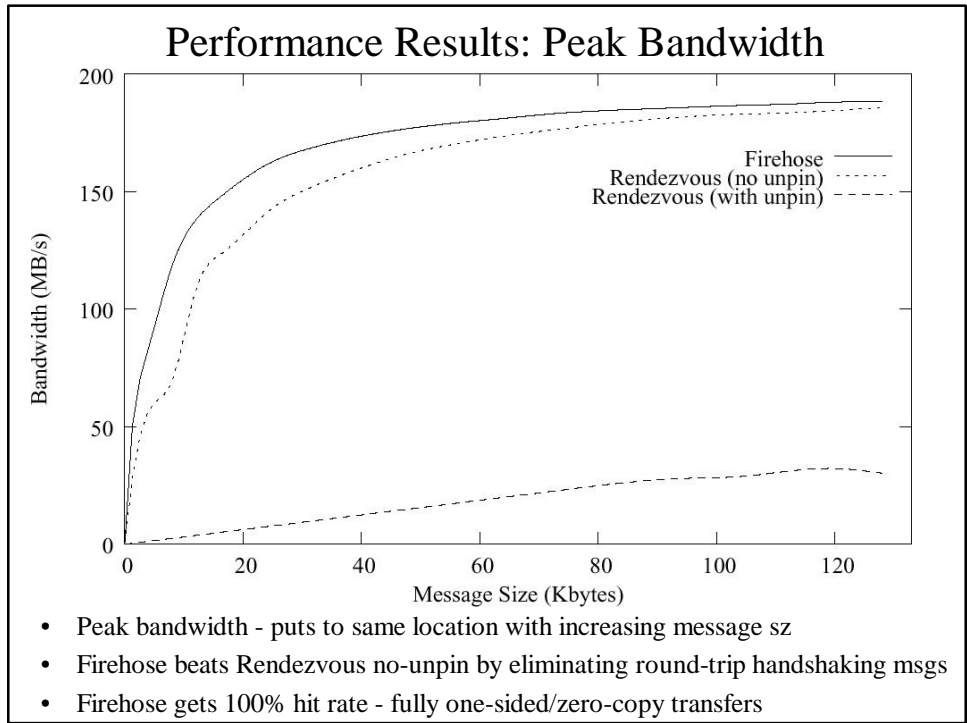
## Firehose: Implementation Details

- Implemented on Myrinet/GM as part of a GASNet impl.
  - Fully non-blocking (even for firehose misses) and supports multi-threaded clients - also need refcounts on firehoses to prevent races
  - Use active messages to perform remote Firehose operations
  - Currently only one-sided for puts, because GM lacks RDMA gets
    - For now, gets implemented as an active message and a firehose put
  - Physical memory consumption never exceeds  $M + \text{MAXVICTIM}$  (both tunable parameters) - may be much less, based on access pattern
- Data structures used:
  - Local bucket table: bucket virtual addr => bucket ref count
  - Bucket Victim FIFO: links buckets w/ refcount = 0
  - Firehose table:
    - (remote node id, bucket virtual addr) => firehose ref count
- All bookkeeping operations are  $O(1)$ 
  - Overhead for all metadata lookups/modifications for a put < 1?s

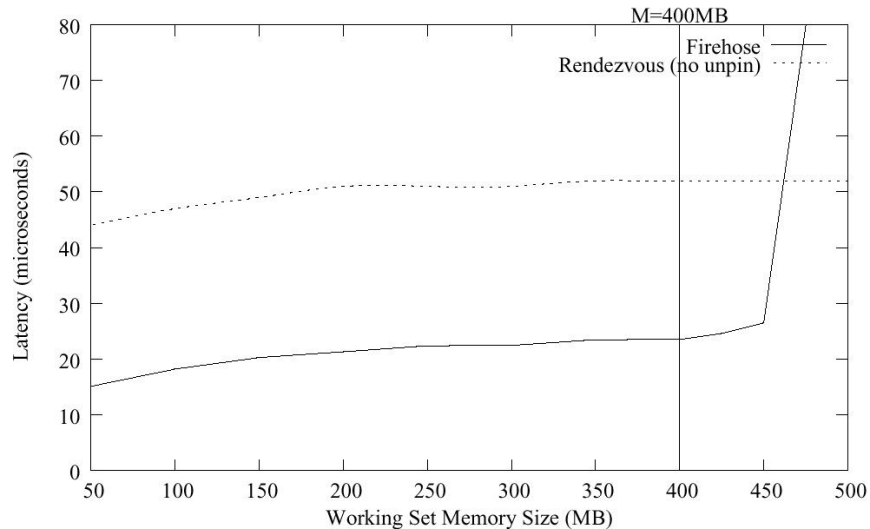
## Application Benchmarks

App Name	Total Puts	Registration Strategy	Total Runtime	Average Put Latency
Cannon Matrix Multiply	1.5 M	Rendezvous with-unpin	5460 s	5141 ?s
		Rendezvous no-unpin	797 s	34 ?s
		Firehose (hit: 99.8%) (miss: 0.2%)	781 s	14 ?s 46 ?s
Bitonic Sort	2.1 M	Rendezvous with-unpin	4740 s	522 ?s
		Rendezvous no-unpin	289 s	33 ?s
		Firehose (hit: 99.98%) (miss: 0.02%)	255 s	15 ?s 54 ?s

- Simple kernels written in Titanium - just want a realistic access pattern
  - 2 nodes, Dual PIII-866MHz, 1GB RAM, Myrinet PCI64C, 33MHz/64bit PCI bus
- Rendezvous includes caching of locally-pinned pages
  - Rendezvous no-unpin not a robust strategy for GAS lang, shown for comparison
- Firehose misses are rare, and even misses often hit in victim cache
  - Avg time to move firehose on remote node is 5 ?s and 14 ?s (pin takes 40 ?s)
  - Firehose never needed to unpin anything in this case (total mem sz < phys mem)



## Performance Results: Roundtrip Put Latency



- 8-byte puts, randomly distributed over increasing working set size
- Rendezvous no-unpin is "cheating" beyond 450 MB
- Rendezvous with-unpin not shown: is about 6000 microseconds

## Conclusions

- Firehose algorithm is a good registration strategy for GAS languages on pinning-based networks
  - Get the performance benefits of Pin-Everything approach (without the drawbacks) in the common case and degrade to Rendezvous-like behavior for the uncommon case
  - Exposes one-sided, zero-copy RDMA as common case
  - Amortizes cost of registration and synchronization over multiple operations, and avoids cost of repinning recently used pages
  - Cost of handshaking and registration negligible when working set fits in physical memory, and degrades gracefully beyond
- Future Work
  - Use Firehose for an Infiniband-GASNet implementation
  - Make MAXVICTIM adaptive for better scaling when access pattern unbalanced (bucket "stealing") avoid unpin-repin costs
  - Extend to RDMA gets in GM2 (soon to be released)

<http://www.cs.berkeley.edu/~bonachea/gasnet>