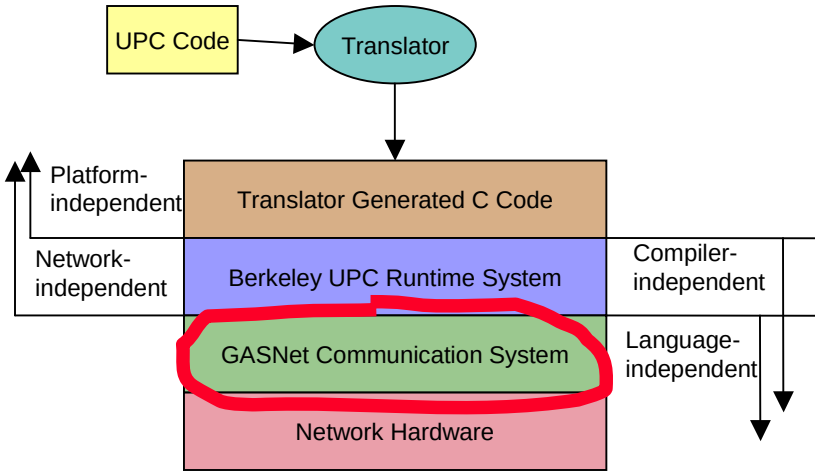
GASNet: A High-Performance, Portable Communication System for Partitioned Global Address Space Languages

Dan Bonachea

Kathy Yelick, Christian Bell,
Wei Chen, Jason Duell,
Paul Hargrove, Parry Husbands,
Costin Iancu, Mike Welcome



http://upc.lbl.gov

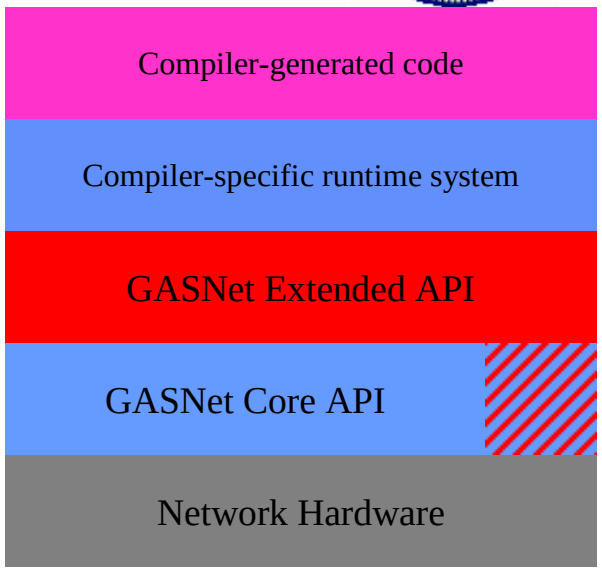
GASNet: High-Level Outline

- GASNet Design Overview and Implementation Status
- Firehose: A DMA Registration Strategy for Pinning-Based Networks
  - The "right way" to handle memory sharing for PGAS languages on difficult but common NIC hardware
- GASNet vs ARMCI comparison
  - what Firehose buys us
- GASNet Extensions for Non-Contiguous Remote Access (Vector, Indexed and Strided)

GASNet Design Overview - Goals

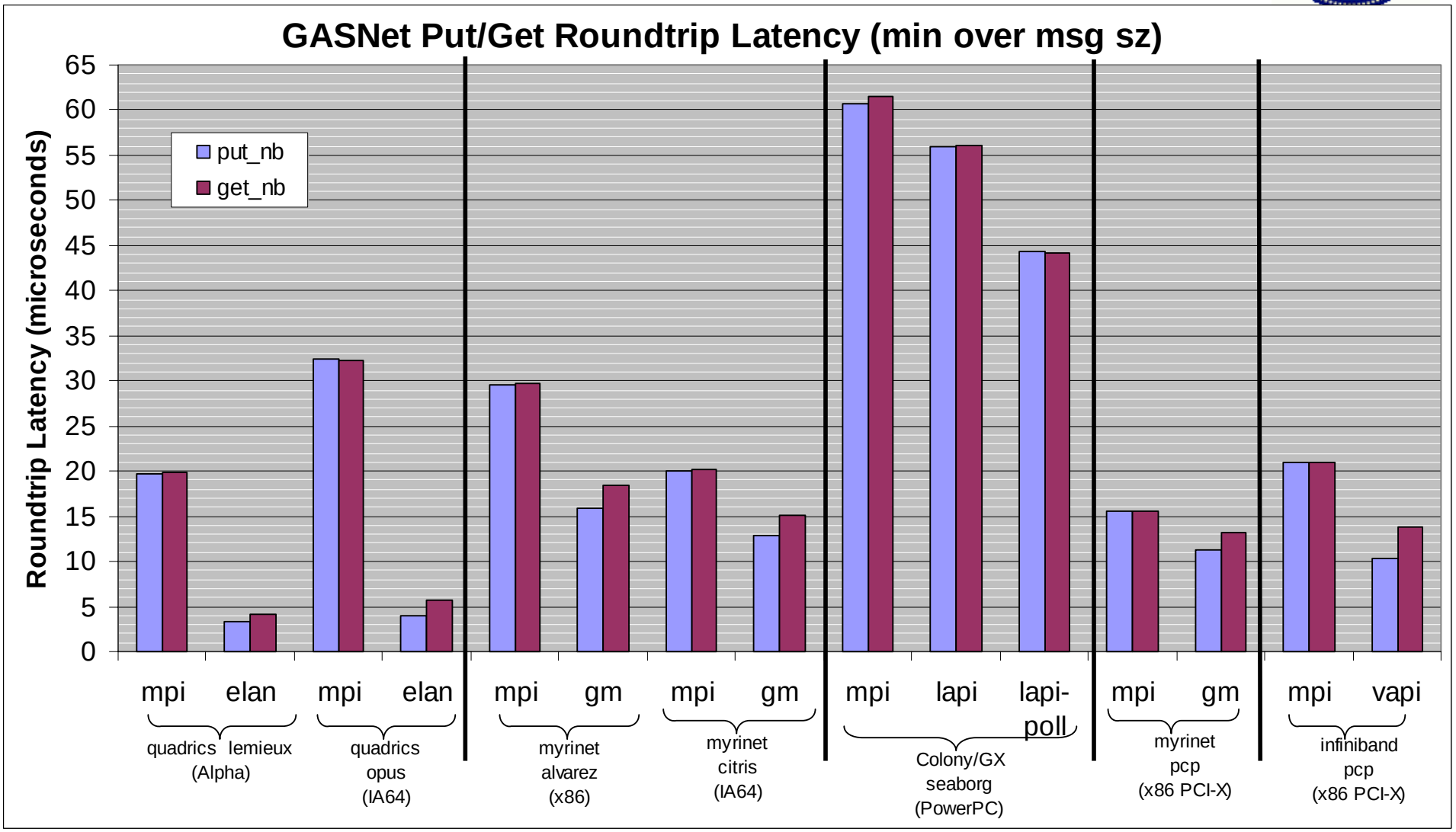
- Language-independence: support multiple PGAS languages/compilers
  - UPC, Titanium, Co-array Fortran, possibly others..
  - Hide UPC- or compiler-specific details such as pointer-to-shared representation
- Hardware-independence: variety of parallel arch., OS's & networks
  - SMP's, clusters of uniprocessors or SMPs
  - Current networks:
    - Native network conduits: Myrinet GM, Quadrics Elan, Infiniband VAPI, IBM LAPI
    - Portable network conduits: MPI 1.1, Ethernet UDP
    - Under development: Cray X-1, SGI/Cray Shmem, Dolphin SCI
  - Current platforms:
    - CPU: x86, Itanium, Opteron, Alpha, Power3/4, SPARC, PA-RISC, MIPS
    - OS: Linux, Solaris, AIX, Tru64, Unicos, FreeBSD, IRIX, HPUX, Cygwin, MacOS
- Ease of implementation on new hardware
  - Allow quick implementations
  - Allow implementations to leverage performance characteristics of hardware
  - Allow flexibility in message servicing paradigm (polling, interrupts, hybrids, etc)
- Want both portability & performance

GASNet Design Overview - System Architecture

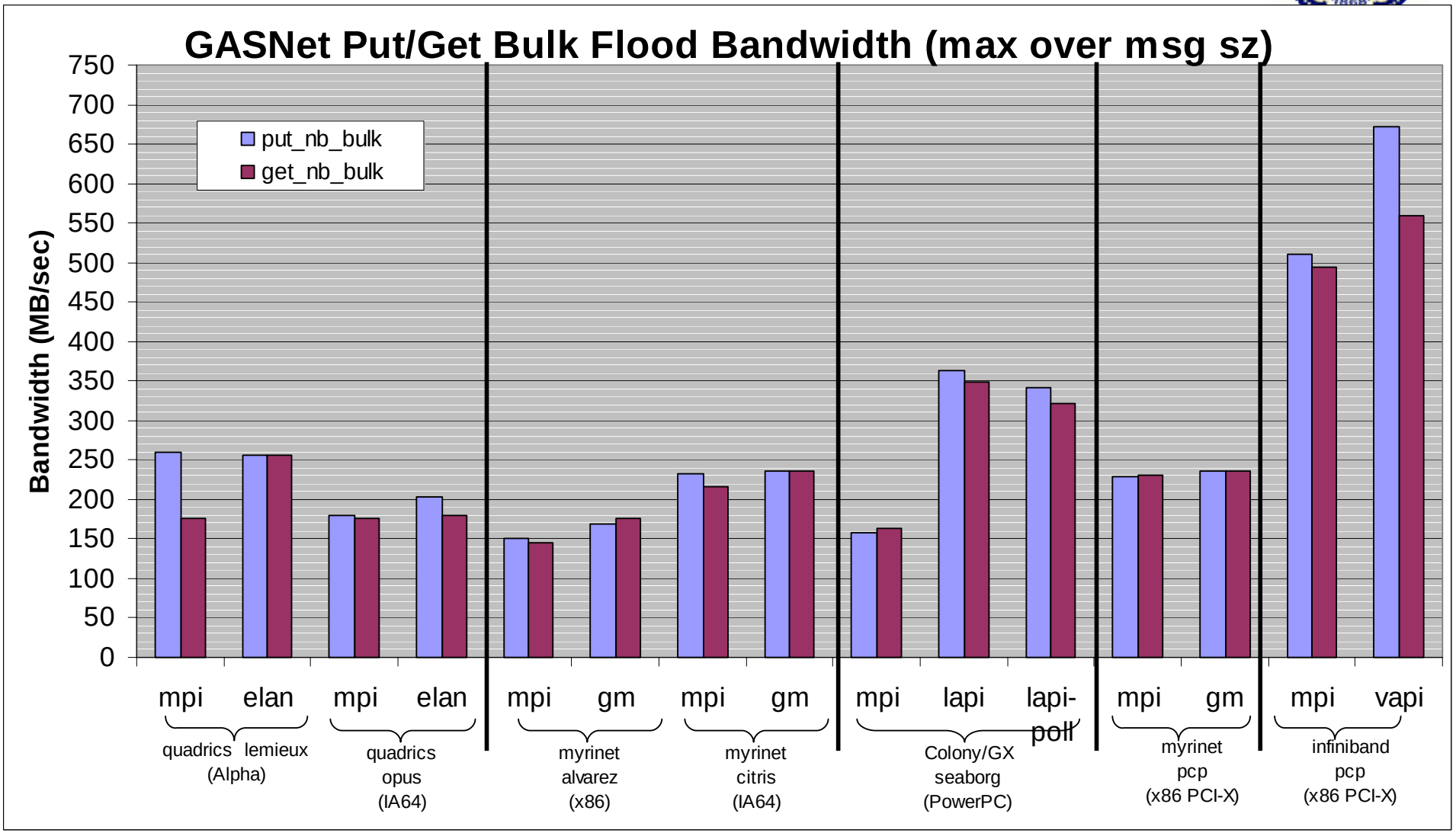


- 2-Level architecture to ease implementation:
- Core API
  - Most basic required primitives, as narrow and general as possible
  - Implemented directly on each network
  - Based heavily on active messages paradigm
- Extended API
  - Wider interface that includes more complicated operations
  - We provide a reference implementation of the extended API in terms of the core API
  - Implementors can choose to directly implement any subset for performance - leverage hardware support for higher-level operations
  - Currently includes:
    - blocking and non-blocking puts/gets (all contiguous), flexible synchronization mechanisms, barriers
    - Just recently added non-contiguous extensions (coming up later)

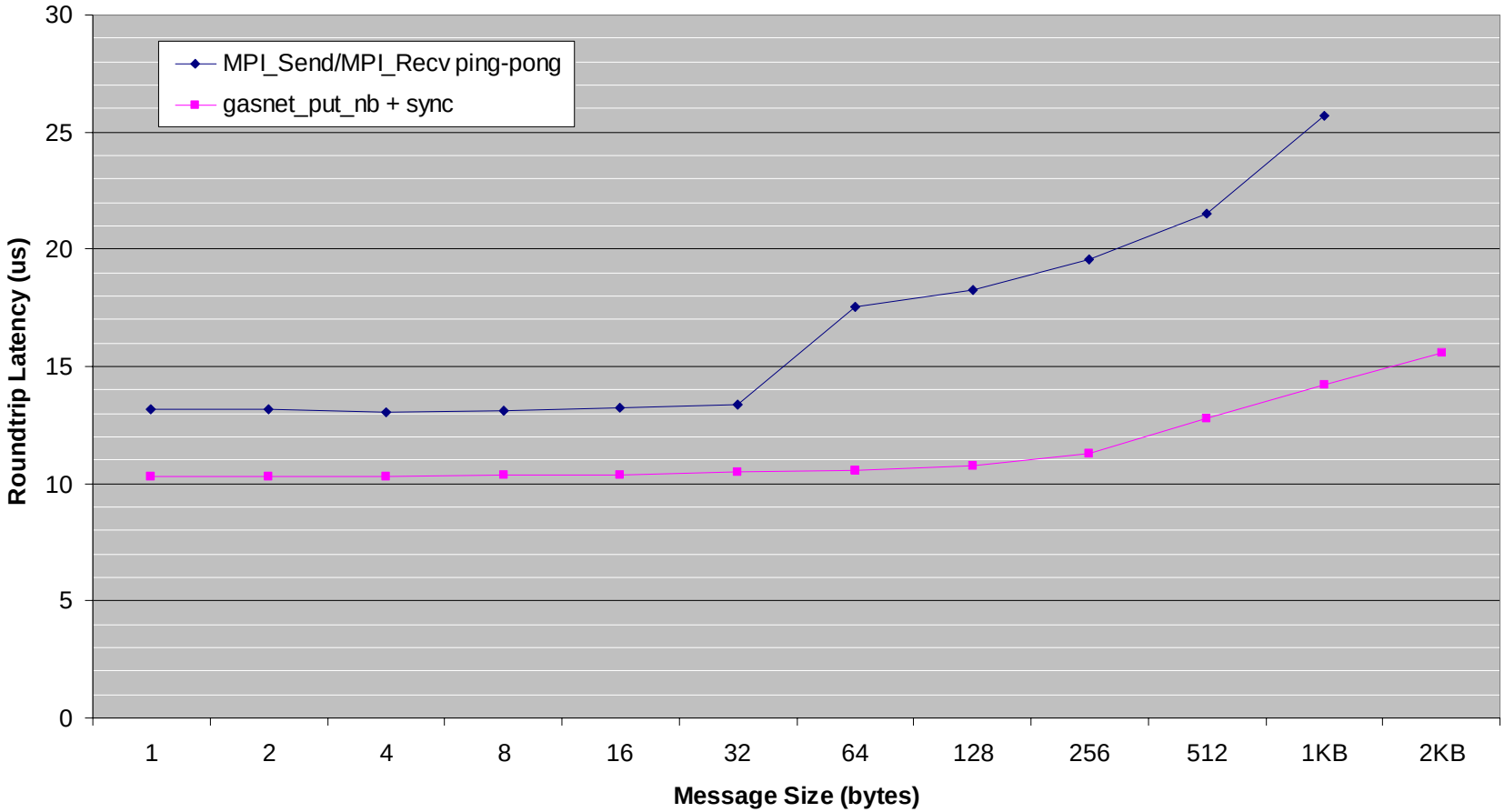
GASNet Performance Summary



GASNet Performance Summary



GASNet vs. MPI on Infiniband



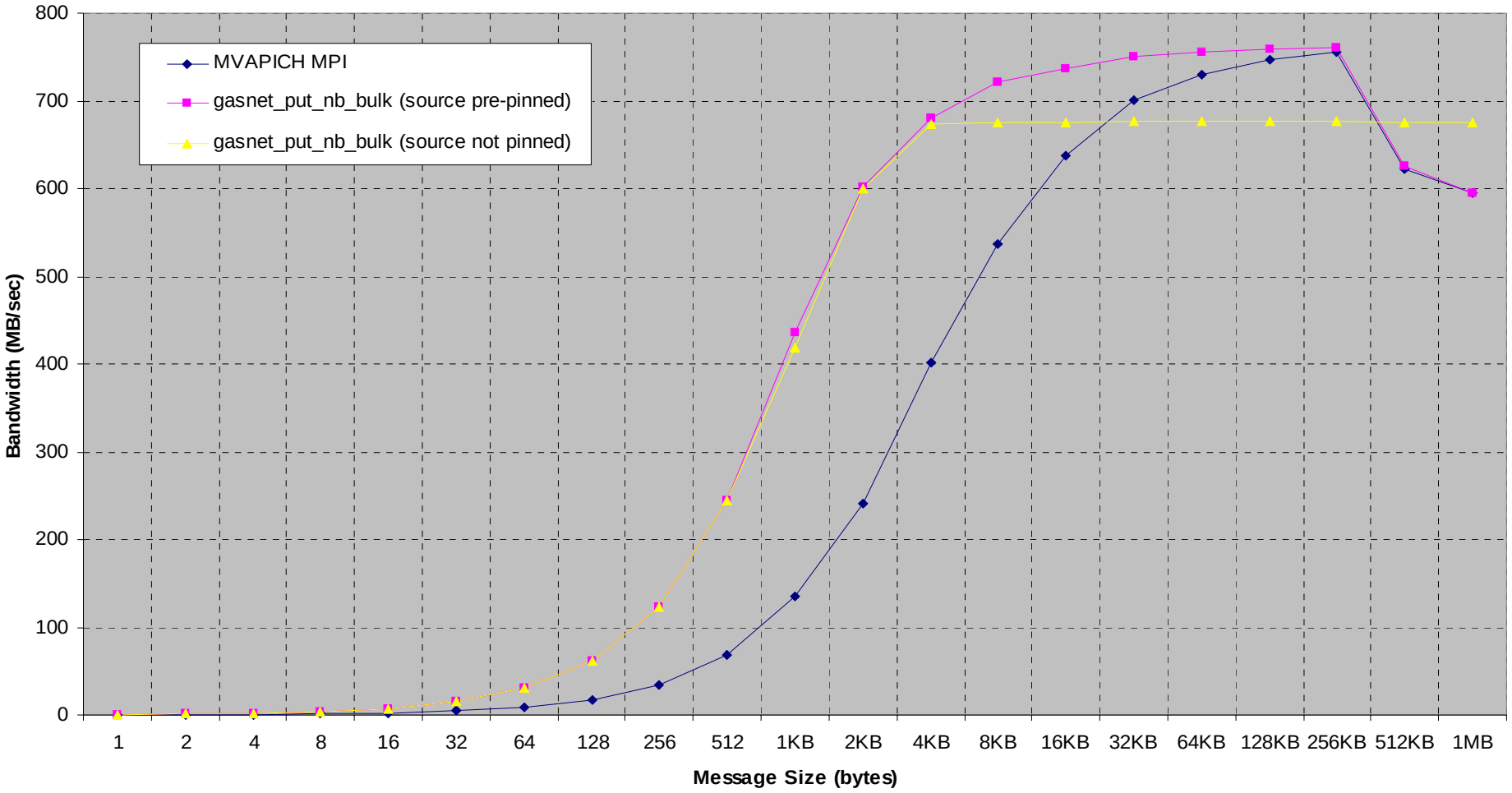
OSU MVAPICH widely regarded as the "best" MPI implementation on Infiniband

MVAPICH code based on the FTG project MVICH (MPI over VIA)

GASNet wins because fully one-sided, no tag matching or two-sided sync.overheads

MPI semantics provide two-sided synchronization, whether you want it or not

GASNet vs. MPI on Infiniband



GASNet significantly outperforms MPI at mid-range sizes - the cost of MPI tag matching

Yellow line shows the cost of naïve bounce-buffer pipelining when local side not prepinned - memory registration is an important issue

Firehose: A Distributed DMA Registration Strategy for Pinning-Based High Performance Networks

Dan Bonachea (Design)
Christian Bell (GM)
Paul Hargrove (Infiniband)
Rajesh Nishtala (GM-SMP)

Problem Motivation: Client

- Partitioned Global-address space (PGAS) languages
  - 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, Dolphin)
  - 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 VERY expensive!
    - Especially on Myrinet - average is 40 microsec to register one page, **6000** microseconds to deregister one page (cf. 12us round-trip RDMA time)
    - 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

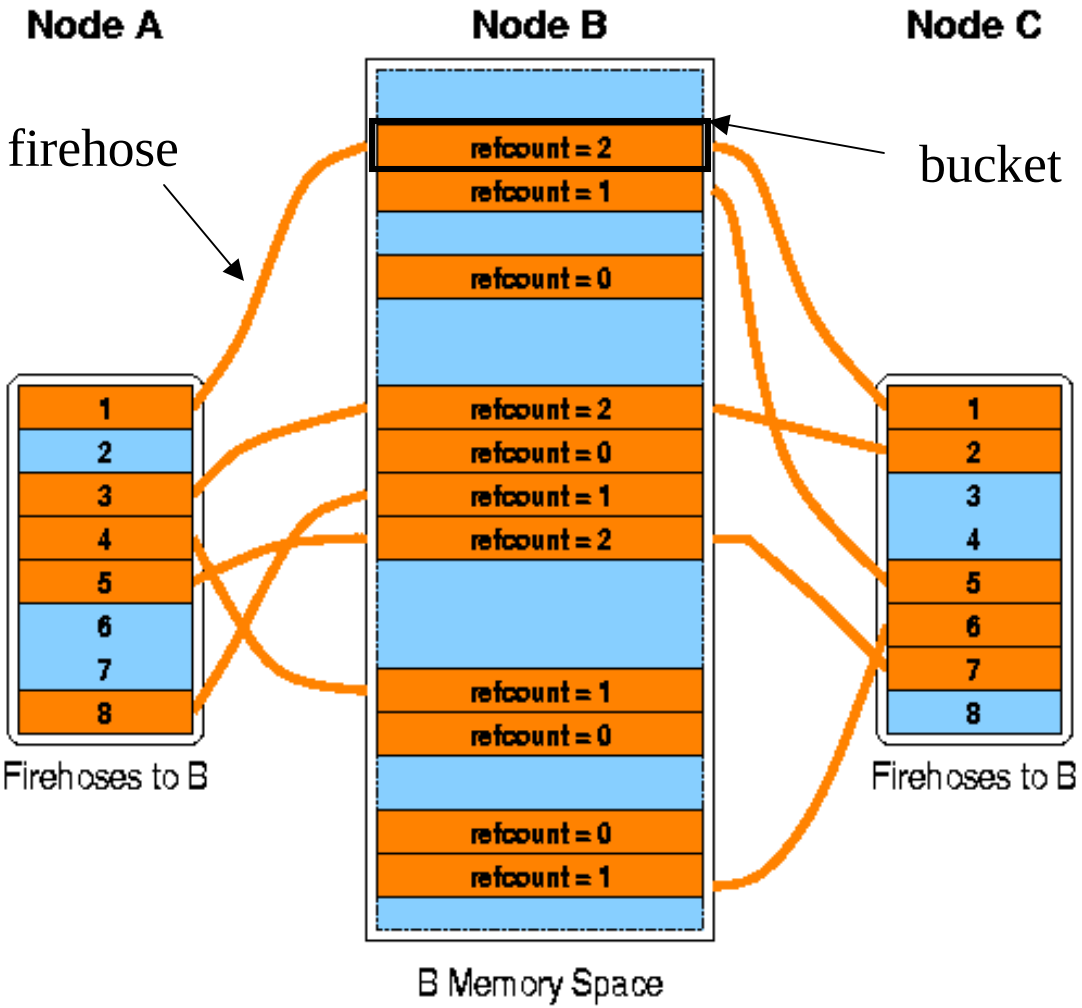
Approach	Zero-copy	One-sided	Full VM avail	Notes
Hardware-based (eg.Quadrics)	✓	✓	✓	Hardware manages everything No handshaking or bookkeeping in software Hardware complexity and price, Kernel modifications
Pin Everything	✓	✓	✗	Pin all pages at startup or when allocated (collectively) Total usage limited to physical memory, may require a custom allocator
Bounce Buffers	✗	✗	✓	Stream data through pre-pinned bufs on one/both sides Mem copy costs (CPU consumption/overhead, prevents comm. overlap), Messaging overhead (metadata and handshaking)
Rendezvous	✓	✗	✓	Round-trip message to pin remote pages before each op Registration costs paid on every operation
Firehose	✓ (common case)	✓ (common case)	✓	Common case: All the benefits of hardware-based Uncommon case: Messaging overhead (metadata and handshaking)

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 to avoid re-pinning costs

- For details, see Firehose paper on UPC publications page (CAC'03)

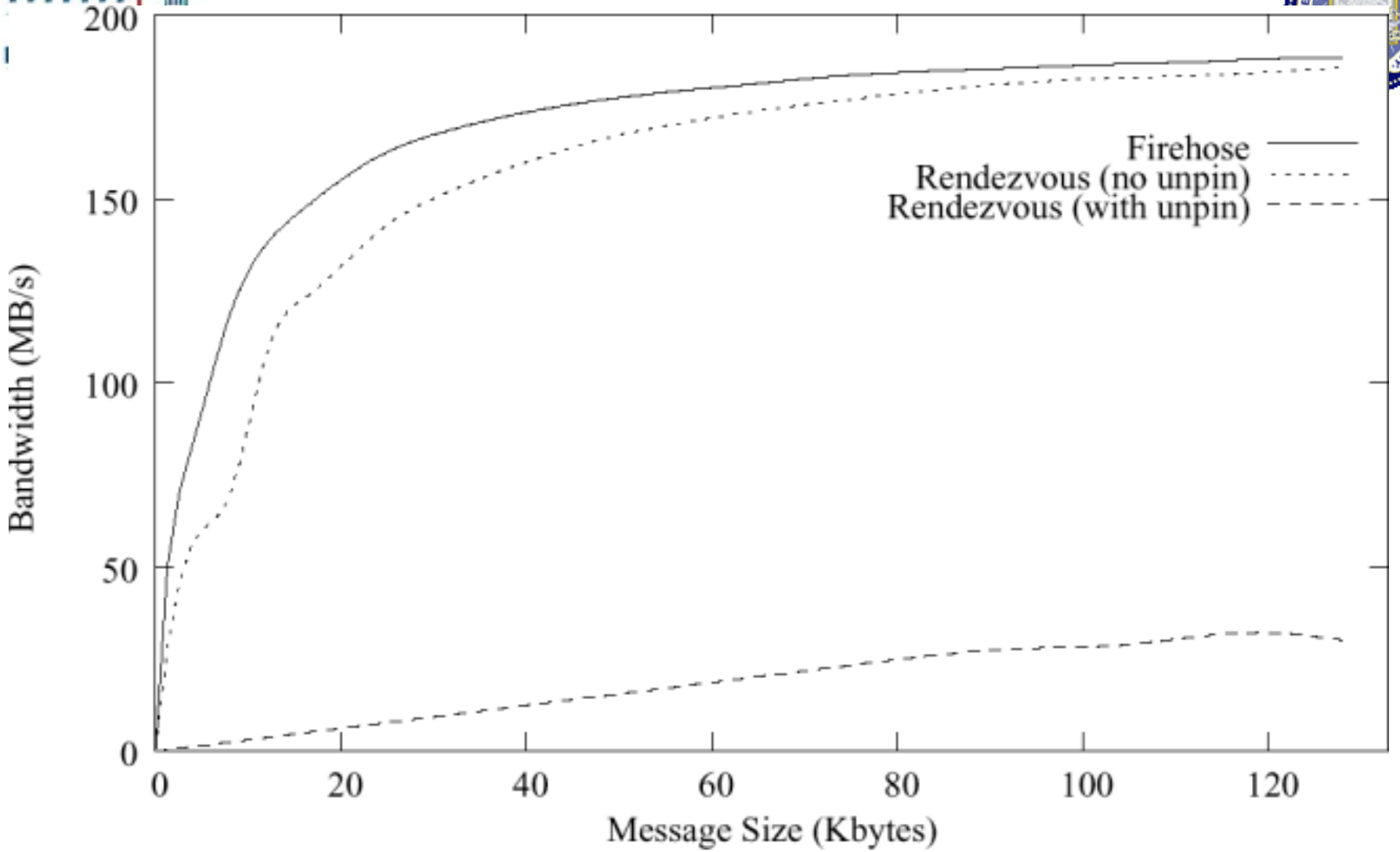


Application Benchmarks

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

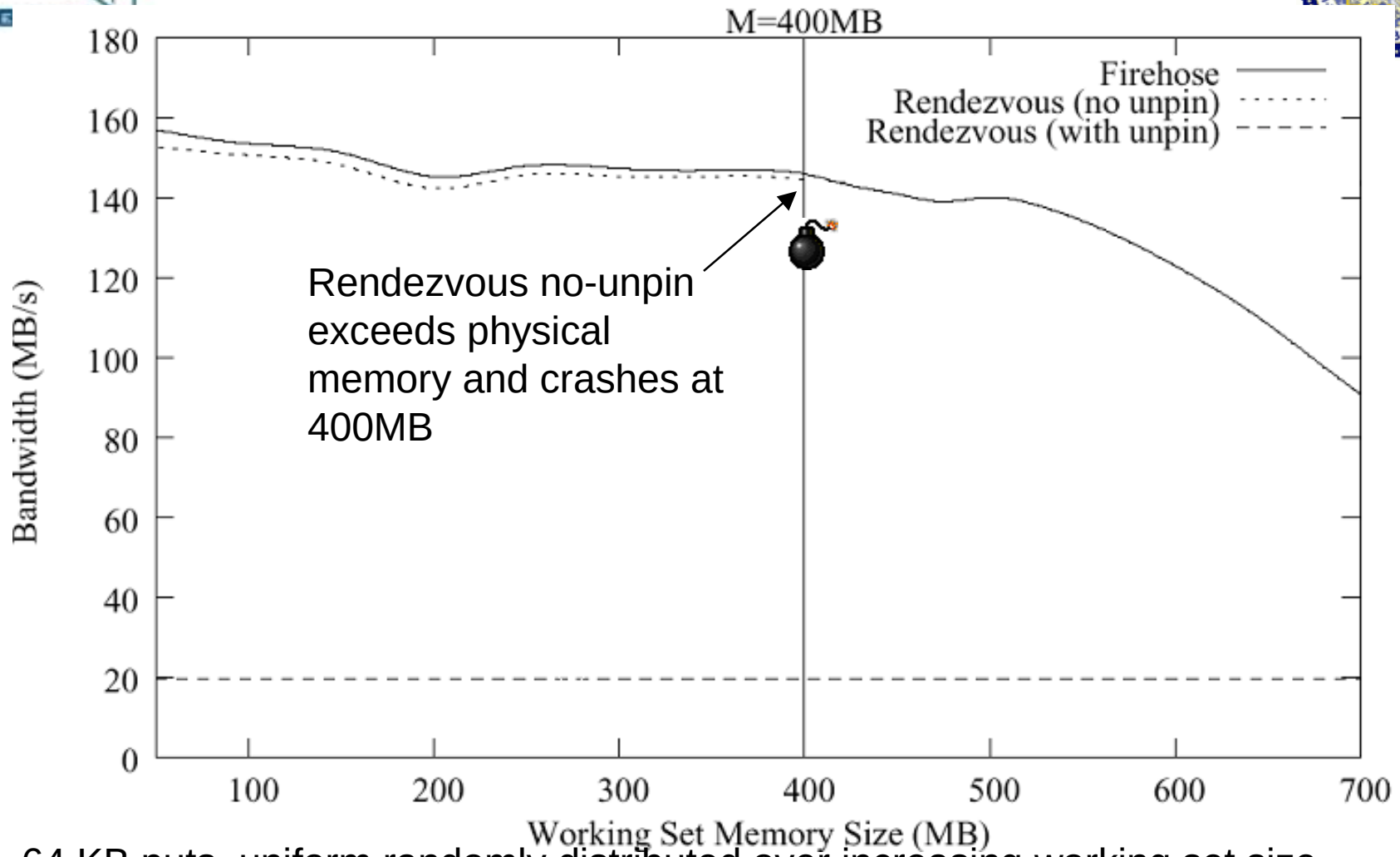
- Simple kernels written in Titanium - just want a realistic access pattern
  - 2 nodes, Dual PIII-866MHz, 1GB RAM, Myrinet PCI64C, 33MHz/64bit PCI bus
- Firehose misses are rare, and even misses often hit in victim cache
  - Firehose never needed to unpin anything in this case (total mem sz < phys mem)

Performance Results: "Best-case" Bandwidth



- Peak bandwidth - puts to same location with increasing message sz
- Firehose beats Rendezvous no-unpin by eliminating round-trip handshaking msgs
- Firehose gets 100% hit rate - fully one-sided/zero-copy transfers

Performance Results: "Worst-case" Put Bandwidth



- 64 KB puts, uniform randomly distributed over increasing working set size
  - worst-case temporal and spatial locality
- Note graceful degradation of Firehose beyond 400 MB working set

Current/Future Firehose Work

- Recent work on firehose
  - Generalized Firehose for Infiniband/VAPI-GASNet (region-based), prepared for use in Dolphin/GASNet
  - Algorithmic improvements for better scaling when access pattern unbalanced (bucket "stealing") avoid unpin-repin cost
- Current/Future work on Firehose:
  - Improving pthread-safe implementation of Firehose
    - implementing optimistic concurrency control between client threads, to maximize firehose resource utilization
    - Fixing a few tricky race conditions

Firehose Conclusions

- Firehose algorithm is an ideal registration strategy for PGAS languages on pinning-based networks
  - Performance of Pin-Everything (without the drawbacks) in the common case, degrades to Rendezvous-like behavior for the uncommon case
  - Exposes one-sided, zero-copy RDMA as common case
  - Amortizes cost of registration/synch over many ops, uses temporal/spatial locality to avoid cost of repinning
  - Cost of handshaking and registration negligible when working set fits in physical memory, degrades gracefully beyond

GASNet/ARMCI Comparison

- ARMCI - Aggregate Remote Memory Copy Interface (PNNL) v1.1Beta (most recent avail as of May 5th, 2004)
  - Used by Global Arrays, Rice CAF and GPSHMEM
  - Portable platforms: SMP, MPI, PVM, TCP Sockets, SHMEM
  - Native platforms: LAPI, GM, Elan, VAPI, Hitachi, Fujitsu
- GASNet
  - Currently used by Titanium, Berkeley UPC, GCC/UPC
  - Future clients: Rice CAF, MPICH
  - Portable platforms: SMP, MPI, Ethernet UDP, SHMEM
  - Native platforms: LAPI, GM, Elan, VAPI, Cray X-1, Altix
  - In-progress: Dolphin

GASNet/ARMCI Interface Issues

- **GASNet:**
  - Provides AM for extensibility
    - support pack/unpack, dist. memory alloc, dist monitors, dist GC, accumulate, firehose
  - Full job bootstrapping support
    - Tailored specifically for the needs of parallel PGAS languages - hides the ugly details
  - Very flexible/expressive non-blocking put/get and sync modes
    - allows polling-based, interrupt-based or hybrid message handling
- **ARMCI:**
  - no bootstrapping support (must use MPI)
  - all remotely accessible memory must be collectively allocated and explicitly pinned - but not guaranteed to be contiguous or aligned across nodes
  - not pthread-safe, requires process-based client model
  - Data server process often involved, using SysV shared memory
    - Many of their communication paths are two-sided and two-copy
    - context switch overheads, CPU cache pollution & cycle stealing (interrupt-based)
    - Needs SysV kernel mods on some OS's, RDMA+SysV often buggy
  - Accumulate and locking support, but no general client extensibility

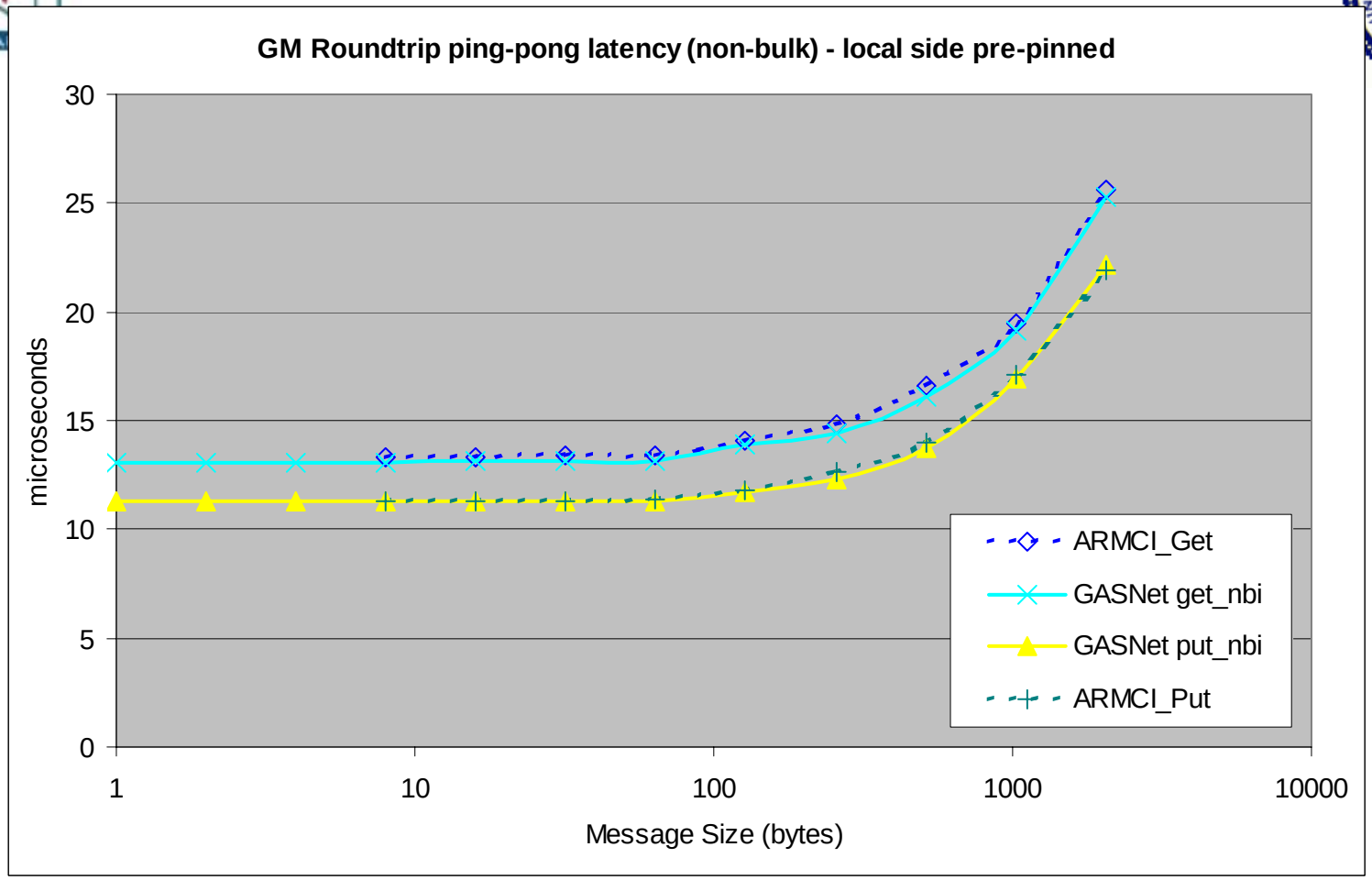
GASNet vs ARMCI performance

- Test Machine:
  - 4-node, Dual Intel P4 Xeon 2.8GHz, 512KB L1 cache, 4GB main memory
  - Red Hat Linux 8.0, 2.4.18-14smp kernel, glibc 2.2.93-5
  - GM: LANai 10 PCI-X D NIC, GM 2.0.6
  - VAPI: Mellanox Cougar (InfiniHost-A1) PCI-X NIC, firmware 3.0, software version 3.0.1, DivergeNet 8-port InfiniBand 4X switch
  - gcc 3.2.2, default compile options for each system
- Contiguous Put/Get Tests:
  - ping-pong latency test and flood bandwidth tests
  - vary whether either side was explicitly pre-pinned

ARMCI Caveats

- According to ARMCI docs:
  - Their Infiniband port is still "initial"
  - Their Myrinet port is "not fully optimized yet"
- Numbers are from modified version of their tester
  - small changes to get an apples-to-apples comparison
  - message sizes, MB=2^20, Put synchronization
- Some of the results differ from their published results
- Only testing contiguous put/get - ARMCI excels at non-contiguous, which GASNet has just added
- Unable to get their newly added non-blocking support to work
  - Have contacted them, working to resolve issues

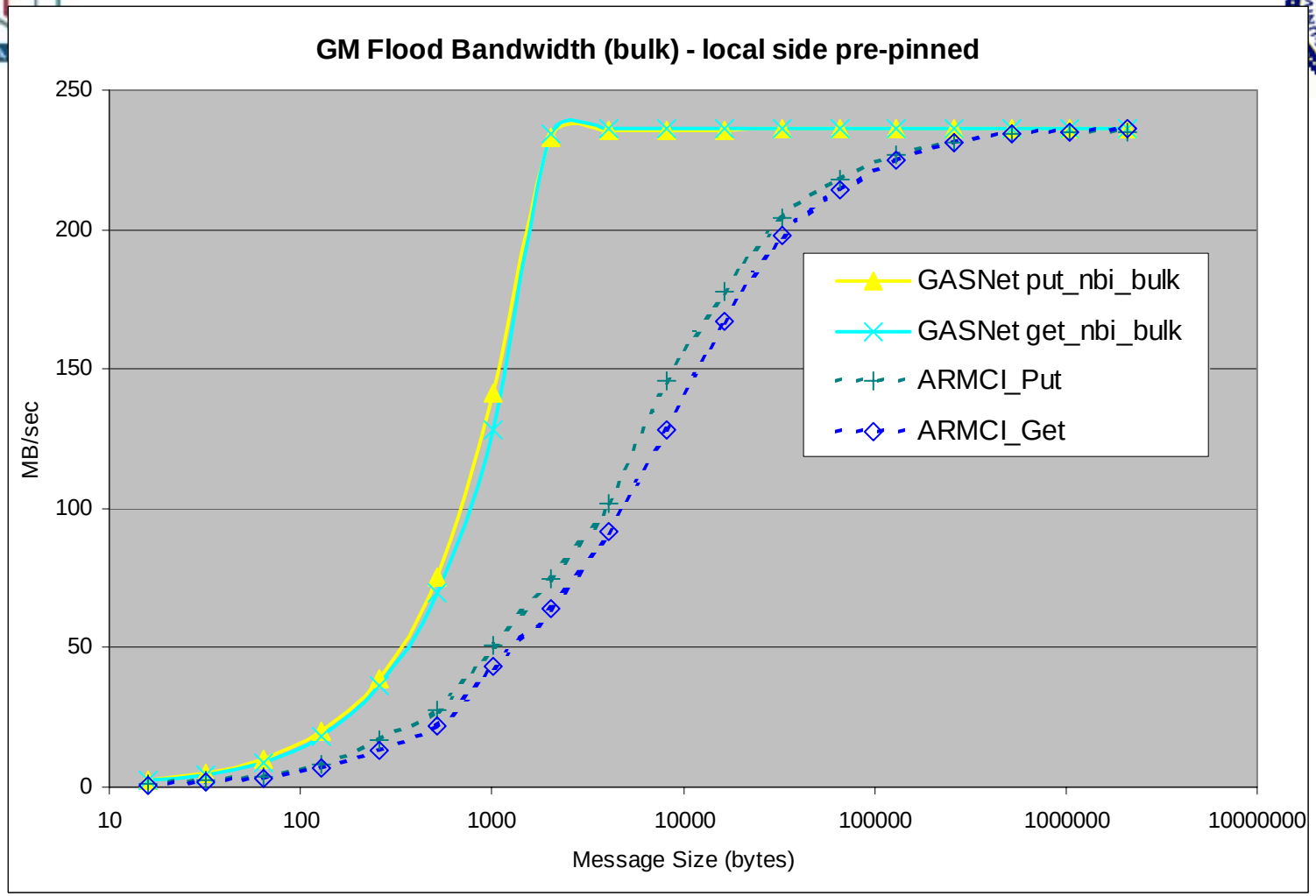
GM: Local mem pinned (upc_memcpy)



Simple GM_Put/GM_Get, so all configs basically run at hardware speed

GASNet about 0.3 us faster -
less function call overhead, code path carefully tuned for low latency in small operations

GM: Local mem pinned (upc_memcpy)

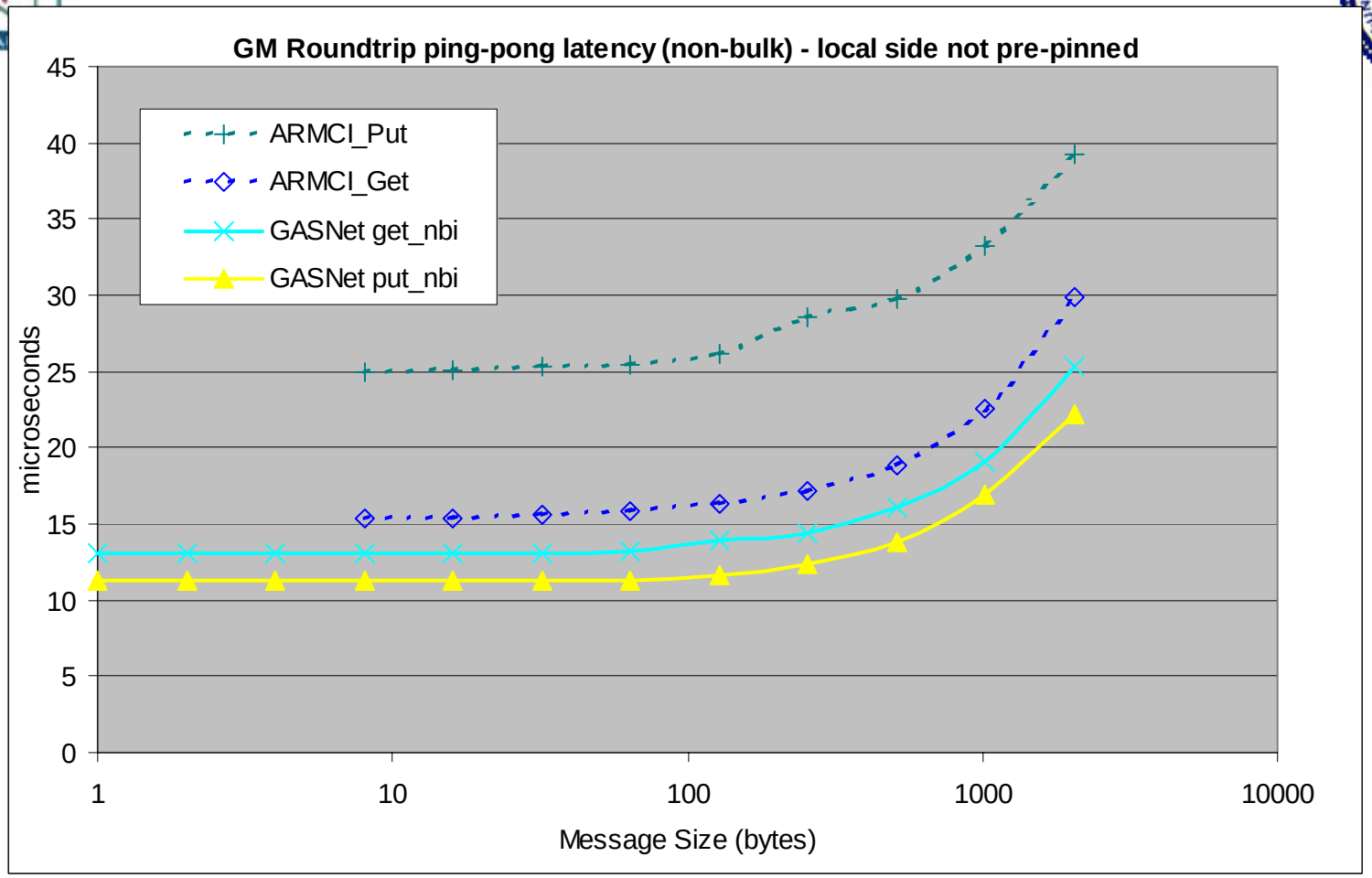


GASNet saturates at 2KB, ARMCI at 256KB

Huge difference is due to ARMCI "pseudo-blocking" semantics

ARMCI recently added non-blocking, but the impl. on Myrinet doesn't seem to work

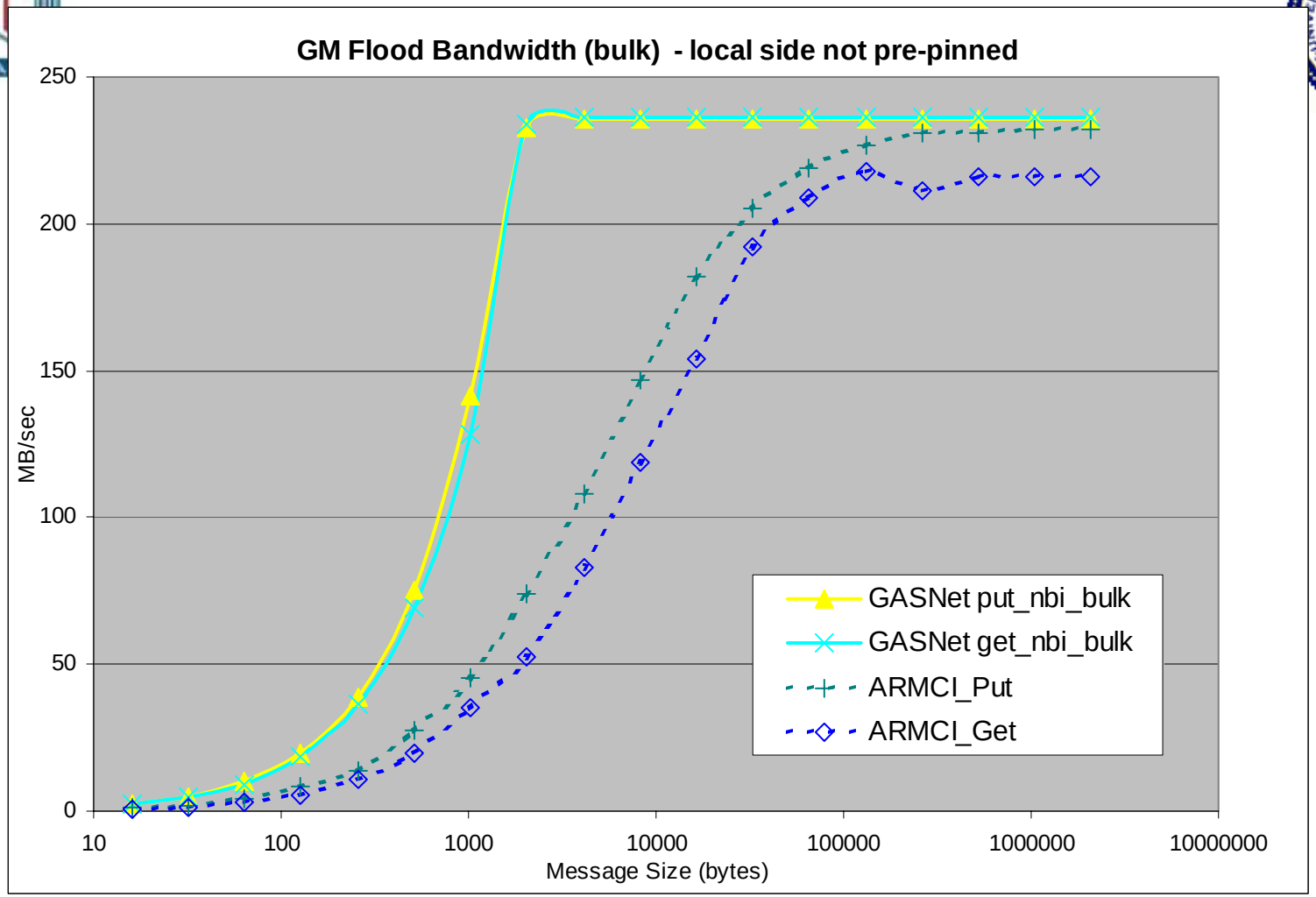
GM: Local mem not pinned (upc_memput/get)



GASNet uses firehose to dynamically pin the local pages

ARMCI falls back to a two?-copy message send scheme, pipelining through bounce buffers to a separate server process on the remote side - synchronization costs hurts small message latency

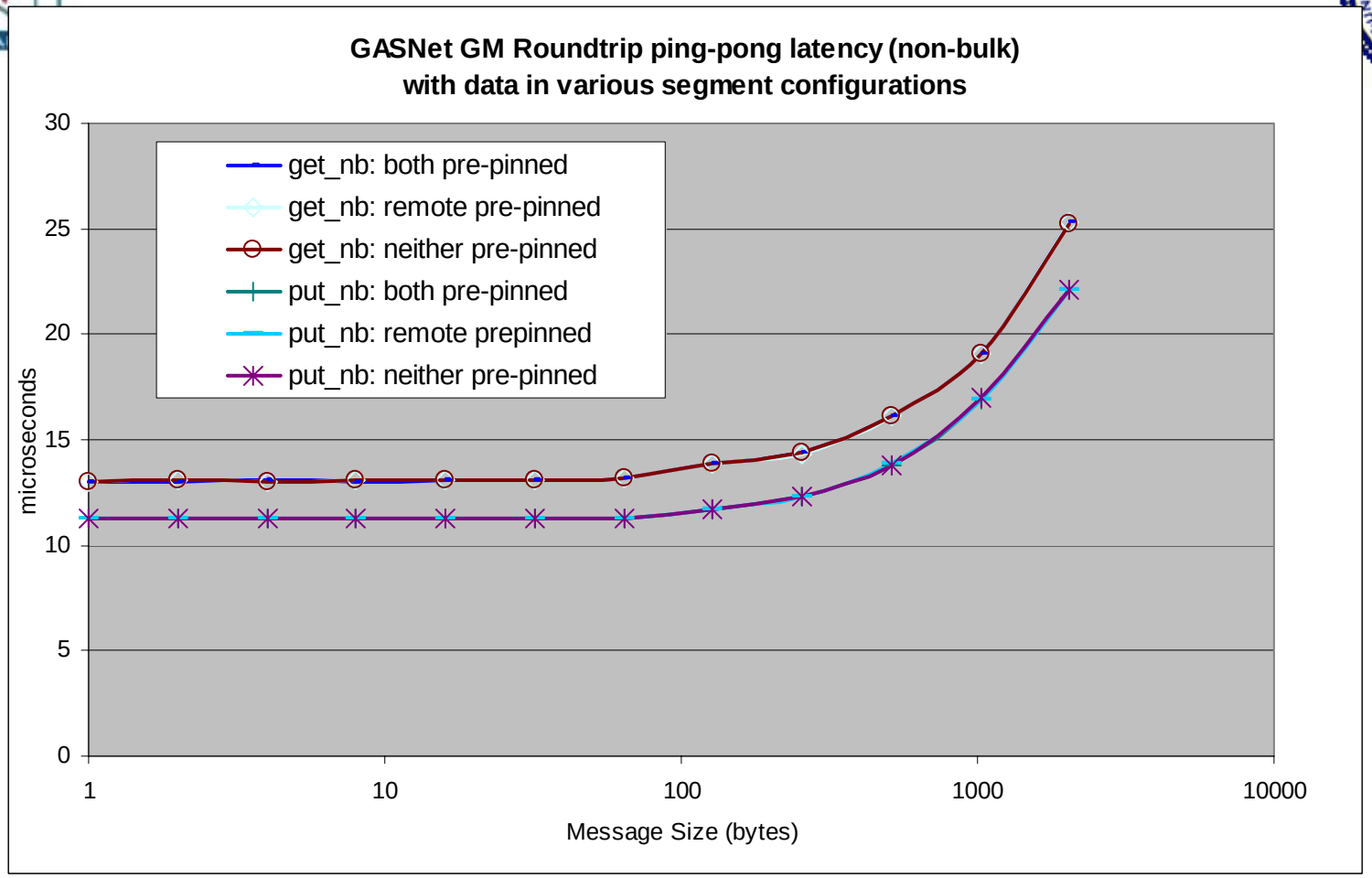
GM: Local mem not pinned (upc_memput/get)



GASNet bandwidth quickly saturates to the hardware max - dynamic pinning of local side pages to enable RDMA for large transfers, with lazy unpinning to amortize the pinning cost

ARMCI bounce buffer pipelining performs worse at small/medium sizes and reaches lower saturation bandwidth, especially for gets

GM: Firehose vs. Pre-pinned

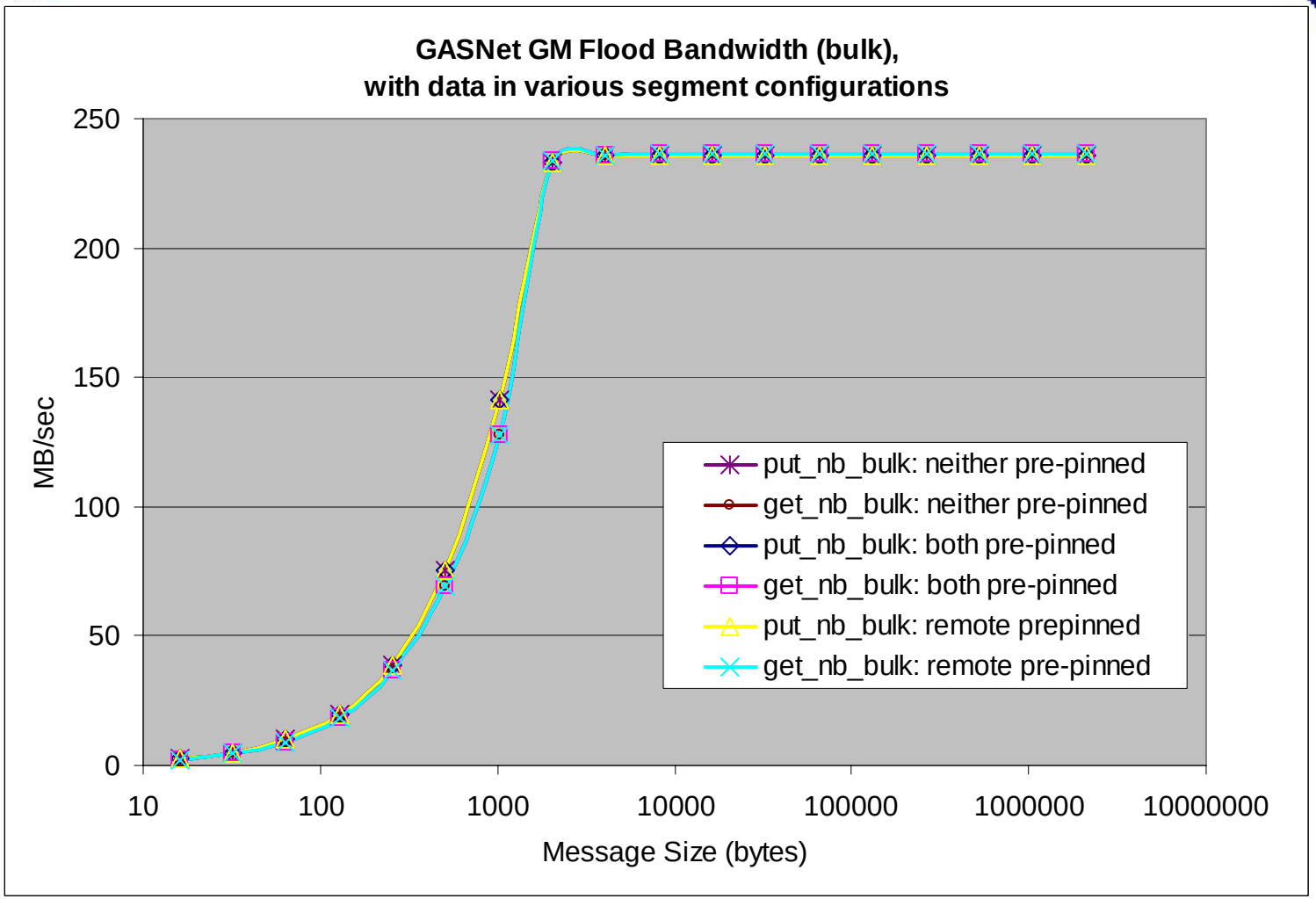


Firehose is well-tuned on GM - gives performance nearly identical to raw RDMA pre-pinned approach for access patterns with good temporal or spatial locality

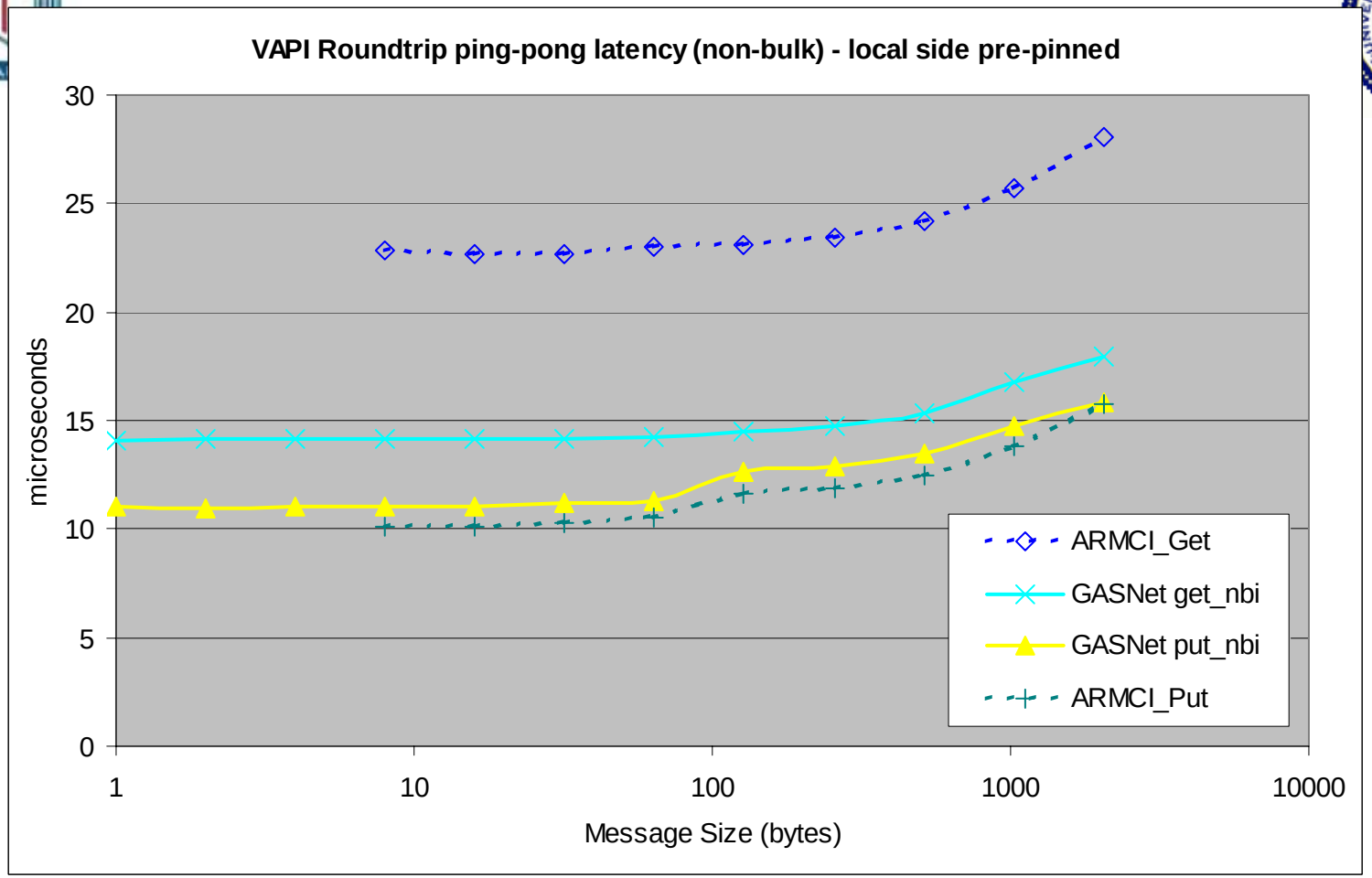
Allows Put/Get to anywhere in the VM space, but avoids limitations of pre-pinned approach

ARMCI forbids the case where remote memory is not collectively pre-pinned

GM: Firehose vs. Pre-pinned



VAPI: Local mem pinned (upc_memcpy)

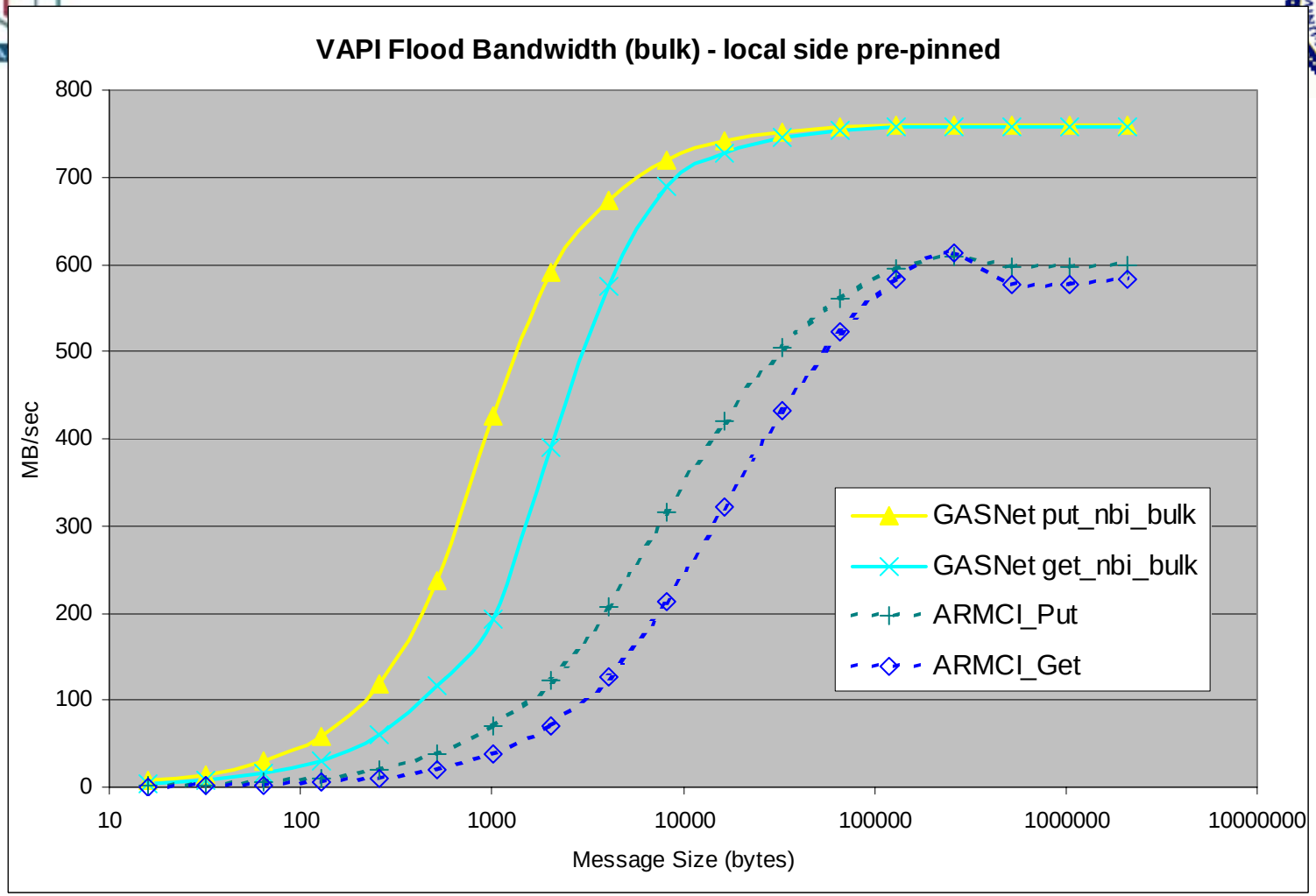


Simple RDMA Put/Get, so GASNet runs almost at hardware speed

Small firehose latency overhead (1us) for puts - needs more tuning

ARMCI Get has significantly worse latency - unclear why (performance bug?)

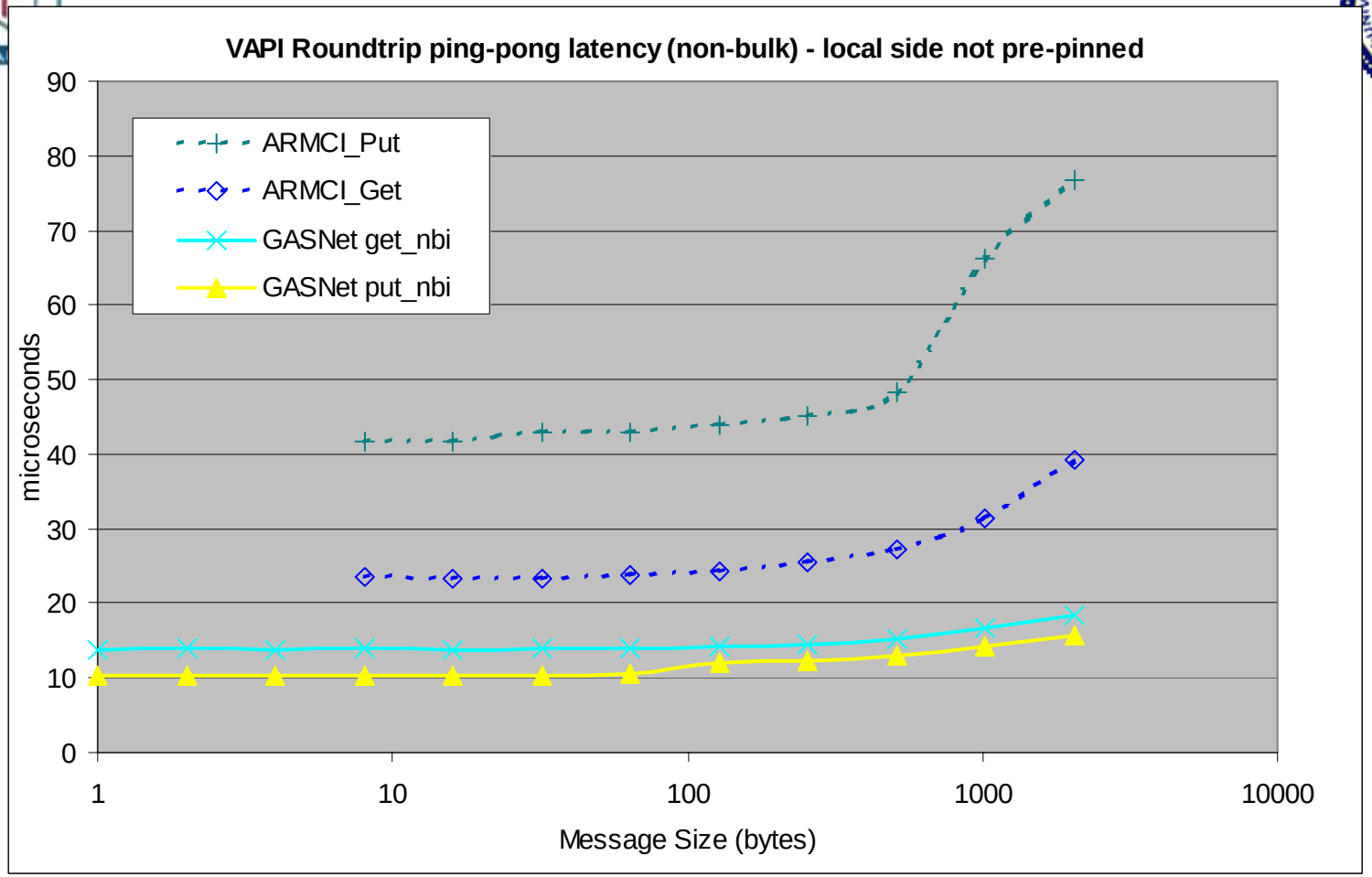
VAPI: Local mem pinned (upc_memcpy)



GASNet provides the raw hardware RDMA bandwidth - even beats the MVAPICH MPI-over-Infiniband (not shown here). Put/Get difference is a hardware characteristic.

ARMCI-VAPI appears to need more tuning

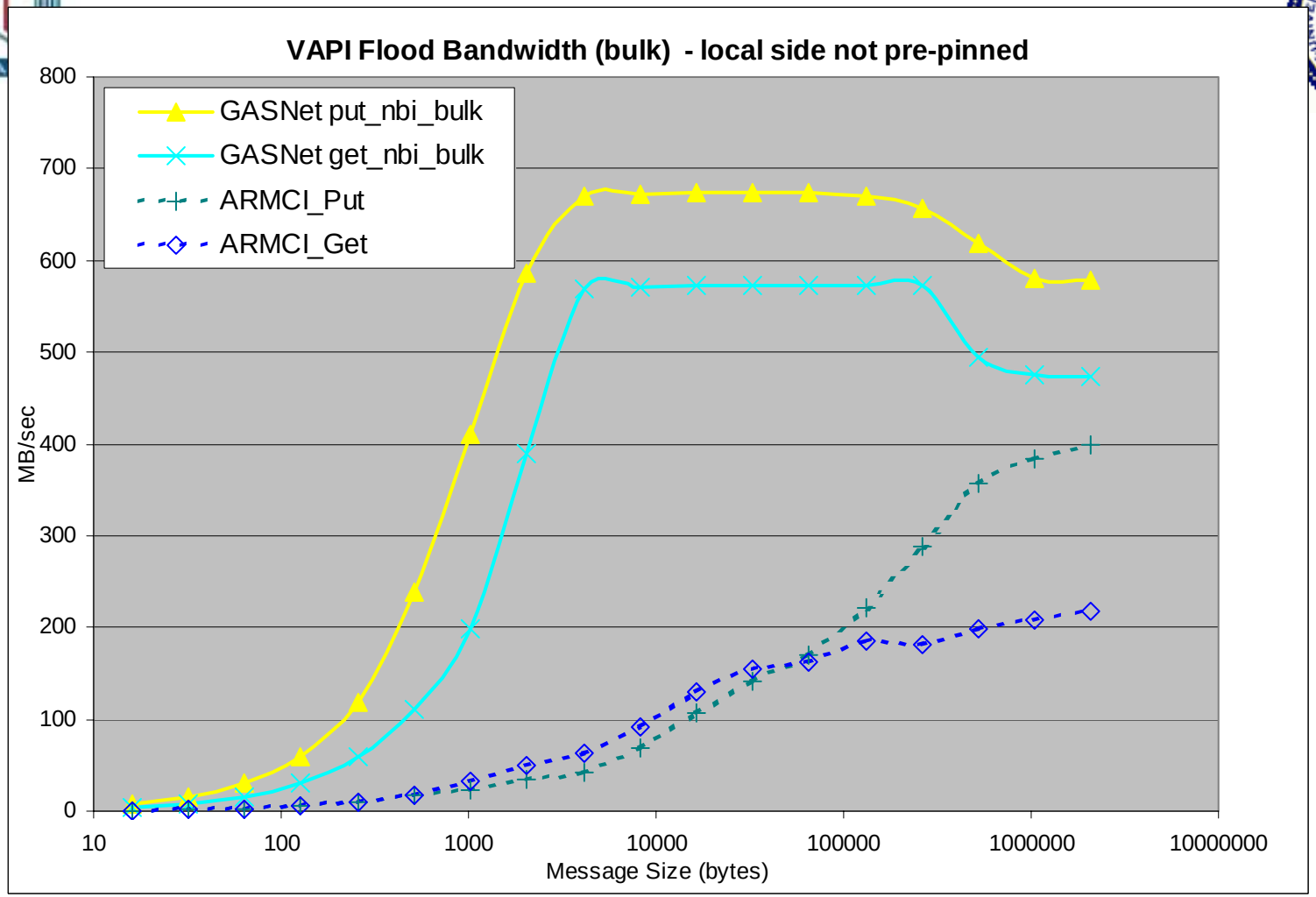
VAPI: Local mem not pinned (upc_memput/get)



GASNet one-sided pipelining through preallocated local-side bounce buffers - small memcpy overhead, then RDMA

ARMCI pipelining through bounce buffers appears to be involving the separate server process on the remote side - synchronization costs hurts small message latency

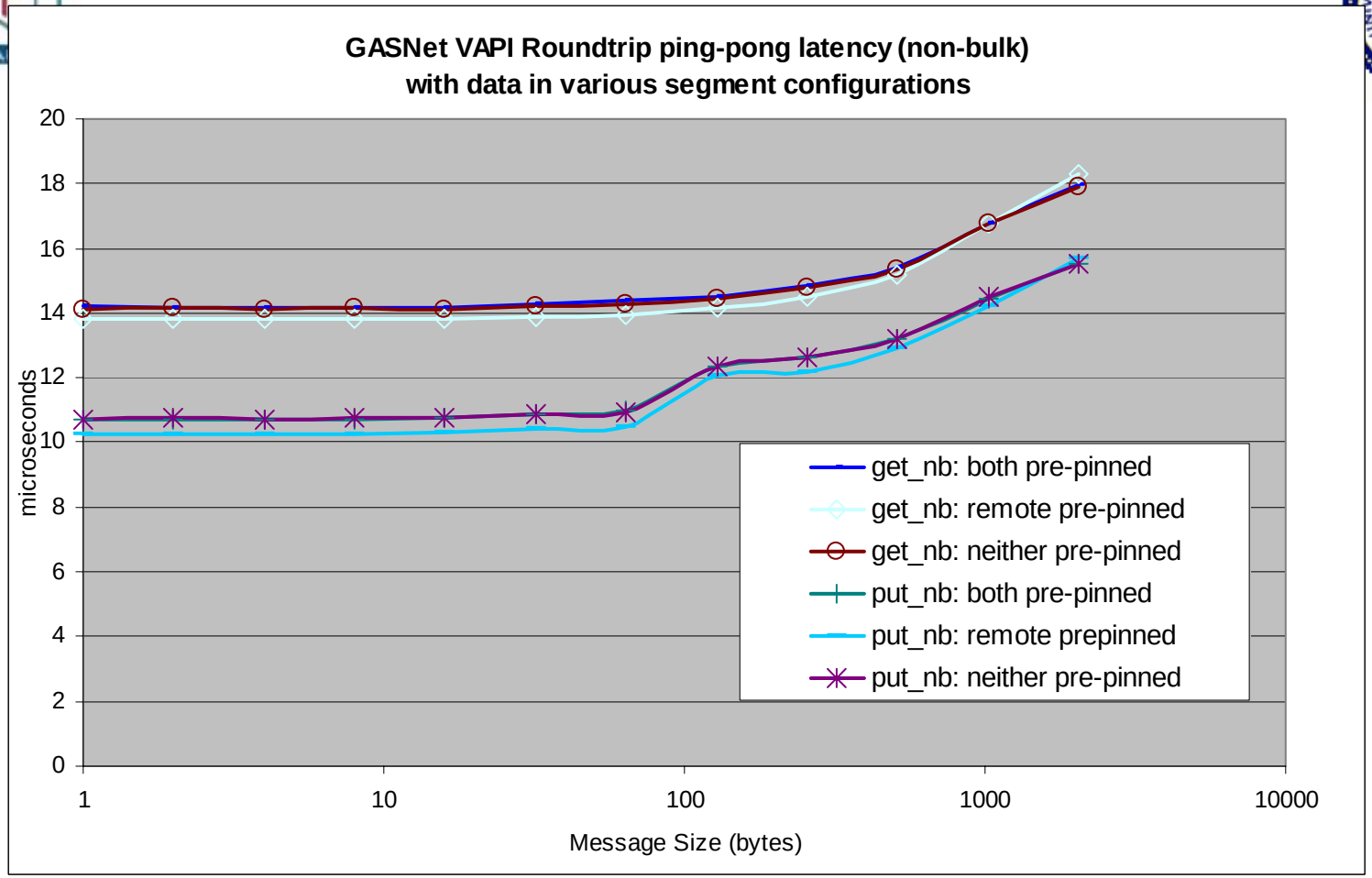
VAPI: Local mem not pinned (upc_memput/get)



GASNet bandwidth reduced when local side not pinned - pipelining through 4KB prepinned bounce buffers. Not using firehose dynamic pinning of local side pages, but probably should

ARMCI two-sided bounce buffer pipelining approach performs worse at all sizes and reaches lower saturation bandwidth, especially for gets

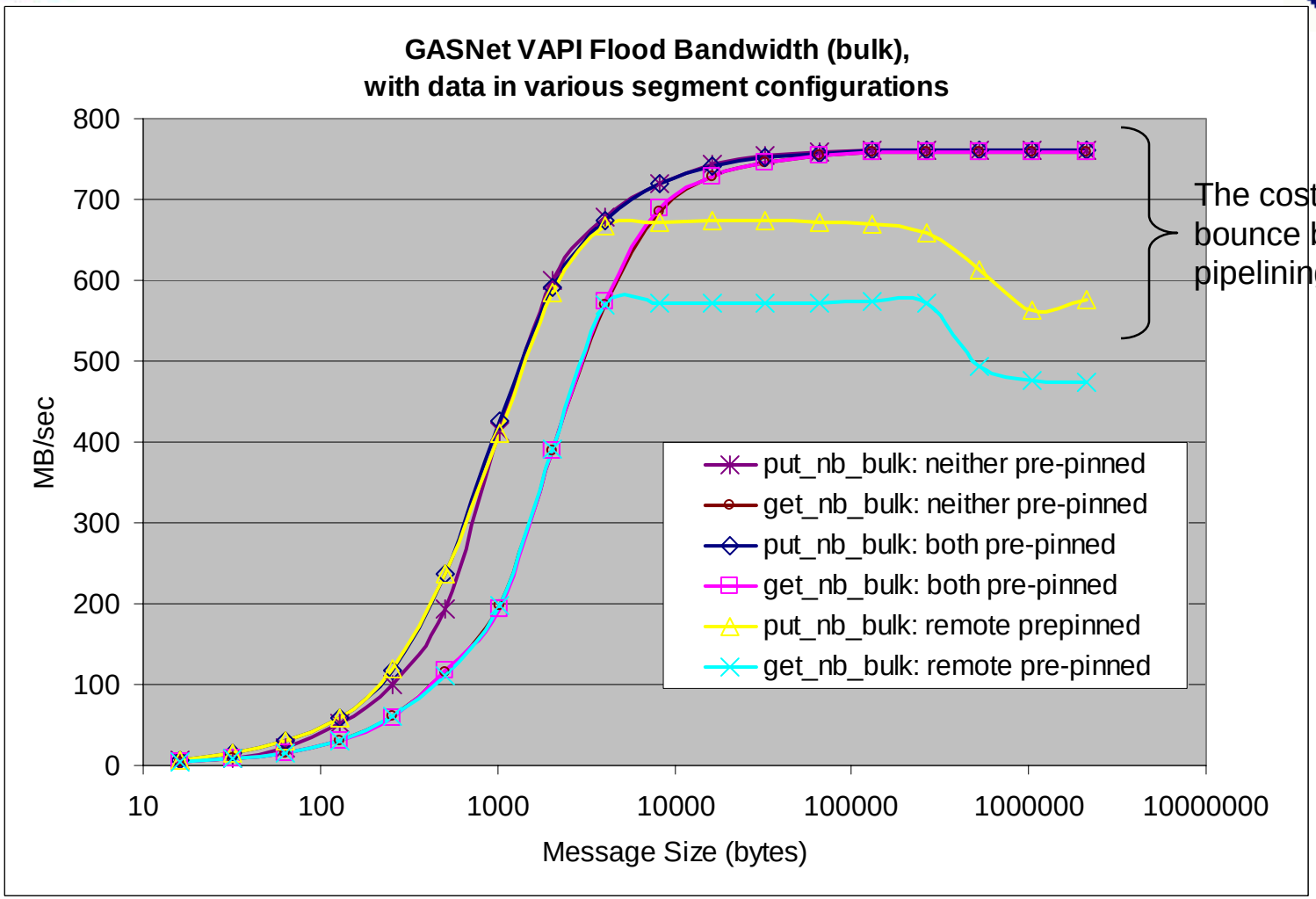
VAPI: Firehose vs. Pre-pinned



Firehose small-message latency very close to pre-pinned approach - small bookkeeping overheads

ARMCI forbids the case where remote memory is not collectively pre-pinned

VAPI: Firehose vs. Pre-pinned



"Put/Get with local not prepinned" currently using bounce buffer pipelining - should probably switch to firehose to get dynamic pinning of local pages

GASNet vs ARMCI conclusions

- Both layers run on most HPC systems of interest
  - Both natively target the major high-performance networks
  - Both have portable MPI/Ethernet based ports
- GASNet designed specifically for PGAS languages/compilers
  - Provides exactly the semantics needed for PGAS
  - Extensibility and bootstrapping features very important
- Preliminary performance comparisons very positive
  - GASNet can meet or exceed ARMCI performance
  - Hard work on Firehose gives us a significant advantage on pinning-based networks (Myrinet, Infiniband, Dolphin)
- GASNet has well-tuned non-blocking contiguous put/get
  - ARMCI has just added these
- ARMCI has well-tuned non-contiguous put/get
  - GASNet has just added these

GASNet Extensions for Non-Contiguous Remote Access

Vector (variable-length scatter/gather),

Indexed (fixed-length scatter/gather) &

Strided (regular non-contiguous)

a.k.a. "VIS"

Application Motivation for VIS

- Many applications have non-collective, non-contiguous (ie sparse) remote access patterns
  - irregular cases: SPMV, distributed graph data structures
  - regular cases: remote sub-array access (ghost value exchange)
- Most natural way to write these algorithms leads to a fine-grained comm.
  - naïve translation to individual remote accesses performs poorly on modern networks
- Want communication aggregation optimizations
  - Save by aggregating small messages into larger ones (ie pack/unpack), possibly with help from hardware
  - Allow sophisticated users to directly express aggregate non-contiguous communication
  - Provide compilation target so optimizer can express automated aggregation (ie message coalescing)

UPC Support for Non-Contiguous Remote Access

- Proposal for extending upc_mem{put,get,cpy} library (sent to UPC community list on Feb 10)
  - Includes orthogonal non-blocking extensions
  - New flavors of upc_mem{put,get,cpy}
  - See full proposal on the UPC publications page
- Vector
  - src and dst are each a list of variable-sized contiguous regions
- Indexed
  - src and dst are each a list of fixed sized contiguous regions
- Strided
  - src/dst are each a set of regularly sized and spaced regions
  - sufficient for expressing arbitrary rectangular sections over dense N-d arrays

Network Hardware Support for VIS

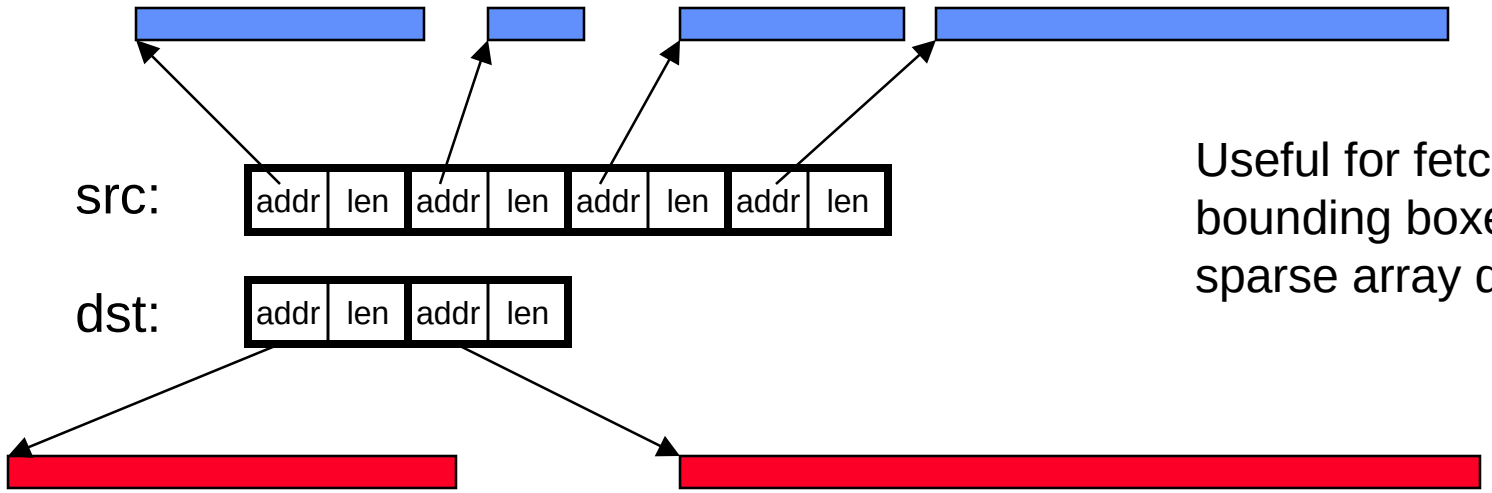
Hardware support for non-contiguous RMA varies widely:

Hardware	Vector	Indexed	Strided	RDMA
Cray X-1		YES		N/A
Quadrics Elan		YES		YES
Infiniband	local-side	local-side		YES
Myrinet (MX)	YES			YES
IBM LAPI	YES	YES	YES	NO

want the library/compiler VIS features to exploit the hardware support where available, without rewriting the compiler for each platform

New GASNet Interfaces for Non-Contiguous Remote Access

- **Vector** - src/dst are list of variable length contiguous regions:

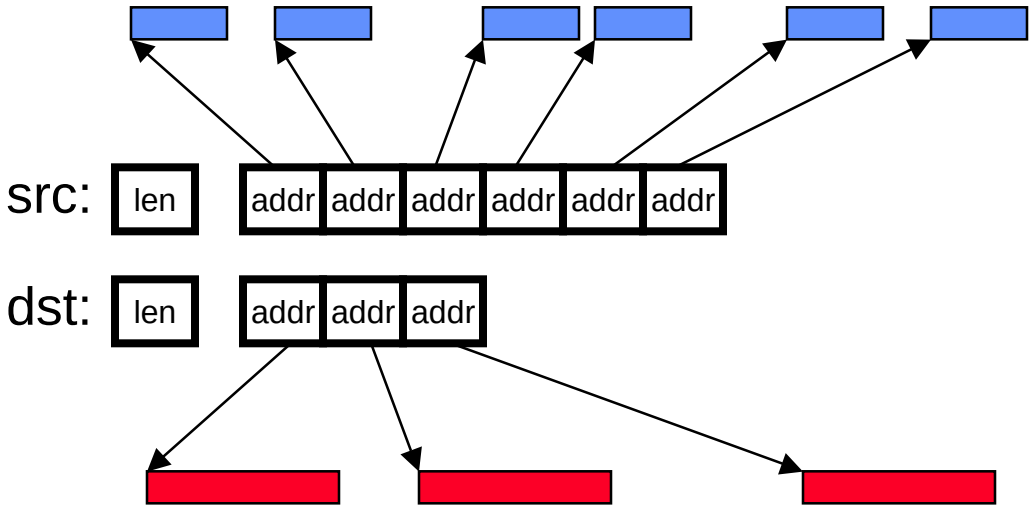


Useful for fetching bounding boxes, sparse array data

- Source/Destination region counts and sizes may differ
  - only total data sz must match
- Most general and flexible option - least hardware support
- Blocking and non-blocking variants (explicit & implicit handle)

New GASNet Interfaces for Non-Contiguous Remote Access

- **Indexed** - list of fixed-length contiguous regions:

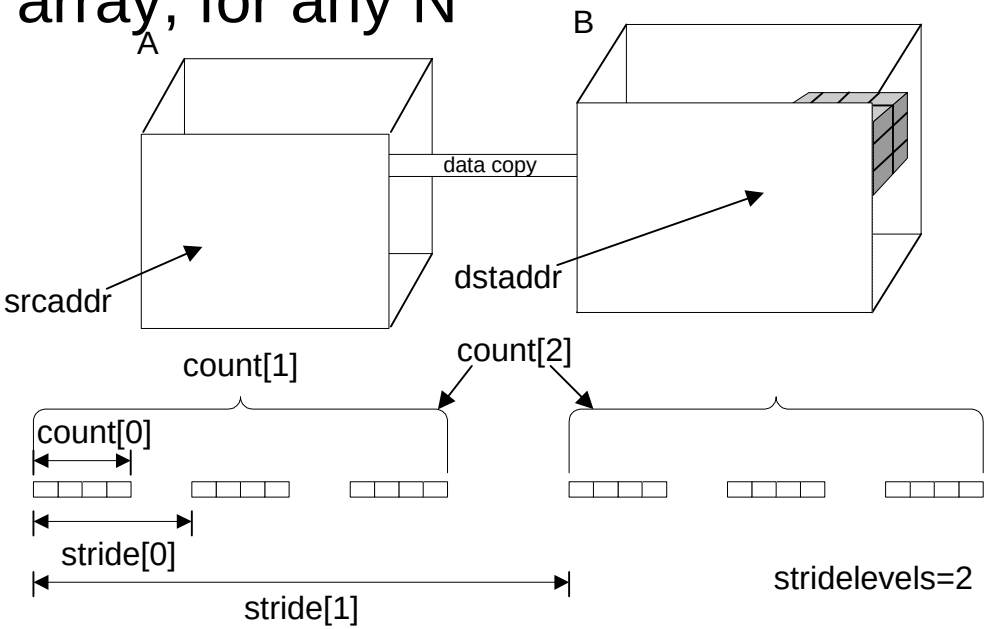


Useful for fetching irregular set of array elements, inspector/executor

- More restrictive than vector interface
  - Less metadata due to fixed size
  - Closer to most available network hardware support
- Also have non-blocking variants (explicit/implicit handle)

New GASNet Interfaces for Non-Contiguous Remote Access

- **Strided**: regularly spaced/sized accesses
- src/dst specify an arbitrary rectangular section on an N-d dense array, for any N



Useful for fetching any non-contiguous section of a dense array

- Most restrictive access pattern - least metadata
  - metadata size is linear in dimensionality (N)
- Also have non-blocking variants (explicit/implicit handle)

GASNet VIS Implementation Status

- Reference implementation underway
  - In terms of existing put/get (RDMA) - done
  - In terms of each other (eg strided over indexed) - done
  - In terms of core API Active Messages - in progress
  - Internally maintain many different algorithmic options to allow experimentation and tuning
  - Select algorithm based on hardware characteristics, transfer parameters (size, sparsity, etc) and current network status
- Completed GASNet VIS hook-up to runtime & UPC library source level
  - Translator can generate VIS calls - message coalescing
  - Berkeley UPC users can already call them as a library
  - Still pushing for lang. acceptance of memcpy extensions

GASNet VIS Future Work

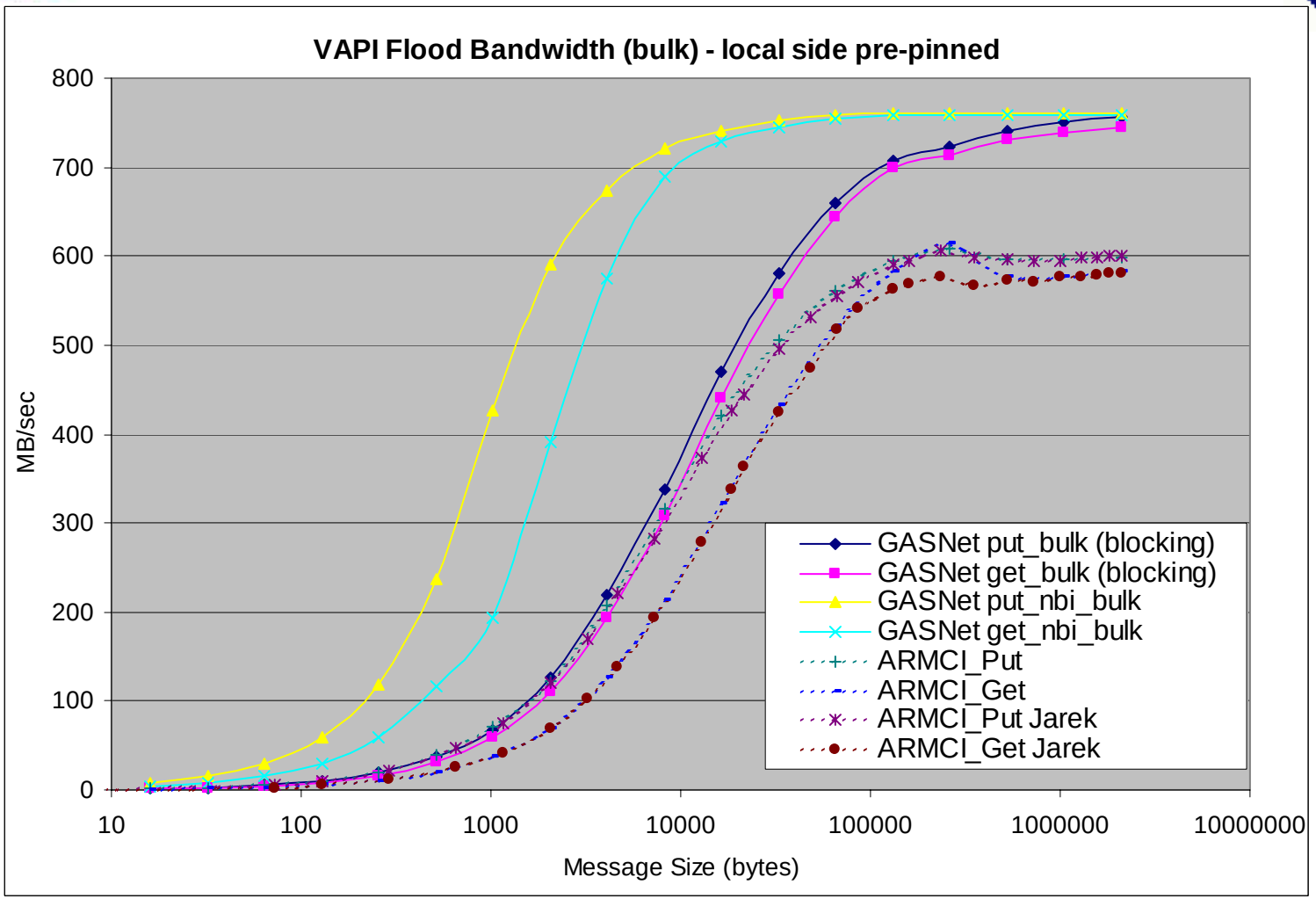
- Network-specific implementations and hardware exploitation next
  - Use the reference implementation as a starting point
  - Directly leverage available hardware support to tune
  - Starting with Quadrics/Elan4 (FY04)
  - Move on to VIS support over Myrinet/MX, Infiniband/VAPI (local-side only), IBM/LAPI (software), Cray X1 (vector load/store)
- Hook up to Titanium array library
- Investigate compiler-generated VIS calls
  - message coalescing, inspector/executor
- Performance experimentation & tuning
  - Microbenchmarks
  - Application-level benchmarks
    - Programmer-inserted calls to VIS functionality
    - Compiler-generated VIS calls


BERKELEY LAB

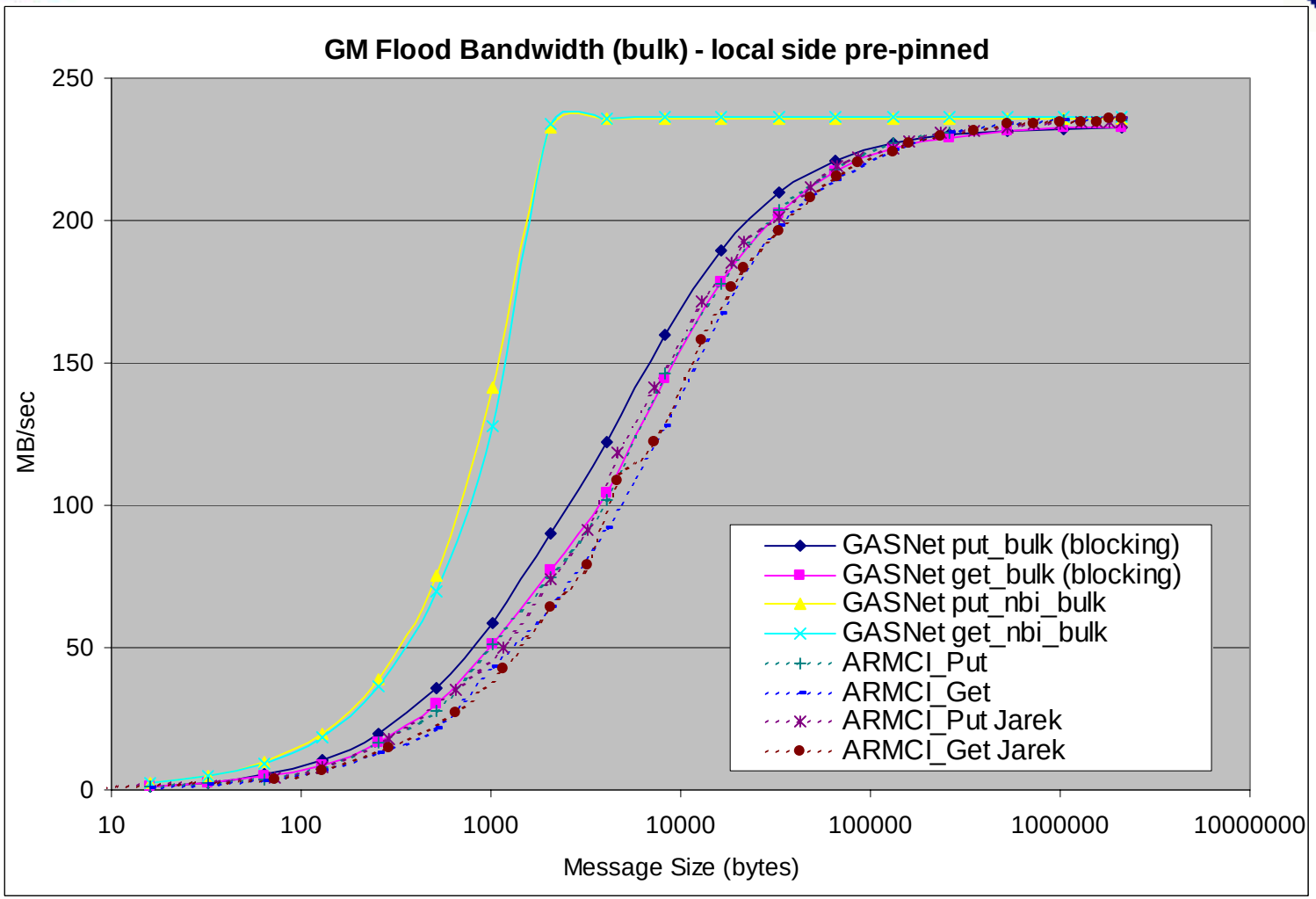

THE UNIVERSITY OF CALIFORNIA BERKELEY
1868

EXTRA SLIDES

VAPI: Full Data, both pre-pinned



GM: Full Data, both pre-pinned



GASNet/ARMCI Engineering Issues

- System size:
  - ARMCI: 33426 LOC, 3371 lines of comments (10%)
  - GASNet: 76081 LOC, 17590 lines of comments (23%)
- High-level design
  - ARMCI:
    - Grown evolutionarily, no configure script (painful to install)
    - code is messy - all the networks and platforms are interleaved in the same poorly commented files
  - GASNet:
    - Designed from scratch for low-latency/overhead for small put/gets and high bandwidth zero-copy for large put/gets, one-sided operation
    - Layered design with clean, well-documented internal interfaces and a template conduit to streamline porting and conduit creation

Lines of Code breakdown

ARMCI: 33426 LOC, 3371 lines of comments (10%)

GASNet: 76081 LOC, 17590 lines of comments (23%)

shared infrastructure: 14788

9438 (top) + 651 other + 4699 extended ref

firehose 6708

vapi 7439

lapi 5117

gm 8338

elan 5344

smp 1677

mpi 6497 = 1780 + AMMPI 4717

udp 10820 = 1761 + AMUDP 9059

template 1512

dolphin 4279

shmem/X1 3562

Core API – Active Messages

- Super-Lightweight RPC
  - Unordered, reliable delivery
  - Matched request/reply serviced by "user"-provided lightweight handlers
  - General enough to implement almost any communication pattern
- Request/reply messages
  - 3 sizes: short (<=32 bytes),medium (<=512 bytes), long (DMA)
- Very general - provides extensibility
  - Available for implementing compiler-specific operations
  - scatter-gather or strided memory access, remote allocation, etc.
- AM previously implemented on a number of interconnects
  - MPI, LAPI, UDP/Ethernet, Via, Myrinet, and others
- Includes mechanism for explicit atomicity control in handlers
  - Even in the presence of interrupts & multithreading
  - Handler-safe locks & no-interrupt sections

Extended API – Remote memory operations

- Orthogonal, expressive, high-performance interface
  - Gets & Puts for Scalars and Bulk contiguous data
  - Blocking and non-blocking (returns a handle)
  - Also have a non-blocking form where the handle is implicit
- Non-blocking synchronization
  - Sync on a particular operation (using a handle)
  - Sync on a list of handles (some or all)
  - Sync on all pending reads, writes or both (for implicit handles)
  - Sync on operations initiated in a given interval
  - Allow polling (trysync) or blocking (waitsync)
- Useful for experimenting with a variety of parallel compiler optimization techniques

Extended API – Remote memory operations

- API for remote gets/puts:

```
void   get    (void *dest, int node, void *src, int numbytes)
handle get_nb (void *dest, int node, void *src, int numbytes)
void   get_nbi(void *dest, int node, void *src, int numbytes)

void   put    (int node, void *src, void *dest, int numbytes)
handle put_nb (int node, void *src, void *dest, int numbytes)
void   put_nbi(int node, void *src, void *dest, int numbytes)
```

- "nb"/"nbi" = non-blocking with explicit/implicit handle
- Also have "value" forms that are register-memory, and "bulk" forms optimized for large memory transfers
- Extensibility of core API allows easily adding other more complicated access patterns (scatter/gather, strided, etc)

Extended API – Remote memory operations

- API for get/put synchronization:
- Non-blocking sync with explicit handles:

```
int  try_syncnb(handle)
void wait_syncnb(handle)

int  try_syncnb_some(handle *, int numhandles)
void wait_syncnb_some(handle *, int numhandles)
int  try_syncnb_all(handle *, int numhandles)
void wait_syncnb_all(handle *, int numhandles)
```

- Non-blocking sync with implicit handles:

```
int  try_syncnbi_gets()
void wait_syncnbi_gets()
int  try_syncnbi_puts()
void wait_syncnbi_puts()
int  try_syncnbi_all()  // gets & puts
void wait_syncnbi_all()
```

Network Hardware Support for Non-Contiguous Remote Access

- Cray X-1
  - fixed-size indexed load/stores
- Quadrics libelan
  - recently added fixed-size indexed put-get RDMA
- Infiniband
  - local-side gather sends and scatter recvs
- Myrinet
  - planned support for variable-sized vector put-get RDMA in new MX interface
- IBM LAPI
  - variable-size vector put/get/am and strided put/get (*not* RDMA)

want the library/compiler VIS features to exploit the hardware support where available, without rewriting the compiler for each platform

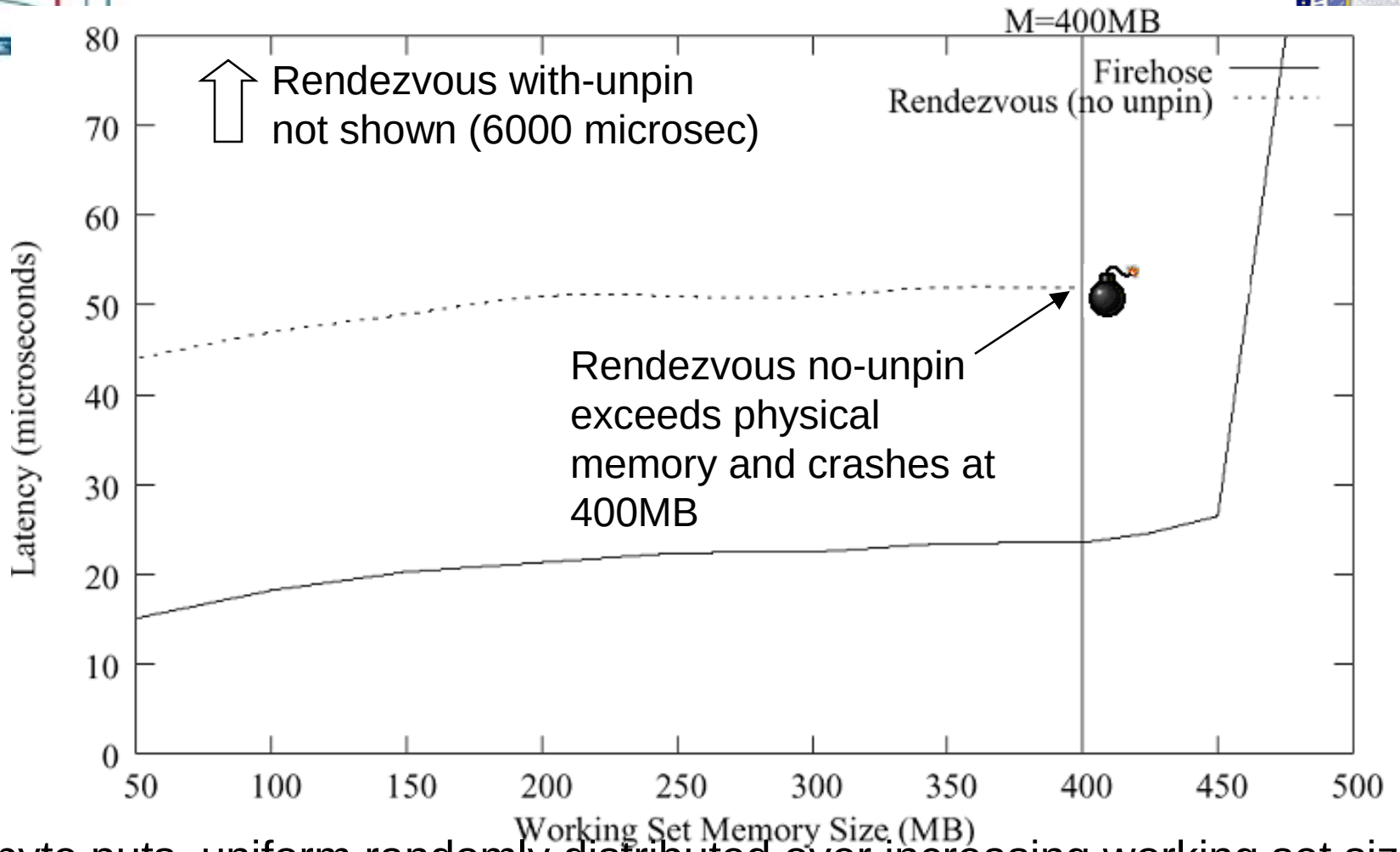
Basic Idea: A Hybrid Approach

- 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 operations
  - Each node sets aside M bytes of physical memory for registration purposes (some reasonable fraction of phys mem)
  - Guarantee F = $\left\lfloor \frac{M}{pagesize*(nodes-1)} \right\rfloor$ 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: 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+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 < 1μs

Performance Results: "Worst-case" Roundtrip Put Latency



- 8-byte puts, uniform randomly distributed over increasing working set size
  - worst-case temporal and spatial locality
- Firehose degrades to match Rendezvous with-unpin performance once physical memory exhausted and unpins are required

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 on one or both sides
  - No registration cost at runtime, Full memory space accessible
  - Often 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)

Code Generation Tradeoffs

- Blocking vs. Non-blocking puts/gets
- Put/Get variety: non-bulk vs. bulk
  - optimized for small scalars vs large zero-copy
  - difference in semantics - put src, alignment
- Put/Get synchronization mechanism
  - expressiveness/complexity tradeoffs
  - explicit handle vs. implicit handle, access regions
- Work remains to explore these tradeoffs in the context of code generation

Titanium Language Support for Non-Contiguous Remote Access

- Titanium N-d Array Library
  - Powerful and flexible language support for directly expressing many high-level array operations on array descriptors
  - Many regular N-d operations lead to non-contiguous access:
    - Restrict, Slice, Permute (transpose)
- Recent irregular extensions to array library
  - Sparse array copy - over irregular Domains and Point lists
  - Scatter/Gather copy - to/from 1-D contiguous buffers
- Implementation status:
  - Fully implemented on Active Messages and contiguous RDMA
  - Want to take advantage of available hardware support for non-contiguous remote access, to improve performance
- Work on automated aggregation optimizations underway

GASNet Vector Interface

- **Vector** - list of variable length contiguous regions:

```
typedef struct {
    void *addr;
    size_t len;
} gasnet_memvec_t;

void gasnet_putv_bulk(gasnet_node_t dstnode,
            size_t dstcount, gasnet_memvec_t const dstlist[],
            size_t srccount, gasnet_memvec_t const srclist[]);
void gasnet_getv_bulk(size_t dstcount, gasnet_memvec_t const dstlist[],
            gasnet_node_t srcnode,
            size_t srccount, gasnet_memvec_t const srclist[]);
```

- Also have non-blocking variants (explicit & implicit handle)
- Source/Dest region sizes may differ - only total data sz must match
- Most general and flexible option - least hardware support

GASNet Indexed Interface

- Indexed - list of fixed-length contiguous regions:

```
void gasnet_puti_bulk(gasnet_node_t dstnode,
            size_t dstcount, void * const dstlist[], size_t dstlen,
            size_t srccount, void * const srclist[], size_t srclen);
void gasnet_geti_bulk(
            size_t dstcount, void * const dstlist[], size_t dstlen,
            gasnet_node_t srcnode,
            size_t srccount, void * const srclist[], size_t srclen);
```

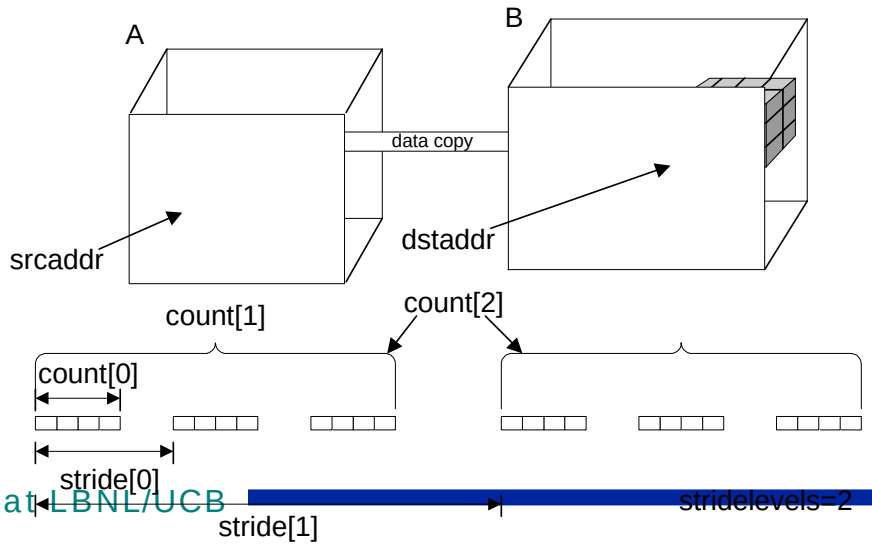
- More restrictive than vector interface
  - Less metadata due to fixed size
  - Closer to most available network hardware support
- Also have non-blocking variants (explicit/implicit handle)

GASNet Strided Interface

- Strided: regularly spaced/sized accesses

```
void gasnet_puts_bulk(gasnet_node_t dstnode,
                      void *dstaddr, const size_t dststrides[],
                      void *srcaddr, const size_t srcstrides[],
                      const size_t count[], size_t stridelevels);
void gasnet_gets_bulk(void *dstaddr, const size_t dststrides[],
                      gasnet_node_t dstnode,
                     void *srcaddr, const size_t srcstrides[],
                     const size_t count[], size_t stridelevels);
```

- Also have non-blocking variants (explicit/implicit handle)
- Most restrictive access pattern - minimal metadata
- Sufficient to express src/dst regions which are an arbitrary rectangular section on an N-d dense array



UPC Vector Interface


```
typedef struct {
    void *addr;
    size_t len;
} upc_pmemvec_t;
typedef struct {
    shared void *addr; // treated as a (shared [] char *) - ie. no wrapping
    size_t len;
} upc_smemvec_t;

void upc_memcpy_list(size_t dstcount, upc_smemvec_t const dstlist[],
                                      size_t srccount, upc_smemvec_t const srclist[]);
void upc_memput_list(size_t dstcount, upc_smemvec_t const dstlist[],
                                      size_t srccount, upc_pmemvec_t const srclist[]);
void upc_memget_list(size_t dstcount, upc_pmemvec_t const dstlist[],
                                     size_t srccount, upc_smemvec_t const srclist[]);
```

UPC Vector Example


```
#define BLKSZ 100
shared [BLKSZ] double A[BLKSZ*THREADS]; /* assume THREADS >= 3 */
upc_smemvec_t srclist[] = {
      { &(A[14]), sizeof(double) }, /* element 14 (from thread 0) */
      { &(A[20]), sizeof(double) }, /* element 20 (from thread 0) */
      { &(A[100]), 50*sizeof(double) }, /* elements 100..149 (from thread 1) */
      { &(A[2*BLKSZ]), BLKSZ*sizeof(double) } /* entire block (from thread 2) */
};
double mybuf[52+BLKSZ];
upc_pmemvec_t dstlist[] = { { mybuf, sizeof(mybuf) } };
upc_memget_list(1, dstlist, 4, srclist);
/* compute on contents of mybuf */
```

UPC Indexed Interface


```
void upc_memcpy_list(size_t dstcount, shared void * const dstlist[], size_t dstlen,
                     size_t srccount, shared const void * const srclist[], size_t srclen);
void upc_memput_list(size_t dstcount, shared void * const dstlist[], size_t dstlen,
                     size_t srccount, const void * const srclist[], size_t srclen);
void upc_memget_list(size_t dstcount, void * const dstlist[], size_t dstlen,
                     size_t srccount, shared const void * const srclist[], size_t srclen);

#define BLKSZ 100
shared [BLKSZ] double A[BLKSZ*THREADS]; /* assume THREADS >= 2 */
shared void * srclist[] = {
    &(A[14]), &(A[15]), &(A[16]), /* element 14..16 (from thread 0) */
    &(A[100]), &(A[110]) /* element 100 and 110 (from thread 1) */
};
double mybuf[5];
void * dstlist[] = { &mybuf };
upc_memget_list(1, dstlist, 5*sizeof(double), 5, srclist, sizeof(double));
/* compute on contents of mybuf */
```

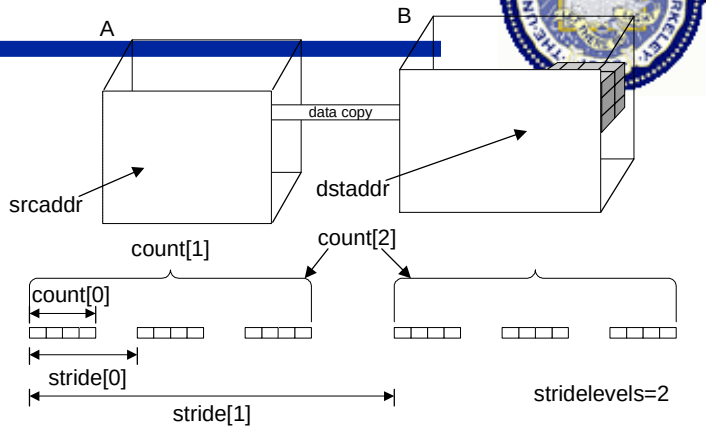
UPC Strided Interface


```
void upc_memcpy_strided(shared void *dstaddr, const size_t dststrides[],
                        shared const void *srcaddr, const size_t srcstrides[],
                        const size_t count[], size_t stridelevels);

void upc_memput_strided(shared void *dstaddr, const size_t dststrides[],
                        const void *srcaddr, const size_t srcstrides[],
                        const size_t count[], size_t stridelevels);

void upc_memget_strided( void *dstaddr, const size_t dststrides[],
                        shared const void *srcaddr, const size_t srcstrides[],
                        const size_t count[], size_t stridelevels);
```

UPC Strided Example



Example: To put a 3-d block of data, shaped 2x3x4, starting at location (5, 6, 7) in A to B in location (8, 9, 10):

```
double A[11][12][13]; /* local array */
shared [] double B[14][15][16]; /* remote array */

srcaddr = &(A[5][6][7]);
srcstrides[0] = 13 * sizeof(double); /* stride in bytes for the rightmost dimension */
srcstrides[1] = 12 * 13 * sizeof(double); /* stride in bytes for the middle dimension */
dstaddr = &(B[8][9][10]);
dststrides[0] = 16 * sizeof(double); /* stride in bytes for the rightmost dimension */
dststrides[1] = 15 * 16 * sizeof(double); /* stride in bytes for the middle dimension */
count[0] = 4 * sizeof(double); /* bytes of contig data (width in rightmost dimension) */
count[1] = 3; /* width in middle dimension */
count[2] = 2; /* width in leftmost dimension */
stridelevels = 2;
upc_memput_strided(srcaddr, dststrides, dstaddr, srcstrides, count, stridelevels);
```

UPC Strided Example