White Paper

An Architectural Comparison of Single-System-Image Architectures and Clusters of Workstations for Interactive Graphics Applications

Bob Kuehne, SGI

Abstract

Shared-memory computing has a rich legacy of success in computational and graphics applications. In recent years, clusters of workstations have become a credible alternative for certain computer applications. Now, with increased graphics capability on commodity machines, clusters of workstations have become an area of interest for graphics visualization as well. This paper explores some of the hardware architectural differences between these two classes of system and the implications of these differences for software applications.

1.0 Introduction

Single-system-image (SSI) computing architectures have been the mainstay of high-performance computing (HPC) for many years. Recently, clusters of workstations (COW) has become another architecture of interest in certain HPC arenas. Interactive graphics visualization software, however, has traditionally been implemented using SSI-based architectures. Recently, using COW architectures to perform interactive graphics (dubbed visualization COW or VizCOW) has become a topic of research. Interactive graphics, however, utilizes a system in different ways than traditional HPC applications. In application domain, compute, or graphics, the goal in using scalable hardware architectures is to achieve software applications scalability. This paper will explore the history, architecture, state of current technology, and software implications in a comparison of SSI and COW architectures for performing high-performance interactive graphics.

2.0 Interactive Graphics Requirements

Interactive graphics applications (IGAs) are those in which the user of the application provides feedback and sees results within some certain time period. This definition is rather vague, but necessarily so, as the limits on the time demands for different application domains varies dramatically. For example, flight simulators may need 60 Hz updates while CAE visualizations may require 20 Hz, and plant walk-through users may be thrilled with 10 Hz. These frame-rate frequencies directly imply certain time limits in which to accomplish work per frame of rendering. For the application domains just described these would be, respectively, .0167 sec and (for 60 Hz), .0333 sec and (for 30 Hz), and .1000 sec and per frame (for a 10 Hz frame rate). Obviously, certain quality constraints must be met as well, but the tradeoff will always exist between quality and performance. Said another way, in different application domains, assuming quality constraints are met, different per-frame time limits are imposed. In IGAs per-frame time limit is one of the most important constraints to which an application must adhere.

Understanding that time is critical for interactive graphics is essential to the subsequent discussions in this paper. The key issues that differentiate COW and SSI systems relate primarily to time management and its subsequent demands on performance. Specific time-related performance issues include data synchronization, bandwidth and latency constraints, and visual synchronization.

3.0 Recent History of Parallel Computing

The history of parallel computation is long and involved, but for this paper, only a few types of systems common in the last decade will be discussed. Bus-based architectures originally dominated the SSI computing space, and continue to do so for small-CPU-count (2–16 CPU) computing systems today. However, as a problem and data size increase, the buses in these architectures quickly become saturated, causing a bandwidth bottleneck. Additionally, contention for memory, exhibited as latency, becomes an issue as well, as there is only one memory subsystem on which all subsystems perform I/O. Bus-based multiprocessor systems are known as symmetric multiprocessing systems (SMP) because all CPUs have equal access to devices in the system. The scalability of bus-based systems rapidly degrades when either the bus or access to memory becomes the bottleneck.

However, in the early 1990s, research at Stanford [15] into distributed-memory hardware architectures led to systems such as the SGI® Origin® [14] server, which alleviated the resource-contention limitation of SMP systems. The solution was a point-to-point switched architecture [11] with processing elements and local memory distributed on that architecture. Access to memory can be much more efficient with memory distributed near processors. This architecture, with nodes (containing memory and processors) distributed around a system is known as nonuniform memory architecture (NUMA). NUMA is a form of distribution shared memory (DSM) but implemented with dedicated memory management hardware and software. As graphics in particular requires a lot of bandwidth, having dedicated, high-bandwidth, switched I/O near those compute elements is also key for NUMA graphics application performance.

Also during the early 1990s, as dedicated NUMA systems were being developed, the computing power of CPUs continued to increase. Commodity hardware became powerful enough for many in the scientific community to consider doing traditional compute applications on collections of commodity hardware. These clusers-of-workstations were driven primarily by

the overall system price/performance ratio. One of the more well-known efforts in this arena began in 1994 and became know-as Beowulf [5,4]. Though COW architectures offer a significant price/performance advantage, they suffer from many of the same I/O bottlenecks as traditional SMP systems. In particular, the networking hardware and software used to interconnect these systems can be a significant bottleneck.

The key point in this historical retrospective is that current high-end NUMA systems look in many ways similar to COWs. Both have an architecture that has many compute nodes, each with memory and processing elements. However, the primary difference between COWs and NUMA systems ultimately becomes that of the performance of the interconnect between and within nodes. This difference manifests itself in different software programming techniques, and, consequently, different potential uses. These differences will be discussed in more detail in section 6.1.

4.0 Parallel Computing System for Graphics

The previous section described the two types of systems compared in this paper from a computing perspective. In recent years, research has begun to appear exploring interactive graphics on COW architectures, such as recent research by Samanta [20]. However, while some research has begun to appear, analysis of the hardware and software interaction has not yet been fully explored. Further, quality issues are largely ignored, as most research has focused on mechanisms by which systems are implemented and workload distributed. Display quality issues for VizCOWs are addressed in section 5.

4.1 Application Resource Demands

SSI and COW systems used in computational applications are often stressed quite differently than for graphics applications. Compute applications designed for COWs are frequently coarsely parallel and designed to spend a lot of time computing and much less time synchronizing with other threads of execution. Hence, computational applications are designed to exercise the CPU and memory subsystem on a particular node but to minimize communication among nodes. This is a necessary consequence of the lower bandwidths and latencies available in a COW. An IGA will often have different requirements, particularly I/O demands at a node.

Graphics applications must send geometric and image data to the graphics hardware, often a complete description of the scene to be rendered. IGAs can mitigate the download bandwidth requirements by choosing data storage modes that reside in the graphics hardware cache. Data which does not fit in the graphics hardware cache must be transferred from main memory to the graphics subsystem on a per-frame basis. Data may not able to be stored in cache because it is time-varying. Examples include CAD data that is being edited or modified, CAE data that is continuously perturbed, dynamically tessellated NURBS surfaces, or other data that may be dynamically generated.

Graphics data that might be transferred per frame include geometric data, texture data, image data, etc. The memory requirements of a single vertex with position (12 bytes), normal (12 bytes), color (16 bytes), and texture coordinates (16 bytes) is 56 bytes [7]. A small CAD model with 100,000 vetices, or approximately 100,000 triangles in extremely long strips, will require nominally 5.34MB of bandwidth. A 60 Hz application drawing this geometry will therefore use 320MB/per second of bandwidth, not including any state changes, setup, etc. A small 256-pixel-square, 8-bit-per-component RGBA image or texture requires 256KB of bandwidth each frame it is downloaded. A full-sized 1280x1024 pixel, 32-bit image will require 5MB of bandwidth. Downloading or uploading full-sized images at 60 Hz will therefore use 300MB of bandwidth. Clearly, texture, image, and vertex transfer can rapidly become a significant use of the bandwidth available. The high-bandwidth demands of IGAs on commodity hardware prompted a hardware evolution from PCI to a higher-bandwidth bus dedicated to graphics. The resultant high-bandwidth data bus for commodity architectures, developed by Intel, is known as AGI [1].

5.0 Multiple Graphics Pipelines

Systems with multiple graphics pipelines are typically used for one of several reasons. One reason is to try to address larger data sets, a second is to render existing data sets faster, a third is to distribute or collaborate data among a number of users, a fourth is to render to large display formats, and a fifth is to increase visual quality. These goals are often complementary and interdependent, and a few techniques used to accomplish these ends will be explored.

A technique used for distributed or collaborative rendering involves using each pipeline, or pipe, to display a unique display to each user. One example of this is a simulation in which multiple participants share access to a common database, with each having a distinct view into that data. It's important that each user sees the same data, but not critical that the views are displayed with exact synchronization. Examples include military tank simulations and multiparticipant collaborative design review.

A multipipe technique used to address larger data sets (and a technique used to scale performance) is to divide the total data among multiple pipes. Much prior research has occurred in this area [9,23], though these techniques are now being implemented in off-the-shelf applications and hardware. Data is rendered to each of several pipelines, and then the image data is captured and composited on another pipeline for final output. These pipelines may attempt to scale the workload through a number of 2D and 3D decompositions.

Another scaling technique involves time-interleaving pipelines where each pipe renders the nth frame of a sequence of visual data, effectively multiplying the performance by the number of pipes used in the time progression. One pipe renders time=0, another renders time=1, and so on, to the last pipe, which renders time=n, where n is the number of pipes in the system. Though performance scales upward, latency also increases by the number of frames in the pipeline. Display decompositions require software synchronization (and for time-based decompositions, hardware synchronization) and large bandwidth for pixel transfers among the pipelines.

A third multipipe technique involves using graphics pipes in combination to create a large tiled display by using multiple output display devices. Examples include the CAVE™ [10] and SGI® Reality Center™ facilities[21]. In these cases, hardware display synchronization is critical or a number of artifacts will quickly destroy the illusion of a coherent display.

The above usage scenarios and scalability implementations describe a few of the possible uses for multipipe systems. In a few cases, no display synchronization is necessary. However, in time-based and multiprojector-based techniques, synchronization of the output visual systems is essential to prevent visual artifacts. Several types of synchronization are common in multipipeline graphics systems, and these will now be described. Details on these techniques can be found in more detail in a white paper by Burns [6].

5.1 Frame Buffer Synchronization

Frame buffer synchronization is necessary when it is desired to have multiple graphics pipelines redraw simultaneously. A buffer swap, the point at which a double-buffered graphics visual copies the contents of the back buffer to the front buffer, is the point at which synchronization needs to occur. Typically, this buffer swap occurs on a vertical-retrace interval, equivalent to the screen-refresh rate. The implication here is that the maximum frame rate for double-buffered application on a particular display device is the screen-refresh-rate. For visual correctness in a multipipe tiled display, all pipes must buffer swap at the same time. Several techniques of varying quality exist to allow this to occur.

5.1.1 Framelock and Genlock

Two types of buffer-swap synchronization are in common usage today: framelock and genlock. Genlock is the capability that allows multiple pipes to refresh each pixel of display synchronously. Framelock is the capability of all pipes within a system to begin redrawing at the same time. Framelock does not ensure that each pipe draws all portions of the display synchronously, but rather that each begins drawing at the same time. Either type of synchroinzation is better than no synchronization at all, but genlock synchronization is the least visually disturbing, and therefore a requirement for the highest quality displays.

5.1.2 Swap-Ready

The ability to have frame-accurate or pixel-accurate synchronized displays is important, but one remaining piece is necessary for a complete multipipe synchronization picture. This piece is the ability for multiple threads on multiple graphics pipelines to defer buffer-swap commands until all pipes have completed rendering. This is necessary because when one pipeline finishes rendering before another, it and the other pipe may not buffer swap on the same vertical retrace. This introduces a visual discontinuity that detracts from the overall quality of the display.

The ability for multiple pipes to signal their readiness to swap is known as swap-ready signaling. Swap-ready signaling may be performed in software over TCP/IP or through a serial port-based mechanism. Software mechanisms such as these are problematic because communicating signaling data with networking or serial hardware involves interaction of different hardware subsystems. This interaction can introduce additional latency into already latency-critical graphics applications. Swap-ready signaling implemented in dedicated hardware as part of the graphics pipeline provides the lowest latency mechanism, though it necessarily involves vendors implementing this feature on their graphics hardware.

6.0 VizCOW/SSI Comparison

As noted earlier, COW and SSI systems are quite similar in overall design, with major differences in software and communications architectures. These differences will now be discussed with an eye towards the graphics demands discussed in section 4.1.

6.1 Hardware Comparison

Typical VizCOW installations resemble traditional COW installations with the addition of graphics hardware. Typical COW installations are constructed from off-the-shelf PC hardware with dedicated high-speed data network hardware. Two common vendors of this hardware are Myrinet [18] and Giganet [12]. As has been previously discussed, bandwidth and latency are key in IGAs, so using dedicated data-network hardware addresses this need directly. However, this network hardware lives on the shared-PCI bus within a commodity node. This bus provides anywhere between 528MB/per second (PCI 66 MHz/64-bit) and 132MB/per second (PCI 33 MHz/32-bit) of bandwidth, depending on the variant of PCI implemented. Note that these bandwidths are theoretical maximums and presume no other resource will be contending for the shared-bus bandwidth. Simultaneous communication of multiple devices such as network controllers, SCSI disk controllers, and others will all require portions of this bandwidth.

Data-network cards are then typically attached to a switch-hub, providing point-to-point connectivity and dedicated bandwidth between nodes within the cluster. Latencies on typical COW dedicated data-network interface cards currently range between approximately 5 ms and 50 ms for round-trip communications. In contrast, latencies in dedicated NUMA hardware can be as low as 540 ns for 4-processor remote memory access and scale to as high as 945 ns for 128-processor average remote memory access [14]. For COWs, the latency due to contention by multiple devices communicating on a particular node topology increases as more nodes in the cluster are added. For example, each node can communicate directly with another node in a switched star topology with a known latency. But as additional links are traversed by data, potentially through multiple data-network switches, the latencies further increase. Choosing efficient topologies to minimize latency for COW architectures is important but is beyond the scope of this paper.

Both SSI and COW systems can contain multiple graphics cards. A variety of systems from various vendors (Hewlett-Packard, Sun, SGI, etc.) exist with multiple graphics adaptors contained within an SSI system. One specific example of a scalable general SSI system with integrated graphics is an SGI® Onyx® family machine with InfiniteReality® graphics. The InfiniteReality graphics architecture in the SGI Onyx family system implements all necessary forms of synchronization [16] described in section 5.1 within a SSI-based NUMA architecture. However, in the VizCOW scenario, commodity graphics cards are used to keep overall system cost low. Commodity cards are designed for commodity market segments and are typically designed for the largest market segment, currently the gaming community. This implies that many of the features that give these cards extreme performance numbers are targeted at markets that are often different than that which a VizCOW will address.

Another issue with commodity hardware is that these graphics cards typically implement no synchronization among and between cards. As discussed earlier, inter-card synchronization is necessary for updates to appear cohesively across multiple displays. Note, however, that commodity technology used in Samanta's tiled display system did not contain frame-synchronization technology, and they resolved this problem through exhaustive load-balancing algorithms and software.[1] In contrast, though a barrier-wait software solution is easier to implement, it suffers from latency and frame-accurate synchronization issues described in section 5.

6.2 Software Comparison

In SSI systems of any particular architecture (NUMA or SMP, for example), a single copy of the operating system runs and controls the overall system. Software running on systems of this sort can use traditional programming models. Data can be shared among multiple threads using explicit shared-memory methods, through implicit shared-data threading models or through any other technique that might be available on a traditional workstation. The operating system manages all aspects of thread migration, memory management, placement, and access. Applications can request specific processor affinity or memory placements if required. In essence, the programming of software for a multipipe SSI system is a task to which programmers can directly apply their existing skills. In a VizCOW system, numerous copies of a particular operating system are run simultaneously, one on each node. One system in the cluster acts as the process distributor and spreads threads of execution among various compute nodes in the cluster. Software running on VizCOWs must parallelize and share data through explicit shared communication mechanisms, such as MPI [7], PVM [19], or software DSM systems such as TreadMarks [2,22].

[1] The main thrust of this research was to attempt to equitably balance workload among nodes in a system. However, one consequence of this was that all pipelines finished rendering very close to each other and therefore buffer-swapped "within a few tens of microseconds of each other."

7.0 VizCOW Analysis

The previous sections explored hardware architecture, software techniques, and graphics demands. The paper now focuses on some analysis of the key problems facing VizCOW architectures and explores current application and market applicability.

7.1 Bandwidth and Latency

Interestingly, results discovered in Lenoski [15] and predating the onset of VizCOW show that having a high-performance switched network feeding SMP nodes can perform quite poorly in certain nonlocal memory-access cases. Specifically, they noted that the SMP node bus bandwidth was the limiting factor in overall system memory bandwidth when the majority of memory accesses by a node were not local to that node. This result is corroborated in a different study of clusters as performed by Cox et. al. [8] in which they compare hardware and software DSM architecture. Specifically, they noted that for low synchronization and communication demands, software and hardware DSM systems can perform comprabably; however, as these demands increase, the software DSM performance rapidly degrades. Both of these findings—that synchronization for software DSM and that node bus-bandwidth are bottlenecks—pose serious problems for distributed interactive graphics applications.

Consider, as an example, using a fast dedicated off-the-shelf interconnect in a VizCOW. Recent VizCOW research using a Myrinet network (Samanta [20]) achieved 26 ms round-trip latency and 100MB-per-second bandwidth in an 8-node cluster. Using this latency implies that, at most, 3.8×10^4 messages can be communicated each second between two nodes in the system. With an application with per-frame time constraints of 60 Hz, only about 640 messages can be communicated each frame between any two nodes in the system. Addressing bandwidth, using the maximum bandwidth capabilities of a 528MB-per-second node bus interface (the PCI 66 MHz/64-bit maximum, there is effectively only 8.8MB of communication bandwidth per frame at 60 Hz. The system assembled by Samanta had bandwidth yielding only 1.7MB per frame. Further, as communications become more complex among multiple nodes with a VizCOW, the bandwidth and latency demands will increase, effectively lowering bandwidth and latency.

Using Samantas bandwidth and latency numbers, 640 messages, and 1.7MB-per-frame at 60 Hz, it's important to calculate the number of vertices that can be distributed between nodes, per frame. Given the previous 56-byte-per-frame per-vertex result, approximately 32,000 vertices per second can be transferred, but give the 640 round-trip per-frame message limit, this further implies that these vertices must be packaged to preserve this constraint as well. The salient point is that in the management of shared data across limited bandwidth, high latency is a problem that requires explicit application awareness.

7.2 Application Software

Due to the nature of a VizCOW architecture, many application modifications are necessary to attempt to mitigate the effects of the lower bandwidth and higher latency inherent in the architecture. Work by Jiang et. al. [13] into scaling HPC applications through virtual shared-memory mechanisms finds that pure software DSM systems can perform well, but also scale poorly. They note in particular that applications with high synchronization demands (latency) and high communications demands (bandwidth) scale poorly on clusters. They further describe that the solution for these issues is a high degree of application restructuring and algorithmic rework. Work by Amza et. al. [3] has been performed to attempt to automate the management of data among nodes in a cluster. Their results show promise for certain application data access patterns, but they conclude that a purely automatic runtime approach is difficult to automate and likely requires user or compiler input to tune.

As with any COW architecture machine, load-balancing is essential to good performance. Work by Samanta explores a variety of load-balancing techniques and achieves very good balancing among nodes in the cluster. As is apparent from a variety of sources thus far in the paper, the communication issues among nodes must be carefully monitored at an application level to balance workload and data transmission, and manage latency. Samanta's results bear this out, showing that any form of load balancing among nodes in their cluster improved performance.

7.3 Cost

One of the primary motivators for using COW architecture machines, including VizCOWs, is cost. Though a direct comparison of cost will not be endeavored at this time, there are some general points that can be made about SSI or VizCOW systems. First, though individual components may appear less expensive in COW architectures, there is overhead in system maintenance, as there is now a network of individual systems, etc., to be maintained. Second, there is the application development overhead as described above, which adds to development costs for either custom VizCOW applications or ideally for commercial vendors who support VizCOWs. Lastly, there may be quality issues in using commodity hardware that keeps the cost low but at the expense of visual quality. Quantification of the cost of quality is difficult.

8.0 Summary

Though little direct research into interactive graphics application scalability on VizCOW architectures has been published, the existing literature evaluating cluster architectures and VizCOWs draws conclusions which have direct and obvious applicability to graphics applications. In particular, VizCOW architectures suffer from a number of limitations, including:

- Bandwidth constraints among nodes in the cluster: Bandwidth limitations are imposed by the bus interface on a node, the data-network adaptor speed, network speed, and the memory architecture on a node.
- Latency among nodes in the cluster: Latency performance is limited by the bus interface on a node, the data-network interface, the graphics interface, and the memory architecture on each node.
- Synchronization issues among graphics pipes in the cluster: Synchronization is limited by capabilities of commodity graphics adaptors used in nodes.
- Application restructuring: Dramatic restructuring is required for applications to scale. Restructuring is necessitated particularly with respect to data locality, parallel algorithms, synchronization, and load balancing.

There is much research yet to be performed to improve the state of VizCOW architectures. VizCOWs show promise for certain problem domains in which the effects of high latencies and low bandwidth can be mitigated and coarse parallelism is evident. However, VizCOW architectures currently have limitations that make it difficult to implement general, high-performance, and scalable graphics applications on them.

Acknowledgements

Thanks to Don Burns for his excellent paper summarizing frame synchronization issues. Thanks particularly to Alan Commike for an exhaustive early review and TeX help. Finally, thanks to all of my reviewers, Janet Matsuda, Thomas True, and Herb Kuehne.

References

[1] AGP. *http://support.intel.com/support/technologies/graphics/agp/*.

[2] C. Amza, A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, February 1996

[3] C. Amza, A.L. Cox, S. Dwarkadas, L.-J. Jin, K. Rajamani, and W. Zwaenepoel. Adaptive protocols for software distributed memory. In *Proceedings of IEEE, Special Issue on Distributed Shared Memory*, pages 467-475, March 1999.

[4] Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, and Charles V. Packer. Beowulf: A parallel workstation for scientific computation. In *Proceedings, International Conference on Parallel Processing*, 1995.

[5] Beowulf. *www.beowulf.org*.

[6] Don Burns. Multichannel synchronization with loosely coupled low-cost IGs. Technical report, SGI, 2000.

[7] Keith Cok, Alan Commike, Bob Kuehne, and Thomas True. Developing efficient graphics software. *SIGGRAPH 1999 Course Notes*, 1999.

[8] A. L. Cox, S. Dwarakadas, P. Keleher, H. Lu, R. Rajamony, and W. Zwaenepoel. Software versus hardware shared-memory implementation: A case study. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 106-117, 1994.

[9] Michael Cox and Pat Hanarahan. Pixel merging for object-parallel rendering: A distributed snooping algorithm. In *Proceedings of the 1993 Symposium on Parallel Rendering*, pages 49-56, 1993.

[10] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM 35*, June 1992.

[11] Mike Galles. Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip. *Hot Interconnects*, 1996.

[12] Giganet. *www.giganet.com*

[13] Dongming Jiang, Brian O'Kelly, Xiang Yu, Sanjeev Kumar, Angelos Bilas, and Jaswinder Pal Singh. Application scaling under shared virtual memory on a cluster of SMPs. In *Proceedings of the 1999 International Conference on Supercomputing*, pages 165-174, 1999.

[14] James Laudon and Daniel Lenoski. The SGI Origin: A ccNUMA highly scalable server. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 241-251, 1997.

[15] Daniel Lenoski, James Laudon, Truman Joe, David Nakahira, Luis Stevens, Anoop Gupta, and John Hennessy. The DASH prototype: implementation and performance. In *25 Years of the International Symposia on Computer Architecture*, pages 418-429, 1998.

[16] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. InfiniteReality: A real-time graphics system. In *Proceedings of the 24th Annual Conference of Computer Graphics and Interactive Techniques*, pages 293-302, 1997.

[17] MPI. *www.mpi-forum.org*.

[18] Myrinet. *myri.com/myrinet/overview*.

[19] PVM. *www.epm.ornl.gov/pvm.home.html*

[20] Rudrajit Samanta, Jiannan Zheng, Thomas Funkhouser, Kai Li, and Jaswinder Pal Singh. Load balancing for multi-projector rendering systems. In *Proceedings 1999 Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 107-116, 1999.

[21] SGI Reality Center. *www.sgi.com/realitycenter/*.

[22] TreadMarks. *www.cs.rice.edu/~willy/TreadMarks/overview.html*.

[23] Scott Whitman. A task adaptive parallel graphics renderer. In *Proceedings of the 1993 Symposium on Parallel Rendering*, pages 27-34, 1993.

Corporate Office
1600 Amphitheatre Pkwy.
Mountain View, CA 94043
(650) 960-1980
www.sgi.com

North America 1(800) 800-7441
Latin America (52) 5267-1387
Europe (44) 118.925.75.00
Japan (81) 3.5488.1811
Asia Pacific (65) 771.0290

3340 (07/12/2002) JI4067