



Scalability Limitations of VIA-Based Technologies in Supporting MPI

Ron Brightwell

**Scalable Computing Systems
Sandia National Laboratories**

Arthur B. Maccabe

**Scalable Systems Lab
University of New Mexico**





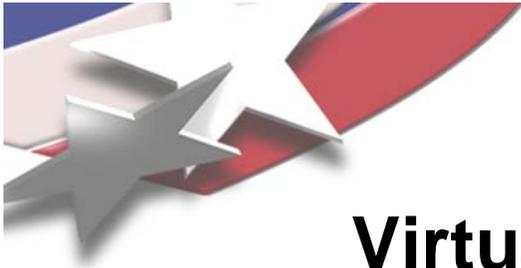
Outline

- **Background**
- **VIA**
- **MPI**
- **Cplant™**
- **Requirements**
- **Analysis**
- **Suggestions**
- **Summary**



Background

- **Zero-copy protocols**
 - Emphasis on eliminating intermediate memory-to-memory copies
 - Focus on end-to-end bandwidth for large messages
- **OS bypass**
 - Bypass the operating system for data transfers
 - Avoid protocol stack
 - Avoid interrupts
 - Decrease host processor overhead
 - Decouple the network from the host processor



Virtual Interface Architecture (VIA)

- **Published by Compaq, Intel, and Microsoft**
- **Provides user-level processes direct access to the network interface**
- **VI is a point-to-point channel between processes**
- **Connection can be reliable or unreliable**
- **Each VI has a send and receive queue**
- **Requests are given to the VI to process asynchronously**
- **Memory must be explicitly registered**
- **API supports one-sided and send/recv operations**



VIA

- **Notification of completion**
 - Examine queue to which descriptor was posted
 - Examine completion queue (select)
 - Asynchronous handlers
- **All queues are traversed in FIFO order**
- **Remote memory operations**
 - Target memory is registered
 - Origin specifies local and remote addresses
 - Operations do not consume descriptors
 - Completion is via memory inspection or additional synchronization protocol
 - Remote read operations are optional
- **1024 connections suggested minimum**



Message Passing Interface (MPI)

- **Communicator**
 - Safe message passing space for point-to-point
 - Collective operations have safe “subspace”
- **User-supplied tags**
 - Wildcards
- **Fully connected communication model**
- **Unexpected messages**
 - Usually at least a two-level protocol
 - Short protocol is eager with receive-side buffering
- **Progress Rule**



Cplant™



- **Large-scale, massively parallel computer**
- **Intended to scale to 10,000 nodes**
- **Scalability is critical**
- **Modeled after the Intel TFLOPS machine**
- **Nearly identical runtime environment**
 - **Scales to 9000+ processors**
 - **Familiar to TFLOPS and Paragon users**
- **Largest current machine is 592 nodes**
 - **#44 on the Top 500**



Phase I - Prototype (Hawaii)

- 128 Digital PWS 433a (Miata)
- 433 MHz 21164 Alpha CPU
- 2 MB L3 Cache
- 128 MB ECC SDRAM
- 24 Myrinet dual 8-port SAN switches
- 32-bit, 33 MHz LANai-4 NIC
- Two 8-port serial cards per SSS-0 for console access
- I/O - Six 9 GB disks
- Compile server - 1 DEC PWS 433a
- Integrated by SNL





Phase II - Production (Alaska)

- 400 Digital PWS 500a (Miata)
- 500 MHz Alpha 21164 CPU
- 2 MB L3 Cache
- 192 MB ECC SDRAM
- 16-port Myrinet SAN/LAN switch
- 32-bit, 33 MHz LANai-4 NIC
- 6 DEC AS1200, 12 RAID (.75 Tbyte) || file server
- 1 DEC AS4100 compile & user file server
- Integrated by Compaq

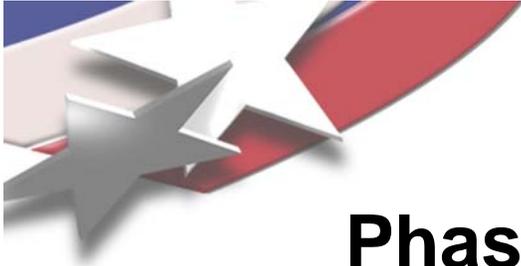




Phase III- Production (Siberia)

- 624 Compaq XP1000 (Monet)
- 500 MHz Alpha 21264 CPU
- 4 MB L3 Cache
- 256 MB ECC SDRAM
- 16-port Myrinet SAN/LAN switch
- 64-bit, 33 MHz LANai-7 NIC
- 1.73 TB disk I/O
- Integrated by Compaq and Abba Technologies



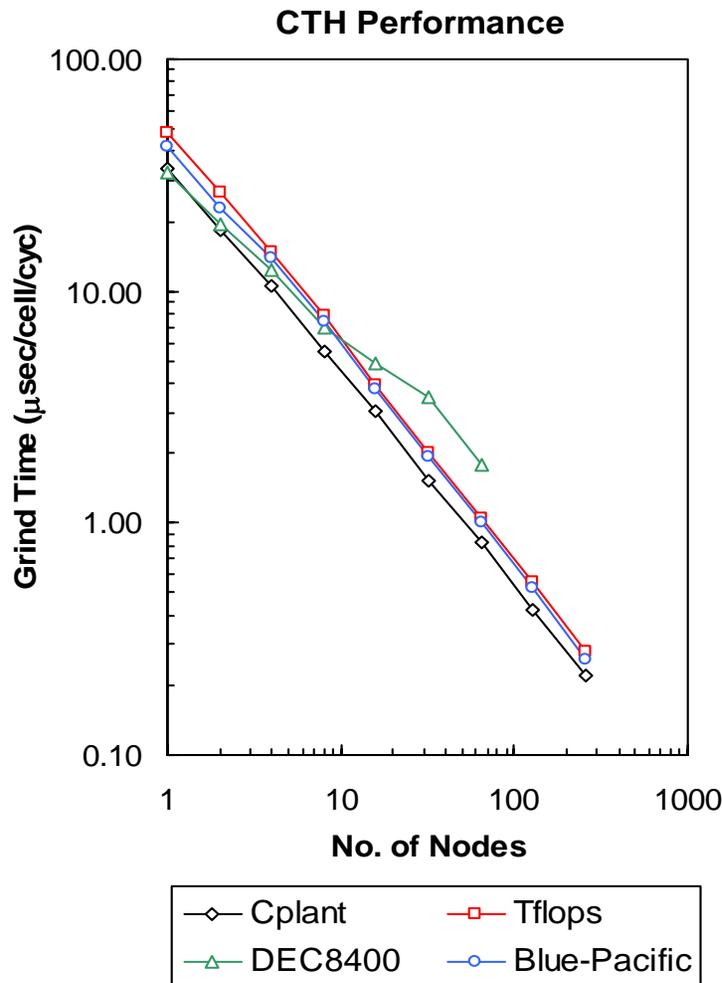


Phase IV – Development (June '00)

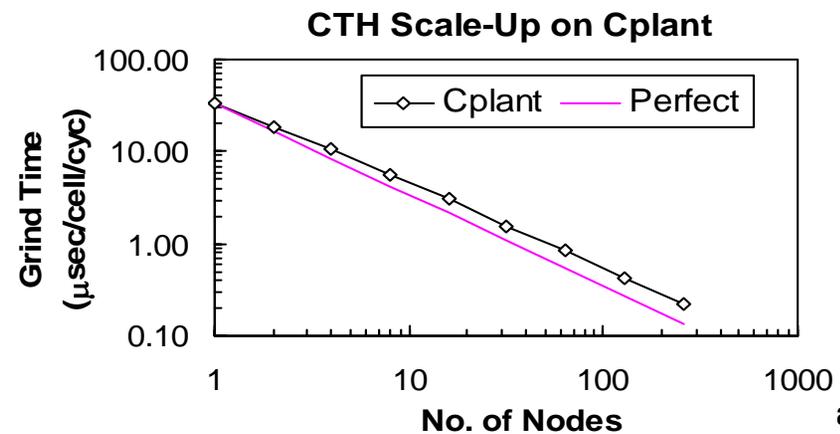
- **1024+ Compaq DS-10 (1U Slate)**
- **466 MHz 21264 CPU**
- **256 MB ECC SDRAM**
- **64-port Myrinet SAN/LAN switch**
- **64-bit 33 MHz LANai-7 NIC**
- **Red/Black switching supported**
 - **256 Black + 1024 Middle + 256 Red**



CTH Performance



# of nodes	Grind Time ($\mu\text{sec}/\text{cell}/\text{cyc}$)			
	Cplant	Tflops	DEC8400	Blue-Pacific
1	34.06	48.58	32.41	41.65
2	18.37	26.73	19.33	22.69
4	10.53	14.78	12.34	14.03
8	5.54	7.78	7.02	7.35
16	3.05	3.96	4.91	3.77
32	1.52	2.02	3.47	1.93
64	0.83	1.05	1.78	1.01
128	0.42	0.56		0.52
256	0.22	0.28		0.26





Cplant™ Runtime Environment

- **yod - Service node parallel job launcher**
- **bebopd - Compute node allocator**
- **PCT - Process control thread, compute node daemon**
- **pingd, showmesh - Compute node status tools**



Runtime Environment (cont'd)

- **Yod**
 - **Contacts compute node allocator**
 - **Launches the application into the compute partition**
 - **Redirects all application I/O (stdio, file I/O)**
 - **Makes any filesystem visible in the service partition visible to the application**
 - **Redirects any UNIX signals to compute node processes**
 - **Allows user to choose specific compute nodes**
 - **Can launch multiple (up to 5) different binaries**



Runtime Environment (cont'd)

- **PCT**
 - **Contacts bebopd to join compute partition**
 - **Forms a spanning tree with other PCT's to fan out the executable, shell environment, signals, etc.**
 - ***fork()*'s, *exec()*'s, and monitors status of child process**
 - **Cleans up a parallel job**
 - **Provides a back trace for process faults**



Runtime Environment (cont'd)

- **Bebopd**
 - **Accepts requests from PCT's to join the compute partition**
 - **Accepts requests from yod for compute nodes**
 - **Accepts requests from pingd for status of compute nodes**
 - **Allows for multiple compute partitions**



Runtime Environment (conc'd)

- **Pingd**
 - Displays list of available compute nodes
 - Displays list of compute nodes in use
 - Displays owner, elapsed time of jobs
 - Allows users to kill their jobs
 - Allows administrators to kill jobs and free up specific nodes
 - Allows administrators to remove nodes from the compute partition
- **Showmesh**
 - Massages pingd output into TFLOPS-like showmesh



Parallel I/O

- **Fyod/Sfyod**
 - Runs on nodes in the file I/O partition
 - Parallel independent file I/O
 - Each compute process opens a single file
- **Third party solution**
 - Use I/O nodes as proxies
 - Use a third party filesystem (currently SGI's CXFS)



Others

- **Support tools**
 - Debuggers
 - Performance debuggers
- **Computational steering**
 - Manipulate a running application in real-time



Cplant™ Requirements

- Target 8192 nodes
- Number of connections
 - **MPI application**
 - Fully connected
 - **PCT's**
 - Fully connected
 - Single persistent connection to allocator
 - Single connection to launcher
 - **Parallel file system**
 - 32:1 compute nodes to I/O nodes
 - **Debugger, Steering**
 - Fully connected?



Requirements (cont'd)

- **Establish connections as needed**
 - **Performance degradation**
 - Initial send/rcv operations incur connection cost
 - Initial send/rcv operations may incur connection breakdown cost
 - **Requires a “listener”**
 - Consumes CPU cycles for accepting connections
 - Consumes memory for extra thread/process
 - **Loss of determinism and predictability**
 - Same application can behave very differently
 - **Loss of fairness**
 - Wildcard receives come from established connections
 - Independent processes share connections
 - **Increases complexity**



Requirements (cont'd)

- **Time to open/close a connection**
 - **Parallel job startup should happen in seconds, not tens of minutes or hours**
 - **TFLOPS can launch 4000-node job in less than 30 seconds**
 - **Cplant™ can launch 580-node job in ~5 seconds**
 - **Assume all connections can be established in $O(n)$ and each connection takes 100 ms:**

8192 nodes x 0.1 sec = 819.2 sec = 13.7 minutes



Requirements (cont'd)

- **Resource reservation**
 - Finite number of connections available
 - Establishing connections is expensive
 - Reserving 8192 connections before attempting to establish them might be prudent
- **Unexpected message buffer space**
 - Use only what is needed
 - Use what is allocated



Requirements (conc'd)

- **Performance**
 - **Necessary but not sufficient for scalability**
 - **While raw VIA provides performance it doesn't support MPI features**
 - **Message selection**
 - **Unexpected messages**
 - **Arbitrary memory regions**



Message Selection

- **No support for message selection within a VI**
- **MPI library code is responsible for selection**
- **Host processor must be involved in all MPI operations**
- **VI per communicator**
 - **Really requires two VI's per communicator (peer and collective)**
 - **Increases VI use**
 - **Still doesn't solve tag matching**



Message Selection (cont'd)

- **Let library do matching**
 - Mandates queue management
 - Context, tag in message header
 - Uses host processor cycles
 - Defeats the intent of OS bypass



Unexpected Messages

- **Buffer must be posted for VI receive to complete**
- **Two-level protocol used in practice**
- **Short protocol**
 - Eager send, buffer at the receiver
- **Long protocol**
 - RTS/CTS or rendezvous protocol

- **Receive descriptor must always be posted**
 - Pre-post a receive for each connection



Unexpected Messages

- **Memory use**
 - Short message size is 4096 bytes
 - 8192 nodes
 - 2 outstanding messages

$$4 \text{ KB} \times 8 \text{ KB} \times 2 = 64 \text{ MB}$$



Unexpected Messages

- Latency
 - Pre-posted receive for 4096 bytes
 - If unexpected, message is copied
 - PCI bandwidth = 120 MB/s
 - Latency (added to base transmission time)

$$\begin{aligned} & (4096 \text{ bytes})(1/(120 \times 1024^2))(\text{sec/byte}) \\ & = 1/(120 \times 256)\text{sec} \\ & = 1,000,000/(120 \times 256)\text{sec} \\ & = 32.6 \mu\text{sec} \end{aligned}$$

- Still need additional user-level flow control



Arbitrary Buffers

- **MPI has no restriction on buffers**
- **VIA provides no explicit ability to discover regions that are registered**
- **Long protocol buffers will have to be registered and unregistered**
- **Ping-pong latency/bandwidth tests do not include time to register/deregister memory**
- **Possibly a limitation of the underlying OS**



Suggestions

- **Connection bundles**
 - A single call to create a group of connections
 - Addresses resource reservation and connection times
- **Optimized connection establishment protocols**
 - Does anybody care?
- **Overflow pools for unexpected messages**
 - Limit the amount of buffering needed per connection
- **Flexible tag matching in the descriptor – no longer FIFO ordering**



Summary

- **VIA has some inherent scalability limitations**
 - Number of connections
 - Time to open/close connections
 - Resource reservation
 - Unexpected messages
 - Performance
- **VIA is probably okay for small- or medium-scale clusters**
- **Eliminating the PCI bus (ala Infiniband) doesn't help most of the problems**