# Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations

Dan Bonachea & Jason Duell

U. C. Berkeley / LBNL

*http://upc.lbl.gov*

# Overview

- We are implementing distributed parallel languages
- MPI is the dominant HPC communication paradigm today for distributed systems
  - Has been touted as an "assembly language for parallel processing" [Gropp, 2001]
  - Would like to use MPI as a portable compilation target for parallel compilers - seems like a natural choice at first glance
- MPI 1.1 (matched send/recv)
  - Two-sided paradigm does not efficiently support one-sided access
  - Fails to expose the high-performance features of modern SAN's (eg. one-sided RDMA) in a useful way
  - Much more efficient to target the underlying vendor network API
- MPI 2.0 RMA (Remote Memory Access)
  - Overly restrictive semantics governing memory access patterns prevent any reasonable implementation strategies

# Background - GAS Languages

- Global Address Space (GAS) languages
  - UPC, Titanium, Co-Array Fortran
- Shared memory abstraction with explicit data layout
  - Threads can implicitly access remote shared data with simple language-level operations (assign/deref)
  - Programmer has explicit knowledge and control over data locality
- Relevant features of GAS languages
  - Non-collective dynamic allocation of shared memory
    - Most or all of the data is potentially shared (think GB's / node)
  - One-sided communication pattern
    - Pattern may be data-dependent, not statically predictable (irregular apps)
  - May access local shared memory via "local" pointers
    - Statically indistinguishable from non-shared (private) memory access
  - Incremental development model
    - Start with naïve app written in shared memory style and tune the bottlenecks
    - Many GAS apps tend to be sensitive to network latencies, at least initially

# Communication Needs of GAS Languages

- One-sided communication (puts, gets, etc)
  - Only initiator is involved - operations proceed independent of any explicit actions by the target (e.g. no calling recv or pin)
  - Can be simulated using two-sided, but must be transparent to client
- Low round-trip latency for small (~8 byte) messages
- Non-blocking remote memory access
  - allows overlapping to hide network latencies
  - need low CPU overhead and network gap to be effective
- Support arbitrary, unpredictable access patterns to shared memory
  - concurrent access to same region of memory by different remote nodes and the local node (possibly via "local" pointers)

# Implementing GAS over MPI 1.1

- MPI 1.1
  - the most widely-implemented HPC communications interface available today
  - portable and highly tuned by vendors (at least for expected common usage cases)
  - All point-to-point messages require strictly matching send/recv operations (two-sided messaging)
- Simulating one-sided messaging over MPI 1.1
  - AMMPI - Berkeley Active Messages over MPI 1.1
  - Uses non-blocking sends & recvs to avoid deadlock
  - Periodic polling within communication ops and library blocking calls - ensures progress and provides the illusion of one-sided
  - Successfully used to run UPC and Titanium on dozens of modern supercomputers and clusters - portability allows quick prototyping

# Network Latency and Overhead Comparison for Popular SAN's

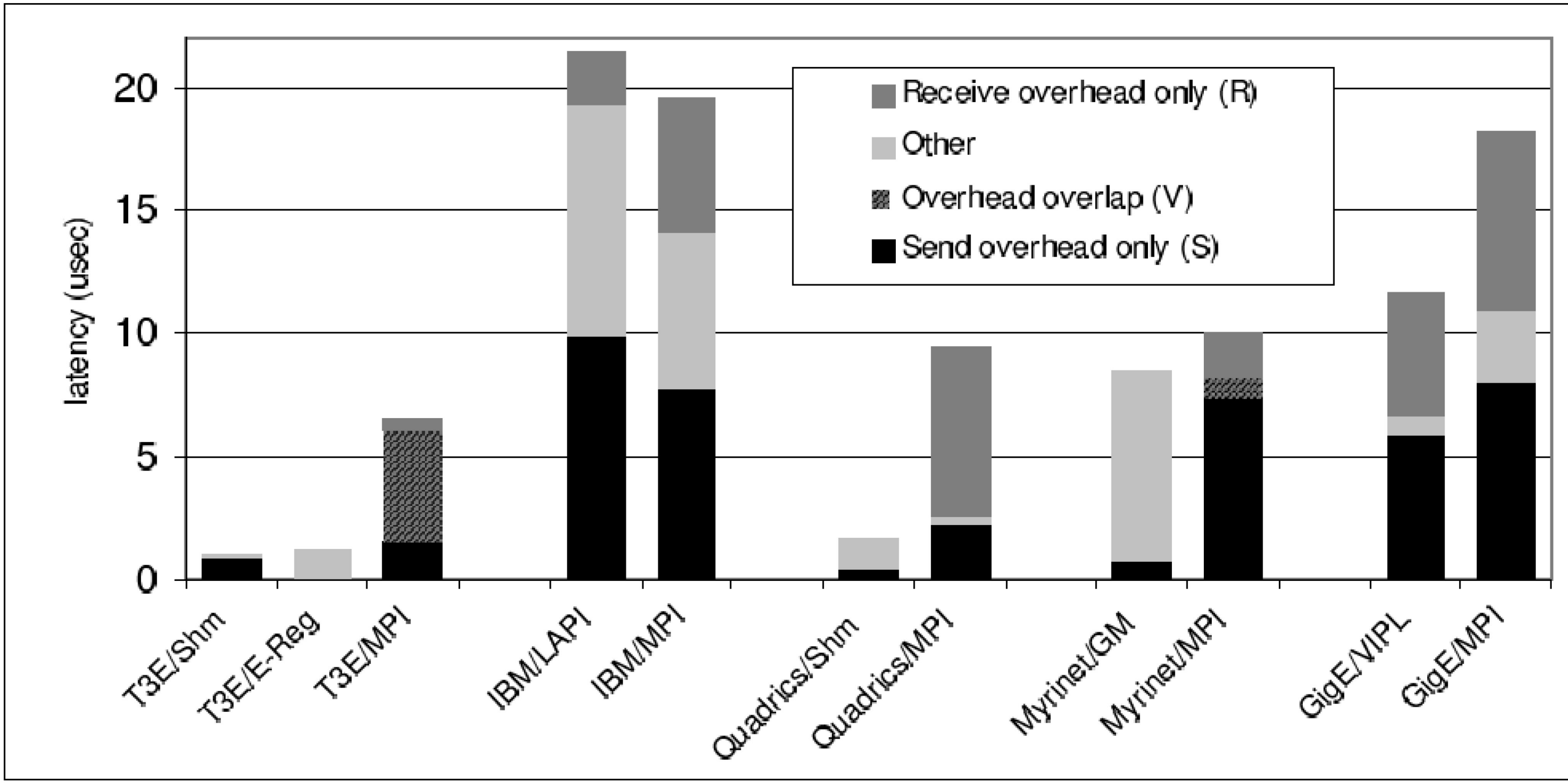
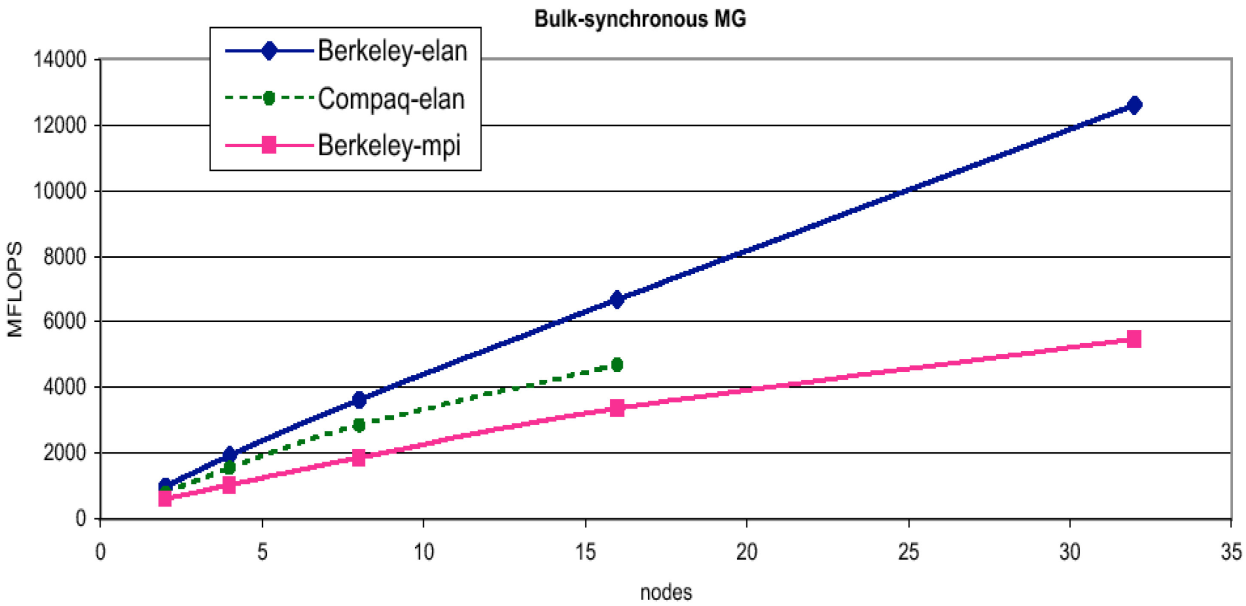


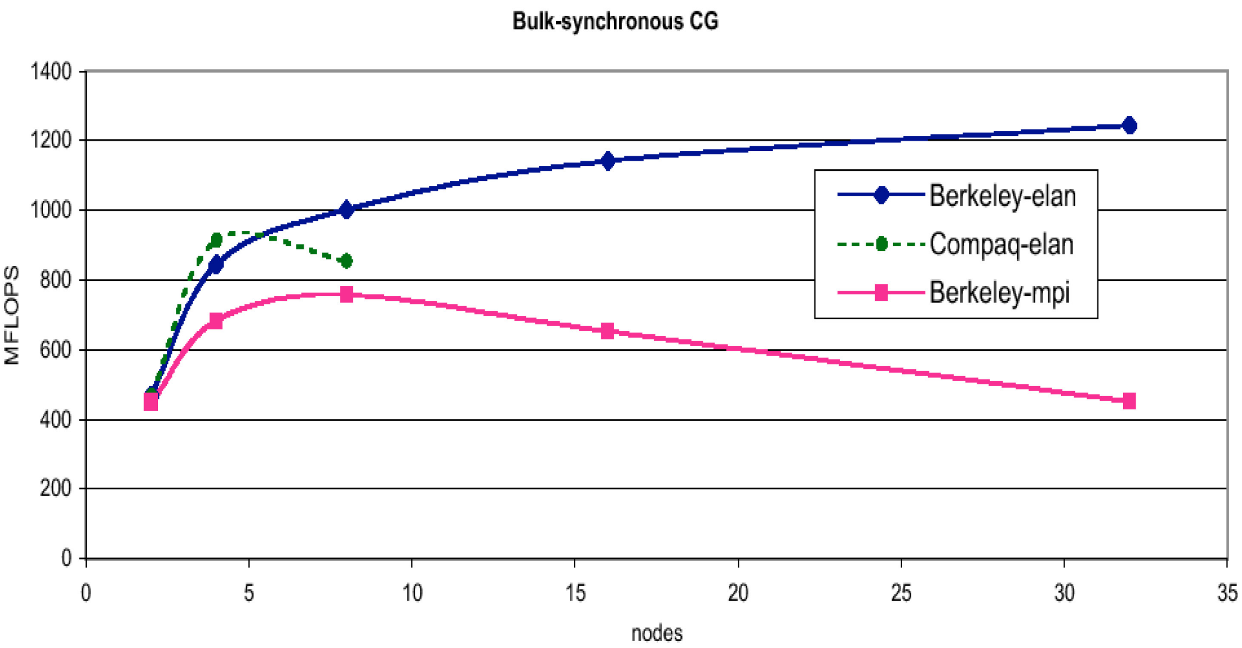

**Figure 3. Send and receive software overheads ($o_s$ and $o_r$) superimposed on end-to-end latency ($EEL$). For MPI on the T3E and Myrinet, the sum of the overheads is greater then $EEL$, and so $o_s = S + V$ and $o_r = R + V$. For the other configurations $o_s = S$ and $o_r = R$.**

# MPI 1.1 vs Vendor Network API Application Performance Comparison

- GASNet - portable, high-performance communication layer designed as a compilation target for GAS languages
  - common compilation target for both UPC and Titanium
  - reference implementation over MPI 1.1 (AMMPI-based)
  - direct implementation over many vendor network API's:
    - IBM LAPI, Quadrics Elan, Myrinet GM, Infiniband vapi, Dolphin SCI, others on the way…
    - http://www.cs.berkeley.edu/~bonachea/gasnet
- Applications: NAS parallel benchmarks (CG & MG)
  - Standard benchmarks written in UPC by GWU
  - Compiled using Berkeley UPC compiler
  - Run over identical hardware and system software
  - ONLY difference is GASNet backend: MPI 1.1 vs vendor API
  - Also used HP/Compaq UPC compiler where avail. (elan-based)

# Bulk-synchronous Apps on Quadrics AlphaServer



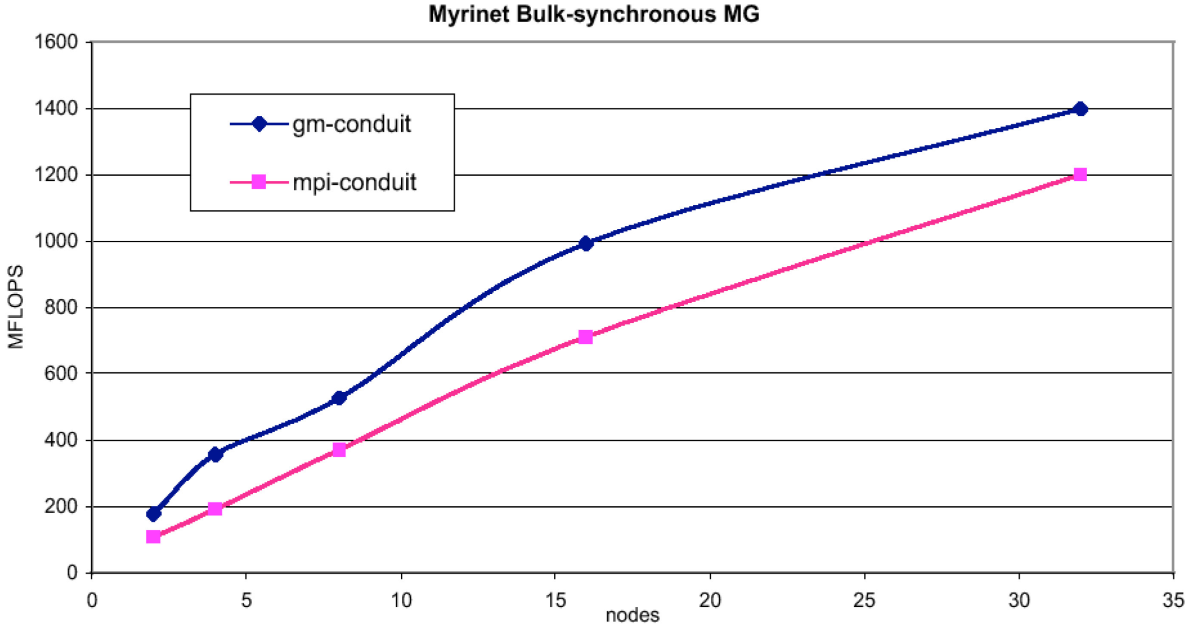

Apps on elan-based layer soundly beat apps on MPI-based layer - better performance, better scaling

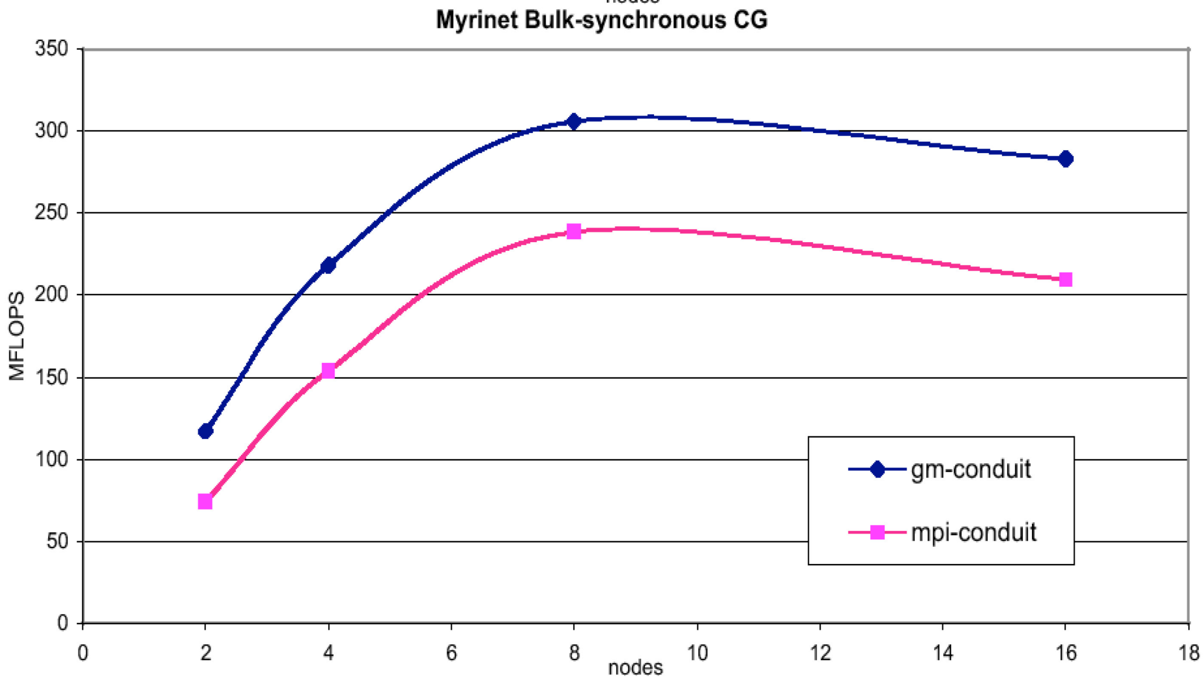
The only difference is the network API!

Results from elan-based Compaq UPC compiler also shown for comparison

Machine:
PSC Lemieux TSC

# Bulk-synchronous Apps on P3-Myrinet 2000 cluster



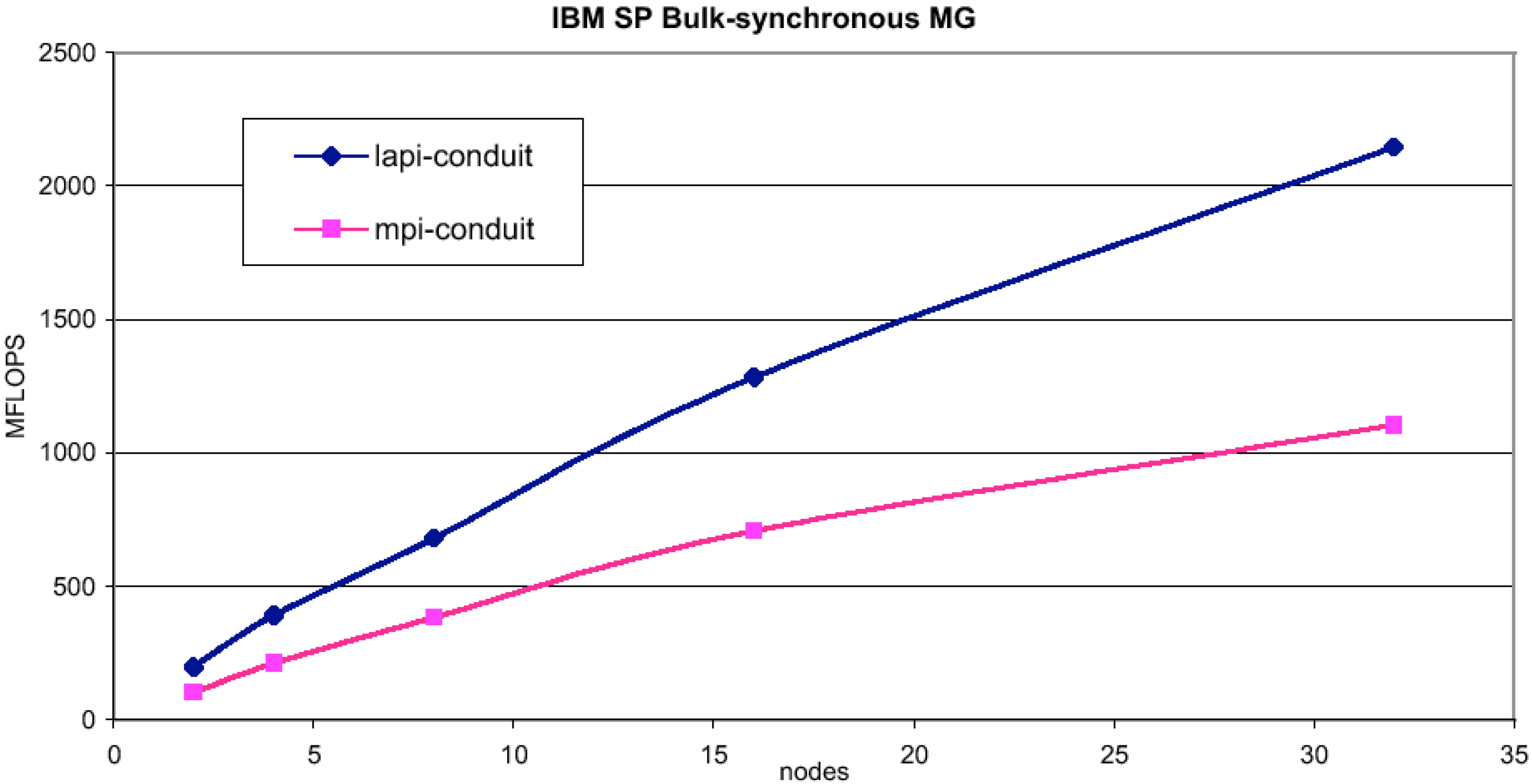




Apps on GM-based layer beat apps on MPI-based layer by ~ 20%

The only difference is the network API!

Machine:
NERSC Alvarez cluster

# NAS-MG on IBM SP-Power3

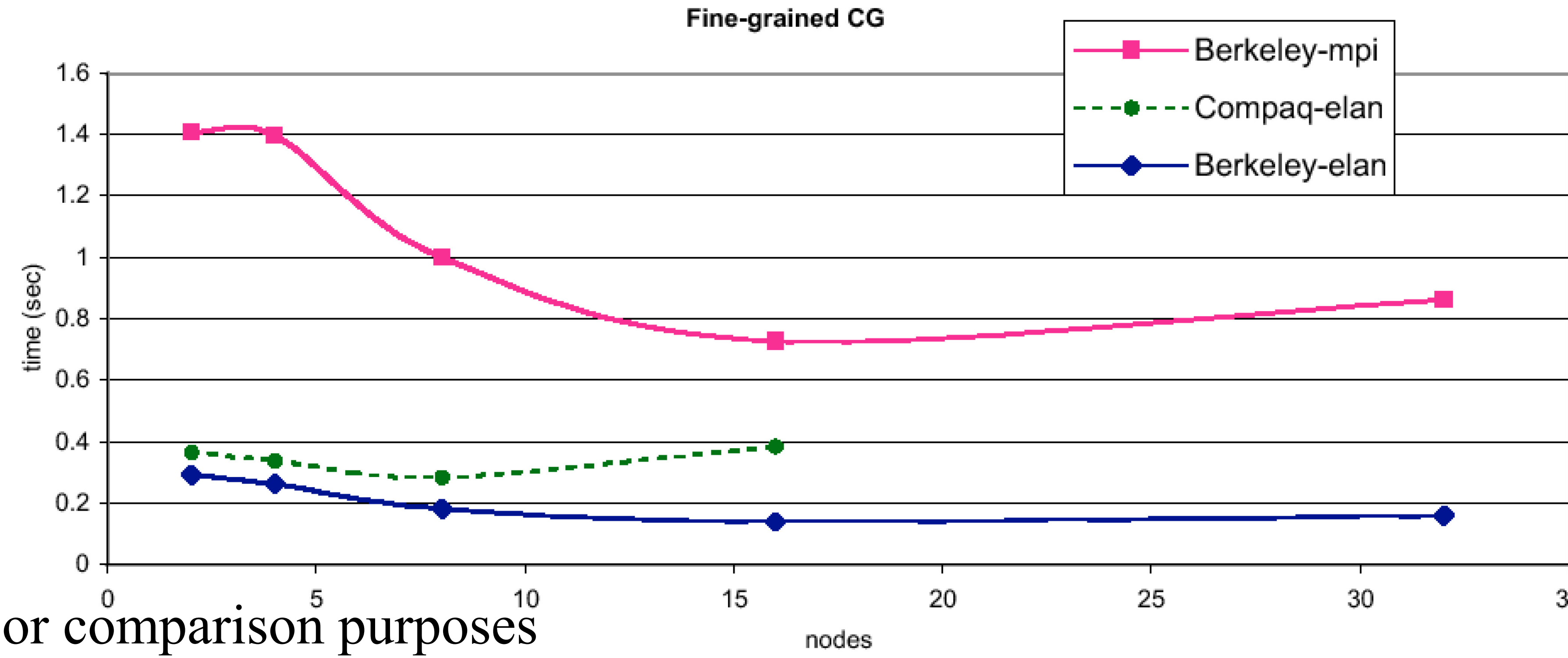



App on LAPI-based layer provides significantly better absolute performance and scaling than same app on MPI-based layer

The only difference is the network API!

Machine: NERSC seaborg SP

# Fine-grained CG on Quadrics AlphaServer




For comparison purposes

- A naïvely-written implementation of CG with fine-grained accesses (8 byte avg)
- All versions scale poorly due to naïve application algorithm
- Absolute performance: elan-based app is more than 4 times faster!
- Small messages reveal the high-latency and high-overhead of the MPI 1.1 layer
- App programmers have to work harder to get acceptable performance on MPI
- elan-based communication layer more suitable for supporting incremental application development and inherently fine-grained algorithms

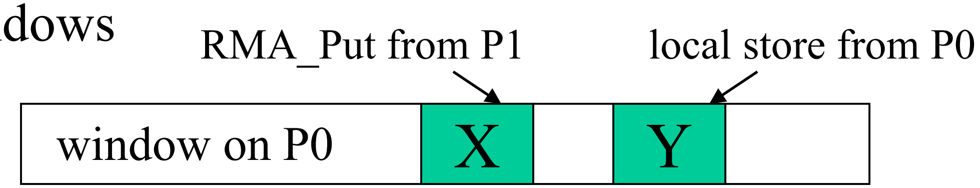
# MPI-RMA Overview

- Remote Memory Access (RMA) added in MPI 2.0
  - Provides a "one-sided" communications interface
    - MPI_Put, MPI_Get, MPI_Accumulate
  - All RMA takes place on an abstract "window"
    - window represents a memory region made available for remote access
    - Created using collective MPI_Win_create call
  - All RMA happens within a synchronization "epoch"
- Two synchronization "modes" for RMA
  - Active target - requires explicit cooperation by the target
    - Not truly one-sided, therefore not useful for our purposes
  - Passive target - only the initiator makes explicit calls
    - MPI_Win_{lock,unlock} must surround RMA calls
    - Conceptually enforce a shared (for reading) or exclusive (for updating) lock over the target window while the RMA is performed

# MPI-RMA Problematic Semantic Restrictions

- Window creation is a collective operation
  - all nodes must participate to expose new regions of shared memory
- Passive-target RMA only guaranteed to work on memory allocated using MPI_Alloc_mem
- *Erroneous* to have conflicting RMA put/get or local load/store to the same location in memory
  - must separate conflicting accesses using epochs
- RMA on a given window may only access a single node's memory during a given access epoch
- Concurrency limits on access to window's memory
  - window may not be concurrently accessed by remote RMA and local load/stores
  - even to *non-overlapping* locations in the window
  - Different windows may overlap in memory, but it's erroneous to have concurrent operations to overlapping windows

example of prohibited behavior:



# Implications of MPI-RMA Semantics for GAS

- Window creation is a collective operation
  - cannot use a window per shared object - because need non-collective, purely local, dynamic allocation of shared objects
  - the only reasonable alternative is to coalesce many shared objects into a window
- Passive-target RMA only guaranteed to work on memory allocated using MPI_Alloc_mem
  - may not work for statically-allocated data
  - no guarantees about how much memory you can get
  - likely to be restricted to a small amount of pinnable memory on some systems (useless for applications with GB's of potentially shared data per node)
- RMA on a given window may only access a single node's memory during a given access epoch
  - Need a window per node to prevent serializing RMA's destined for different nodes
  - No guarantees about how many windows can be created - likely to have scalability problems with larger jobs

# Implications of MPI-RMA Semantics for GAS

- *Erroneous* to have conflicting RMA put/get or local load/store to the same location in memory
  - MPI could generate fatal errors or have other arbitrary behavior after accesses that may be benign race conditions (where GAS has well-defined semantics)
  - Basically impossible to precisely detect all such race conditions at compile time, especially across different nodes
  - Have to conservatively assume almost everything is a potential race, wrap each put in its own exclusive epoch and all gets in a shared epoch
- Concurrency limits on access to window's memory
  - window may not be concurrently accessed by remote RMA and local load/stores, even to *non-overlapping* locations in the window
  - Drastically reduces concurrency to objects in each window
  - Especially bad if a window contains many objects
  - In GAS languages, ANY local load/store operation could potentially touch shared memory - no way to know at compile time
  - In general, would need to wrap EVERY local load/store operation with MPI_Win_{lock,unlock} to be safe. Read: super performance-killer

# Conclusions

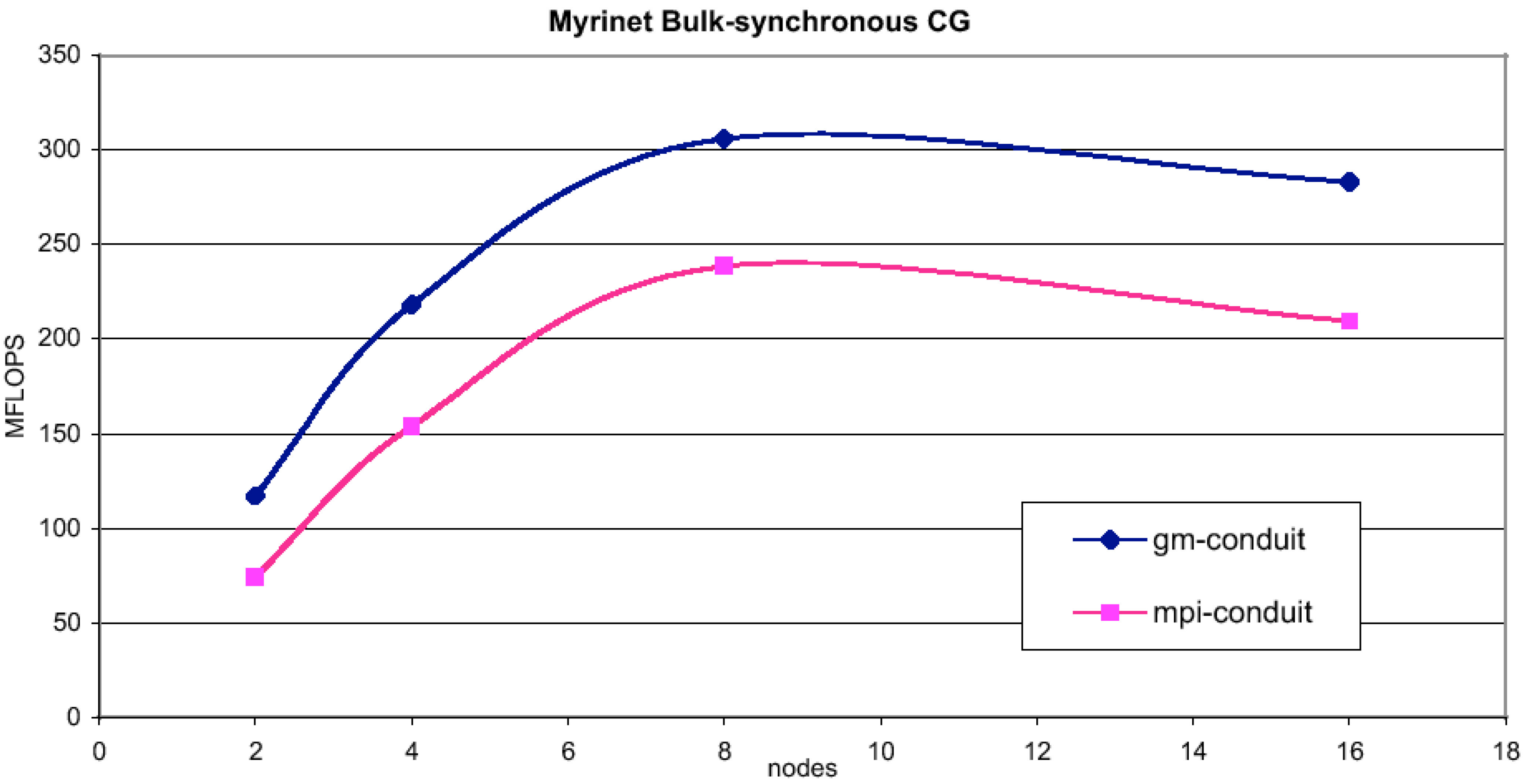
- MPI 1.1
  - Can be used to simulate one-sided messaging for GAS implementation
  - Performance hit imposed by this translation is substantial
    - cost of simulating one-sided messaging over two-sided send/recv
    - non-blocking MPI 1.1 typically has much higher latency and CPU overhead for small messages than underlying vendor layers
  - Much better performance and scaling by using proprietary vendor-provided network APIs
    - Most of which provide natural and efficient one-sided operations, which are supported in modern HPC networking hardware (e.g. RDMA, SHMEM, etc.)
    - Desperate need for a portable networking layer that exposes these hardware capabilities - our answer: GASNet

# Conclusions

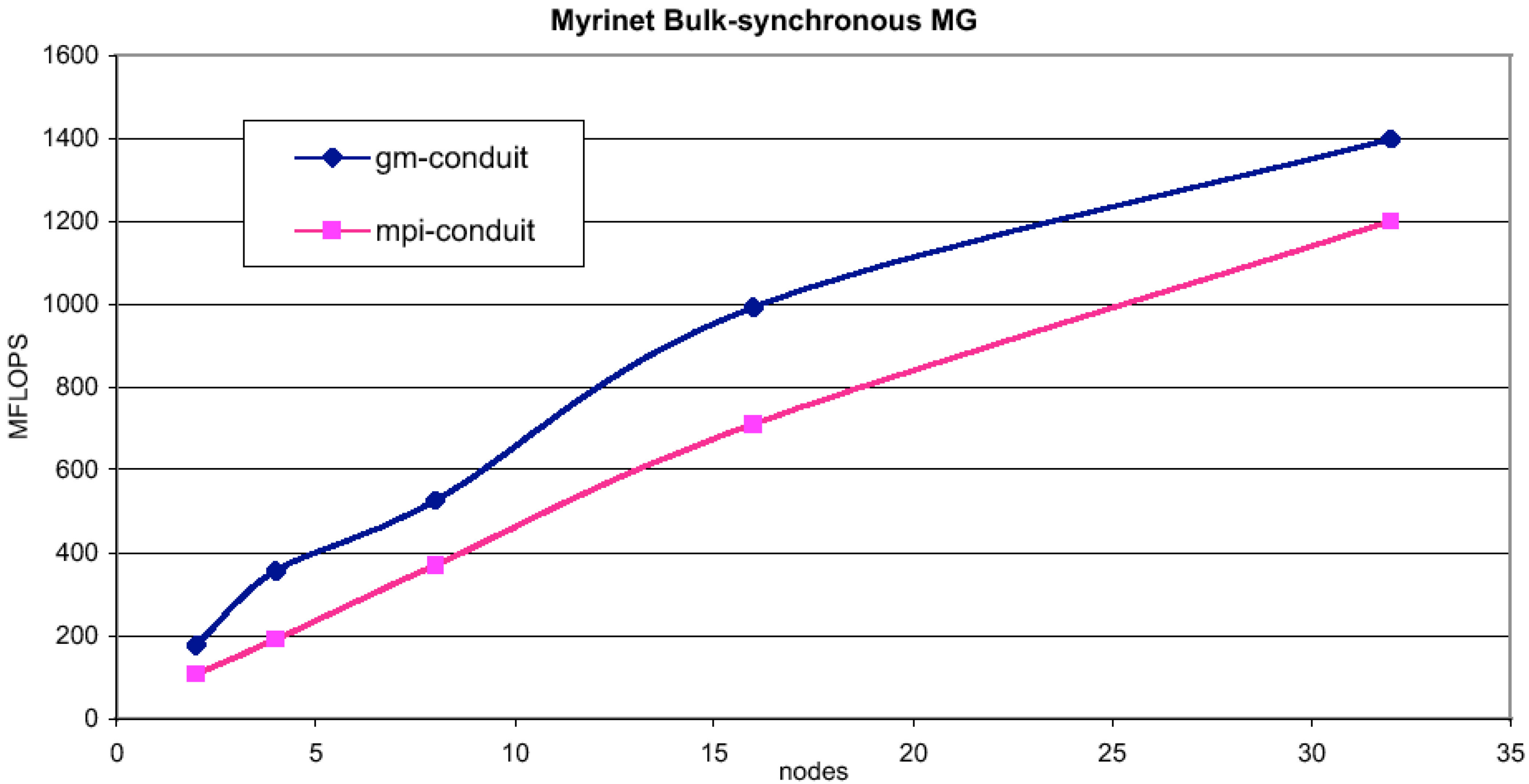
- MPI-RMA is inadequate as a compilation target
  - Current semantics are too restrictive for viable use as an efficient compilation target for GAS languages
    - Overly-strict synchronization rules for the "one-sided" API requires two-sided cooperation in practice to prevent violating the semantics
    - These usage restrictions are fundamental - parallel language compilers cannot efficiently target MPI-RMA without exposing the problematic restrictions at the source level - users unlikely to tolerate such restrictions
    - MPI-RMA is not useful as an "assembly language for parallel processing"
    - May still be useful to programmers who directly write for MPI-RMA and structure their application to obey the restrictions (conflicting evidence)
  - The problem is the MPI-RMA interface specification
    - Portability is the primary reason to consider MPI in the first place, so
    - We're only interested in the semantic guarantees provided by the API (regardless of whether some implementations provide stronger guarantees)
  - The MPI-RMA semantics should be fixed and generalized
    - Perhaps a new synchronization mode that lifts the problematic semantic restrictions? Possibly as an optional feature?
    - Adopt semantics similar to GASNet or ARMCI

# Extra Slides

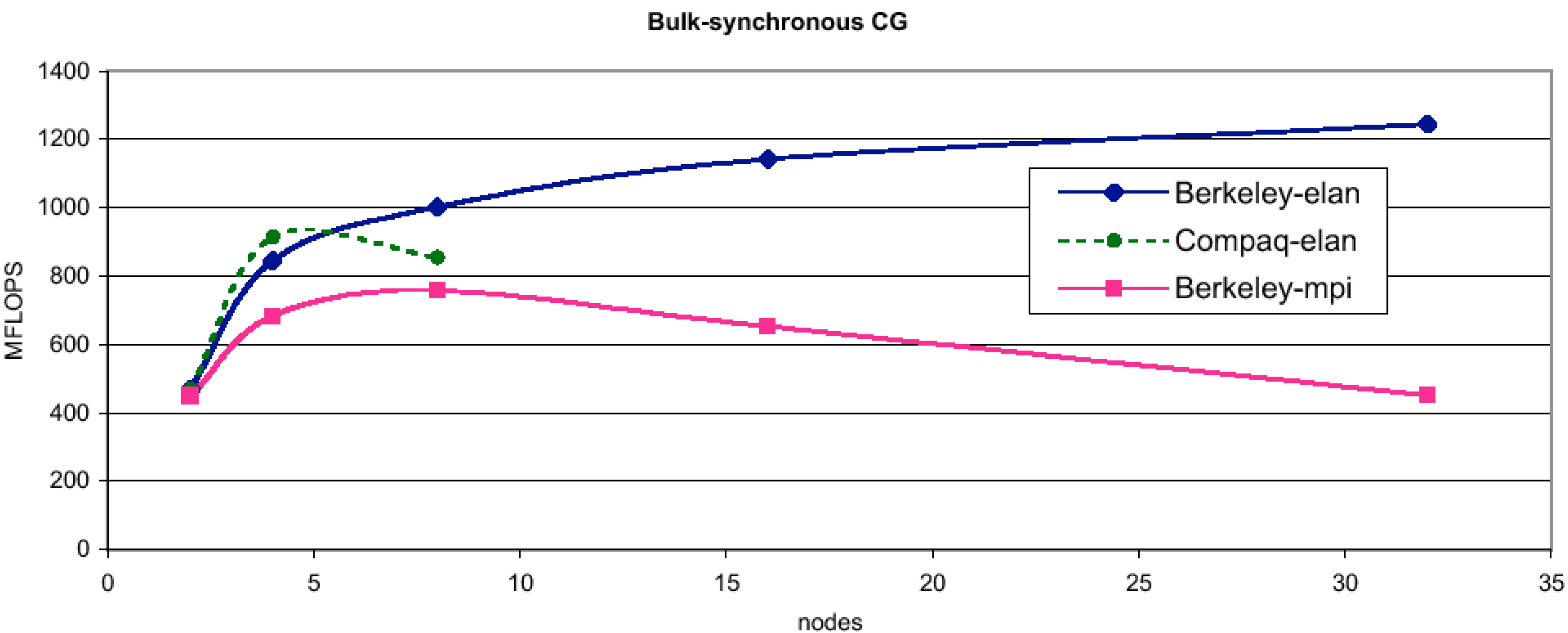
# NAS-CG on P3-Myrinet 2000



# NAS-MG on P3-Myrinet 2000



# NAS-CG on Quadrics AlphaServer



# NAS-MG on Quadrics AlphaServer