Interactive Flow-Field Evaluation on a Distributed-Memory Architecture

Alexander del Pino

delpino@igd.fhg.de
http://www.igd.fhg.de/~delpino
Fraunhofer Institute for Computer Graphics
Wilhelminenstr. 7, 64283 Darmstadt, Germany
Phone ++ 49-6151-155-162
Fax ++ 49-6151-155-199

Thomas Jung, Thomas Stickdorn

tj@first.gmd.de
http://www.first.gmd.de/~tj
sticki@first.gmd.de
German National Reseach Center
for Computer Science
Institute for Computer Architecture
and Software Technology (GMD-FIRST)
Berlin, Germany

Abstract

We describe techniques for parallelizing rendering applications on a distributed-memory multicomputer (MANNA) that is linked to a high-performance polygon rasterizer (VISA).

Several multicomputer nodes and rasterizer channels are interconnected by a highly configurable hierarchy of crossbars. Projected primitives are distributed from computing nodes to rasterizer channels according to their screen regions.

An application prototype for real-time visualization of particles in a flow field from the automotive industry is implemented on this hardware platform.

The system enables a user to place a particle stream interactively in a simulated flow field in order to gain insight into the characteristics of the flow field. The huge amount of data requires that both the evaluation of the flow field and the visualization of the scene be parallelized.

The software architecture of the application prototype is based on the MPSC (Modifier Presenter Sensor Controller) model. We explore how it benefits from the presented scalable hardware architecture in order to meet the requirements of a highly interactive system.

Keywords: Parallel Rendering, Computer Architecture, Scalability, Latency Hiding, Screen Subdivision, Triple Buffering, MPSC model, Parallel and Distributed Applications, Real-time systems

1. Introduction: Parallel Rendering

Applications like the prototype for interactive flow-field evaluation presented below, require *not only* enough computing power for computation of the particle attributes, *but also* the ability to rapidly render vast amounts of graphical data. To increase the performance of a given graphics system, numerous approaches have been developed in order to use more than one processing unit in parallel. On general-purpose architectures, different levels of parallelism can be exploited, ranging from parallelism in the processor itself, to multiple processors operating on shared memory, to distributed-memory systems communicating via network topologies. A wide range of rendering algorithms from surface to volume visualization, with local or global illumination, has been investigated and parallelized.

The rather simple local (or direct) illumination of surfaces via the corresponding Gouraud and Phong-shading algorithms has been integrated into dedicated hardware systems that speed up three-dimensional visualization in high-end workstations [1]. These rendering pipelines typically consist of a geometry unit tightly coupled to a rasterizing unit with associated image, depth and auxiliary buffers. As technology has advanced, these systems have been able to process more and more triangles per second, producing an increasing number of pixels. But today's applications, especially in the domain of virtual reality and visual supercomputing, still demand higher triangle and pixel rates than a single system can provide. Thus it is only natural to investigate whether and how multiple instances of geometry and rasterizing units can be combined to scale to an even more powerful system.

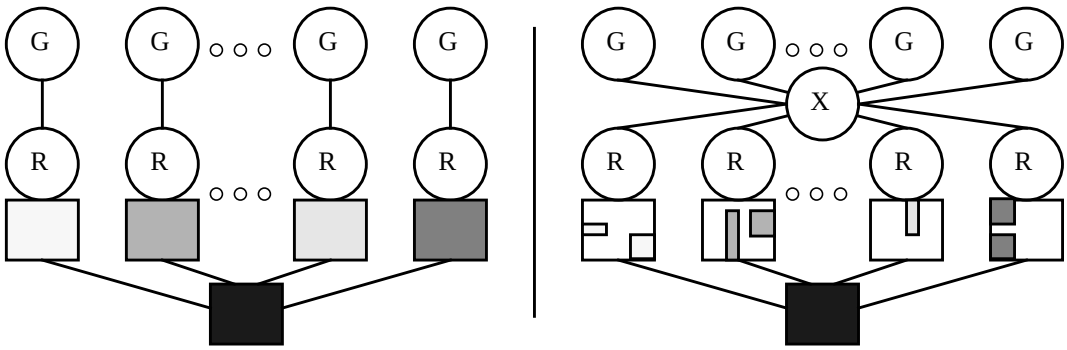

Image composition

Screen subdivision

Figure 1:

In general there are two approaches to scaling a single-pipeline to a multi-pipeline system as shown by [2]. A complex scene is arbitrarily distributed to the geometry units, and its complete image can then be either composed of full-screen images of parts of the scene (composition approach) or of partial images of the whole scene (screen partitioning/ subdivision approach).

Objects projected by a geometry unit may cover an arbitrary screen region. If the geometry unit is directly connected to one rasterizer - the typical case in today's commercial single-pipeline systems - each rasterizer must be assigned to the whole screen region, which means that only the composition approach holds when scaling to more units (see [3]). The main drawbacks of this approach are the need for complicated and expensive composition hardware restricting the range of attributes per pixel, and the restrictions on load balancing between the fixed geometry and rasterizing units.

If it is possible to redistribute the projected triangles from one geometry unit to an arbitrary rasterizer unit, the entire screen region can be divided into disjunct regions that are assigned to individual rasterizers. Combining these disjunct regions to form a single image is considerably less complex, as there is no need for per-pixel compositing operations. What is more, it permits advanced local attributes per pixel and allows more efficient load balancing between geometry and rasterizing modules. The main drawback is that there has to be an efficient redistribution mechanism from geometry units to rasterizers for this screen subdivision approach to work.

Both approaches assume some arbitrary distribution of the scene data over the geometry units in the form of a retained database. Two things have to be taken into account in this assumption. First, the computational complexity of any subset of the scene is neither constant nor completely predictable, so there has to be some kind of dynamic data redistribution to avoid load imbalance. The distributing and load balancing of a scene on a multicomputer has been investigated in several studies [4][5]. Second, any change the application wants to make to the retained database involves communication with the geometry units. Depending on the type of application - whether a retained or immediate mode of operation is used - this communication can become quite extensive and it usually has to be carried out via a single connection from a host to the graphics system.

To sum up, we have found that scaling the traditional rendering pipeline always involves object-space partitioning in the geometry stage whereas the rasterization stage can be scaled by either object- or screen-space partitioning. General-purpose multicomputers usually have a communication network that allows primitives to be redistributed from object- to screen-space partitions, so they often use the screen-subdivision approach. Dedicated rendering pipelines with their single connection fromgeometry unit to rasterizer do not offer this redistribution network and thus tend to apply the image composition approach.

2. The MANNA/VISA Architecture

We present a combined architecture consisting of the general-purpose multicomputer MANNA [6][7][8] (Massively Parallel Architecture for Numerical and Non-numerical Applications) that closely integrates application with geometry processing and dedicated high speed VISA(VISualization Accelerator) rasterizers [9][10][11] . The rasterizers are directly connected to the multicomputer's interconnection network and can thus be fed with projected objects using the screen-subdivision approach. The following sections outline the fundamental advantages of the architecture when running a graphics application.

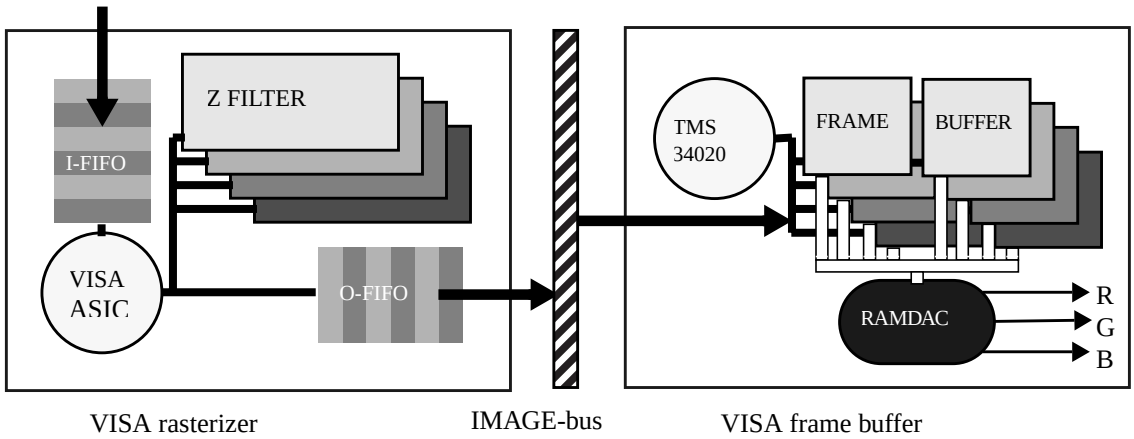

Figure 2: VISA pipeline

2.1. Scalability

2.1.1. Interconnection Network

The individual MANNA nodes are interconnected by a highly configurable hierarchy of crossbars. Each crossbar offers 16 byte-wide bidirectional connections with a transmission rate of 2 x 50 MB/s. A single rack contains 20 nodes connected via a two-crossbar backplane with up to six bidirectional intercrossbar connections (Figure 3). Multiple racks can be connected via an asynchronous ECL connection to provide theoretically unlimited scalability. The hierarchical crossbar topology has been shown to have the same favourable blocking behaviour as the hypercube, but with only one connection per node instead of log N [12].

2.1.2. Screen Subdivision

Given this fully connected network, it is only natural to apply the screen-subdivision approach for scaling beyond a single-pipeline system. Scalability is supported at three levels:

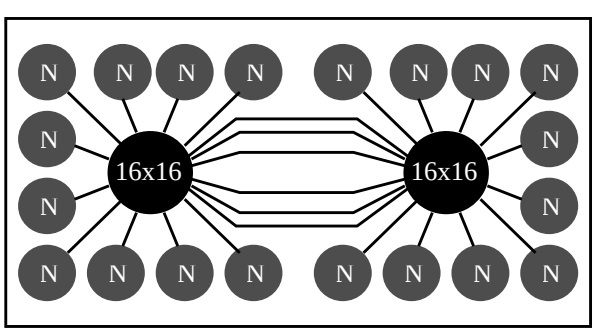

Figure 3: 20 node MANNA with two *clusters*

- the application and geometry stage
- the rasterizer stage
- the frame-buffer stage

At the first level, if more processing power is needed, one can connect several MANNA racks by a hierarchical crossbar topology. The additional network latency or contention using a hierarchy of crossbars is negligible and dominated by a constant latency of the system software [6].

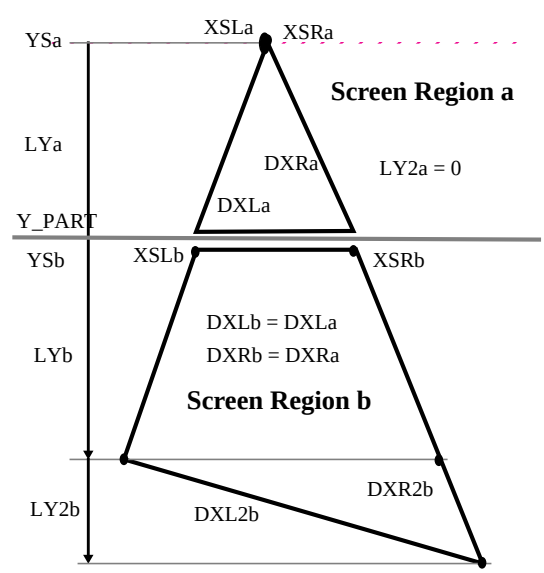

Figure 4: Subdividing triangles

Each VISA-pipeline consists of a rasterizer and a frame-

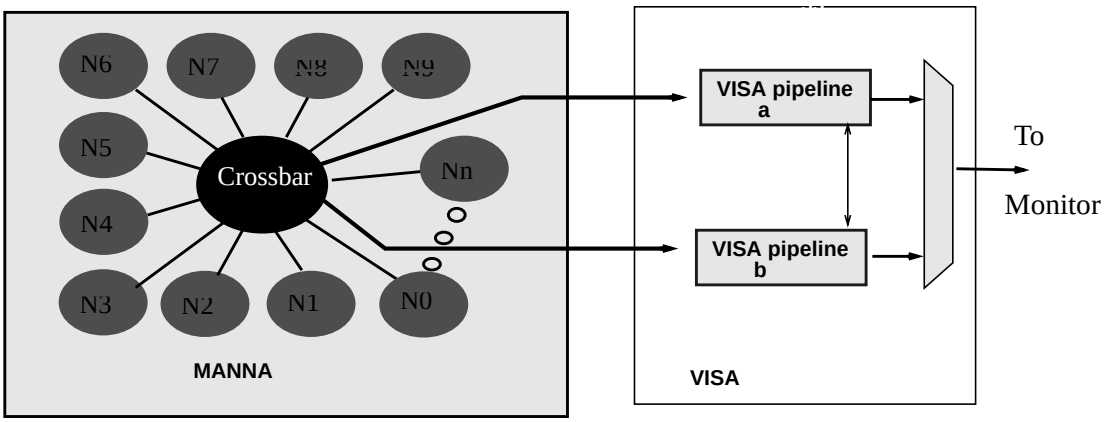

Figure 5: MANNA/VISA architecture

buffer part (Figure 2). The rasterizer offers a sustained input bandwidth of 37 MB/s, which equals roughly 550,000 to 580,000 triangles/s generated with a sustained pixel rate of 36 million per second (MP/s). The input bandwidth of the IMAGE bus and the frame buffer is 40 MP/s.

To achieve higher triangle rates, it is possible to connect up to four rasterizers that work together on one IMAGE-bus/frame-buffer system if the input bandwidth of the IMAGE bus is sufficient.

To also achieve higher pixel rates, the VISA pipelines (rasterizer and frame buffer, see Figure 2) are assigned to horizontal slices of a single frame (Figure 5). Four pipelines, for example, attain a maximum pixel rate of 160 MP/s via horizontal screen subdivision. Several channels can also operate in stereo and/or time-sequential mode to increase the frame rate. In split-screen mode, the horizontal slices of the different frame buffers are vertically synchronized to the video frequency (135 MHz) by means of PLL circuitry. The horizontal slices of each displayed frame can be adjusted dynamically by a workload-monitoring process.

To avoid the overhead of processing the same scanlines in different VISA rasterizers, triangles that cover two or more horizontal slices of the actual frame are split (see Figure 4).

The additional computing overhead for splitting one triangle into a triangle and a trapezoid can be roughly estimated as follows:

- determination of triangle covering two regions (~ 4 OPs),
- computation of new LY for region 0 (~ 1 OPs),computation of new start coordinates YS, XSL, XSR, (~ 7 OPs),
- computation of new ATTRibutes at new XSL, YS position (~ 4 OPs per attribute).

Using four attributes - Z and (R, G, B) for Gouraud shading, or Z, Nx, Ny and Nz (components of the normal vector) for Phong shading - results in approximately 28 operations overhead for every split triangle. Considering that a rasterizer can generate one pixel every 25 ns, one can easily determine the point on which subdividing the primitives becomes more expensive than drawing them twice. A good estimate is in the range of 20 to 30 overlapping scanlines. Typical image statistics of VR applications show that only a small percentage of the triangles in one frame cover 50 - 80% of the total screen area [13]. This means that the computational overhead for splitting polygons is very low compared with the speed gained by rasterizing large triangles only once.

2.2. Configurability

A further advantage is that application and geometry processing can be mapped to the general purpose nodes in a very flexible manner. Analyzing the application, the numerical effort for application and geometry processing can be estimated, resulting in a mapping from computing power to application requirements. If the communication cost can be neglected, the nodes can be divided into geometry and application processors. In other cases, integration of application and geometry processing in one node avoids remote data transmission and can reduce the time complexity of the application.

Figure 6 shows the configurations of three different applications. The left side of the figure depicts an application with low geometry-processing requirements in relation to the application-processing requirements (as e. g. in rigid body simulations where only transformation matrices are transmitted to the geometry processors

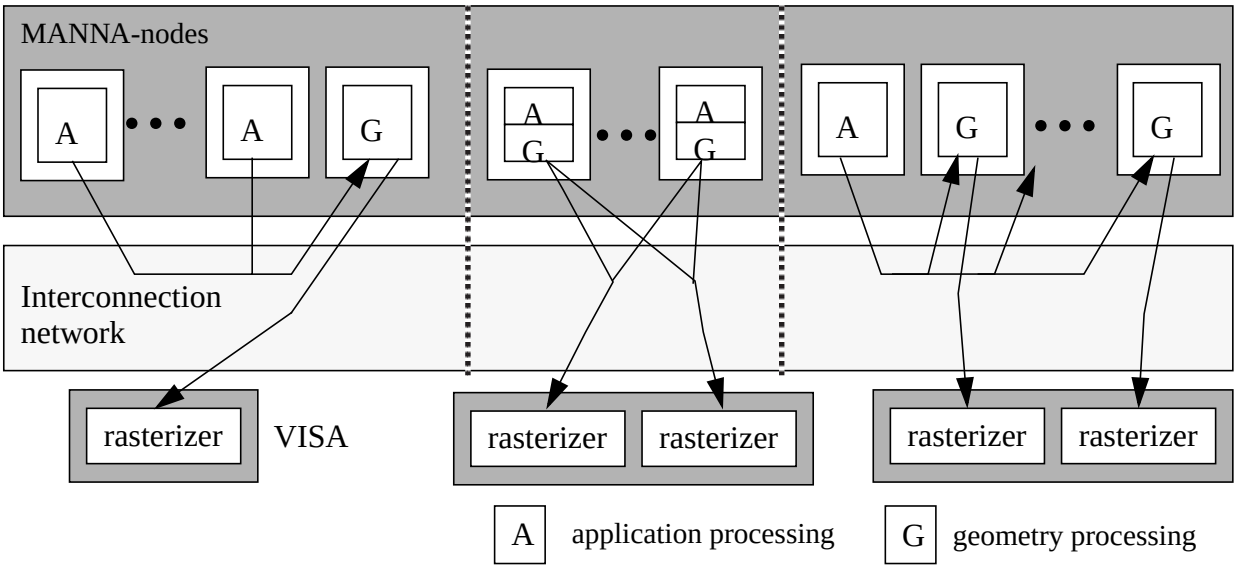

Figure 6: Different configurations

[13]).

Screen subdivision with integration of application and geometry processing is shown in the middle. The right side of the figure shows a geometry-processing-intensive application in stereo mode.

In principle, all three applications can run in parallel on a large MANNA system connected to different VISA systems. In practice, though only one VISA system with up to four rasterizers is used.

2.3. Latency Hiding

Interactive applications require immediate feedback to their actions, which means there can be little tolerance of unnecessary system latency.

2.3.1. CP/AP Mechanism

The MANNA node employs two i860XP processors sharing a fourfold-interleaved memory with 400MB/s access bandwidth (Figure 7). It also contains a bidirectional high-speed interface to the interconnection network with a bandwidth of 50 MB/s per direction. Using both processors for the application yields a sustained performance of 68 MFLOPS compared with its theoretical peak performance of 100 MFLOPS and 100 MIPS simultaneously [6].

If communication cost is relatively high in relation to the computational cost, one processor can be used for communication. The CP/AP-mechanism [14] reduces general communication latency by decoupling application and communication processes. For graphics applications, there are two areas in which this scheme can be effectively applied. One is the distribution process of

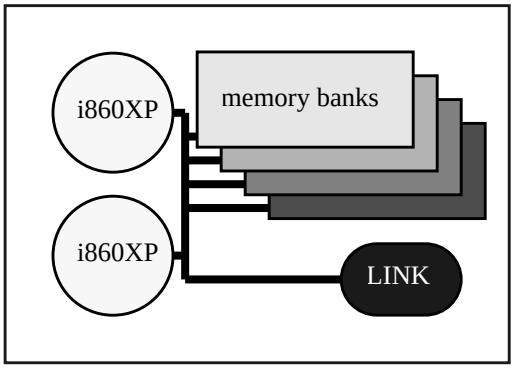

Figure 7: MANNA node

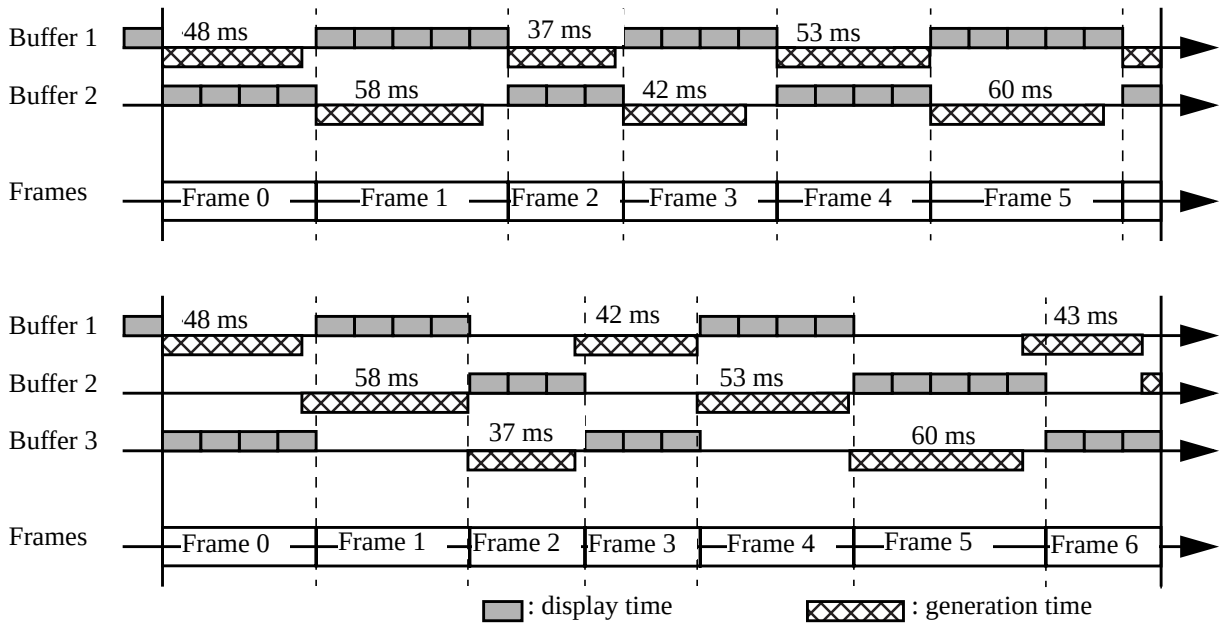

Figure 8: Triple buffering

geometry objects from one computing node to another for the purpose of load balancing. The other is the above-mentioned setup and transport of projected triangles to the appropriate rasterizers. Both processes are communication intensive and can be decoupled from the application, which can thus continue immediately.

2.3.2. Triple Buffering

Another source of latency arises in the frame-buffer system by the use of double buffering. The last frame is usually displayed from one buffer, while the current frame is generated into the second buffer. Before the generation process can go on to the next frame, it has to wait for the current display process to finish. This results in a considerable idle time of the rasterizer, and thus in a reduction of the effective frame rate. Using a third buffer decouples the generation and display processes and leads to an increased frame rate - as can be seen in Figure 8.

The performance gain is largely dependent on the display-refresh rate, the generator rate and its variation. The example in the figure assumes a display-cycle time of 13.33 ms (75 Hz) and a generation time between 37 and 60 ms (17 to 27 Hz).

In a single frame-buffer system, triple buffering can be applied by reducing the screen resolution to 1024x1024 pixels. This results in a division of the available frame memory into four equal partitions. When using two frame buffers in subdivision mode, there is enough memory to maintain a resolution of 1280x1024 pixels. Three frame buffers allow for even higher resolutions and more flexibility in screen subdivision (dynamic screen partitioning). Thus, scaling to more frame buffers not only increases the performance; the additional frame memory can be used to reduce latency as well.

.

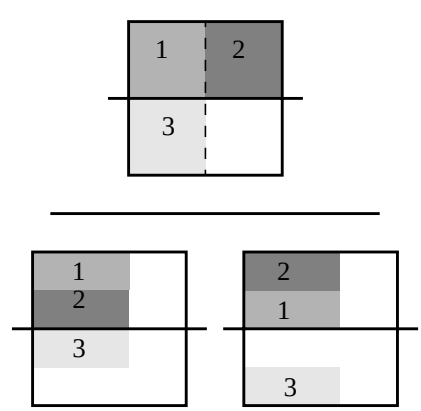

Figure 9: Partitioning frame memory for triple buffering with one or two framebuffers

3. Interactive Flow-Field Evaluation on the MANNA/VISA Architecture

To evaluate the suitability of the above hardware architecture for visual supercomputing applications, we have built an application prototype for the interactive evaluation of a flow field from the automotive industry. We already had some experience in this field from our previous implementation of such an application prototype on a SGI platform [15], but the distributed-memory architecture presented, promised us with new possibilities and challenges. In this section we address some of the problems encountered when building an application prototype, motivate our approach, and finally present the results, we obtained. When we started this work, our clear objective was, to *both* compute the particle positions *and* render the particles together with their context information on the MANNA / VISA architecture. The functionality mapping to the frontend and the MANNA / VISA system is shown in Figure 10.

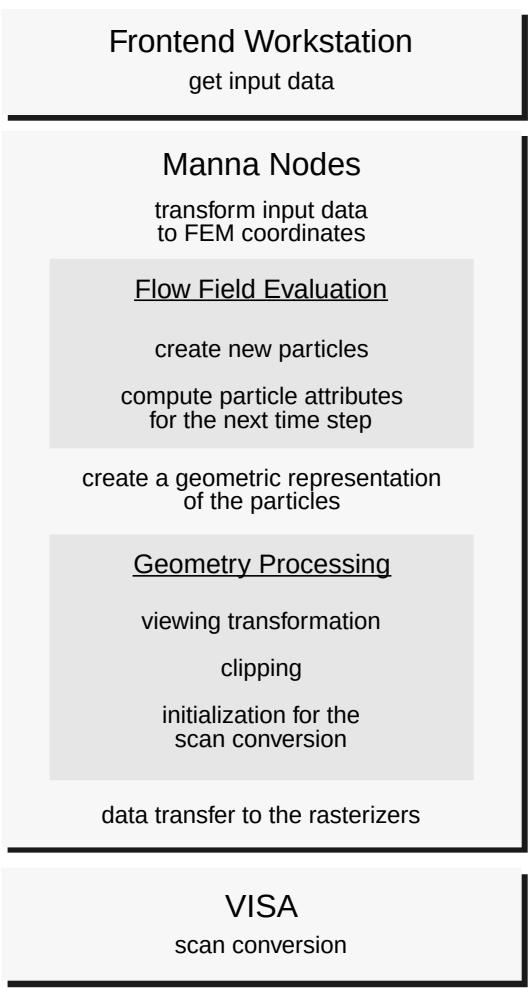

Figure 10: Functionality mapping to the hardware environment

However, many important questions could not be answered *a priori*, e. g.: How many particles is this system able to compute? How should the computing resources be partitioned between particle computation and rendering in order to satisfy the real-time constraints? The key to handling such unknown factors was to apply a highly configurable software design to this application prototype, enabling the hardware configurability features to be exploited and various configurations to be explored with respect to their efficiency. This led us to use the MPSC (Modifier Presenter Sensor Controller) model as a guideline for the internal software structure of the system. This model is sketched here only in the necessary detail, further details can be found elsewhere [16].

3.1. The MPSC Model

MPSC is a functional domain-decomposition model for partitioning distributed virtual environments into four different domains, each domain being characterized by the presence of a particular functionality, as shown in Figure 11. These domains are the modifier, the presenter, the sensor, and the controller domain. The presenter and the sensor domain can together be thought of as a frontend of the system associated with the human user; the modifier and the controller domain together as a backend.

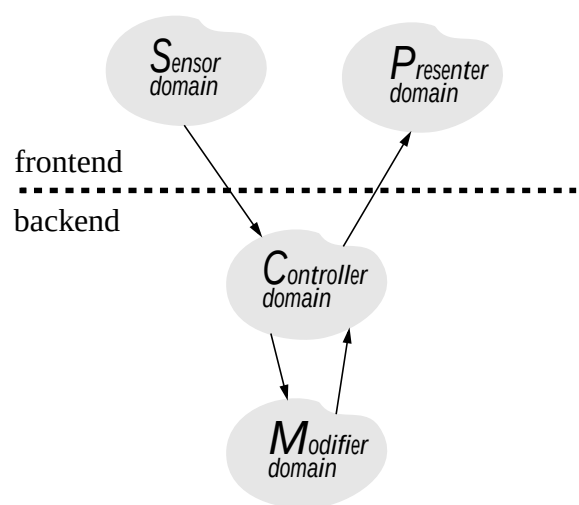

Figure 11: The MPSC model

In the present care, the task of the presenter domain is to provide objects, that are able to render the particles and the context data. The task of the sensor domain is to deliver input data from an input device operated by a human user. The task of the modifier domain is to provide objects that are able to modify the scene data, in our context the particles and their attributes.

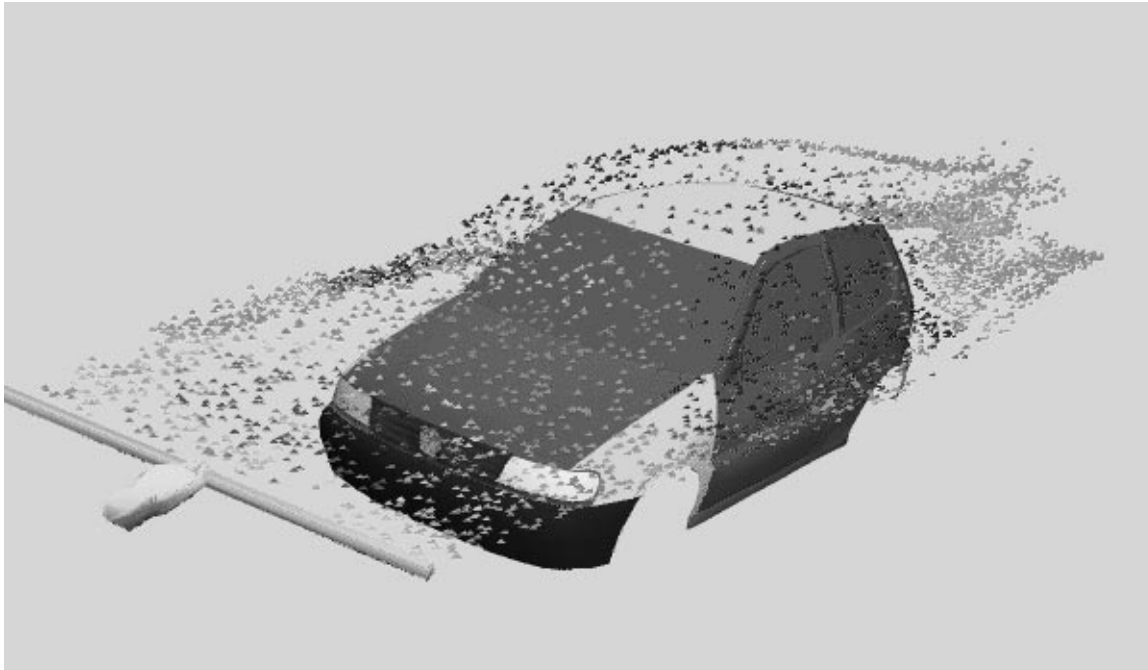

Figure 12: The application prototype, a snapshot of the running system.

Finally, the task of the controller domain is to connect the objects in the previously mentioned domains.

3.2. Flow-Field Evaluation

For the flow field evaluation, which in our case corresponds to the modifier functionality in terms of the MPSC model, we use *particle-simulator* objects. These objects contain a copy of the flow field and a pool of active particles. Evaluation of the local velocity of the particles is done in this static flow field on a regular grid with trilinear interpolation, and the integration of a particle trace is done by a fourth-order Runge-Kutta method [17].

The particle-simulator objects have the method *AddParticle*, in which the positions of new particles can be specified and the particle attributes computed for the next time step. In order to satisfy the real-time constraints and to make the computation time more predictable, the size of the particle pool is limited. If the particle pool is full, the newly specified particle starting positions have to be omitted. Particles become extinct and make room for the insertion of new particles into the pool either because they are leaving the flow field or because their lifetime exceeds an upper boundary. We use an upper-boundary default of 150 time steps, which corresponds to a lifetime of 7.5 seconds assuming 20 time steps per second.

3.3. Creating a Geometric Representation

For each particle simulator, there exists one associated *indexed triangle-list* object. These objects are not a mere data aggregation, they also have the ability to render the contained triangles using the VISA hardware [18]. The size of such an object satisfies the space requirements of a full particle pool of the associated particle-simulator object. This means that dynamic memory allocations can be avoided during runtime, thus making the runtime behaviour of the system more predictable. To create a geometrical representation, the particle positions in the FEM space are transformed into the world coordinates, and a triangle with its centre at that point is inserted into the triangle list. In this way, the local velocity is mapped to the colour of the triangle. If these triangle lists are filled with the geometrical representations of the particles, they can be treated in the same way as the triangle lists that contain the context data.

4. Results

With an internal software structure of this sort we were able to explore various mappings from the objects, that represent the functionality in a particular domain, to the nodes of the MANNA system. The peak performance

we achieved was approximately 23 frames per second with up to 5,000 particles, Figure 12 shows a snapshot of the running system. This application prototype greatly benefits from the fact that - thanks to the hardware architecture - each node at which the particles are computed can also be used for the geometry processing.

We also configured the system to do particle computing and geometry processing on separate nodes in order to simulate heterogeneous architectures. In this case, the particles have to be transferred to the renderer nodes. Here, the size of the particle pool plays a key role, because the smaller the particle-pool size, the greater the communication costs, relative to computation costs. The communication costs can be reduced by computing the particle attributes for several time steps in advance and then transferring them in one message. This means a trade-off between communication costs and increased memory requirements, because we have to keep the particles for several time steps at both the sending and receiving nodes. We call the objects that keep the particles for several time steps *future nodes;* Figure 13 shows an example in which we compute 2 resp. 4 time steps ahead. In all, 1951 particles were computed and transferred, which means that the average particle-pool size is approximately 30 particles. We observe that these curves oscillate with a period of two and four, which is due to the fact that the particle pools only have to be transferred every 2nd and 4th time step.

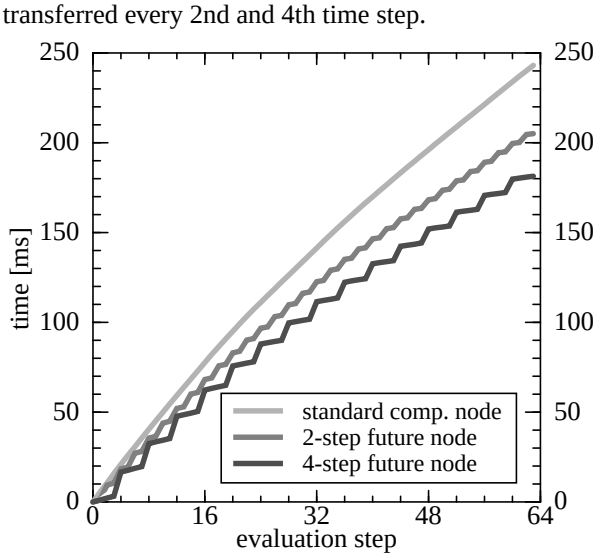

Figure 13: Measured times, when computing particles 2 and 4 time steps in advance

In Figure 14, we normalized the measured times, so that the relative time used for the particle transfer without bundling is always 1.0. Thus we observe that the curves in Figure 13 converge to approximately 0.85 in the case of 2 time steps, and approximately 0.77 in the case of 4 time steps, which reflects the time gain of the eliminated object transfers.

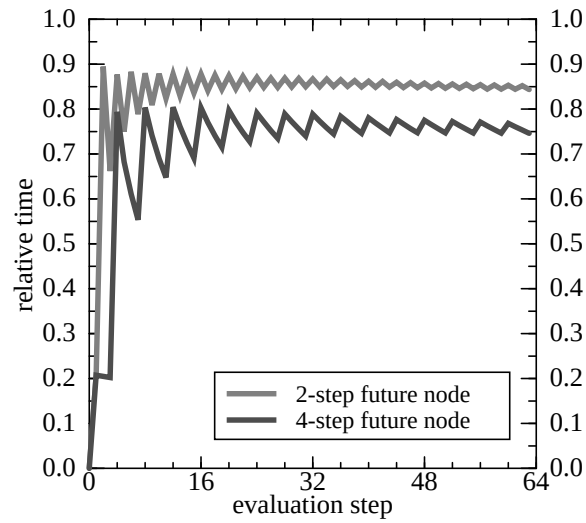

Figure 14: Normalized times of the data from Figure 13

5. Conclusion and Future Work

In this paper, we have presented an architectural alternative to the traditional graphics workstation. Its outstanding features are latency minimization and wide-range scalability. A lot of work still needs to be done to investigate load-balancing mechanisms as well as distribution algorithms for several types of applications. Our current research is concerned with the evaluation of the existing architecture and its further development. This includes the design of a next-generation computing node, advanced mechanisms like adaptive routing in the interconnection network, and a new rasterization ASIC including real-time texturing and bump mapping.

With the implementation of the presented application prototype, we have shown that such an architecture can be used for visual supercomputing. The ability to compute a vast number of particles in real time on the presented architecture allowed us to obtain insights into the characteristics of the flow field under study, that are not attainable with a similar implementation on a graphics workstation. With the appropriate software structure, an application can greatly benefit from the configuration capabilities of the hardware. Such flexibility is achieved by means of the MPSC model, which is a functional domain-decomposition model for parallel and distributed virtual environments. Our object-oriented prototype implementation of the MPSC model serves as a research platform for studying various aspects of distributed multi-user virtual environments. It enables us to use the computing power of parallel architectures like the one presented and to integrate them in a distributed

hardware environment.

Acknowledgements

Fraunhofer IGD. This work was funded by the German Federal Ministry of Research and Technology (BMBF) under Grant No. IR 409 B3 (Parallel High-Performance Computing in Computer Graphics). The data for the flow-field visualization is used by courtesy of the Volkswagen AG. The author would like to thank Prof. Dr.-Ing. Jose L. Encarnação of the Fraunhofer IGD for his support.

GMD-FIRST. We would like to thank the MANNA, PEACE and VISA teams at the GMD FIRST for their work involved in developing the system. Further we would thank Phil Bacon for polishing up the English text. This work was funded by the German Federal Ministry of Research and Technology (BMBF) under Grant No. IR 201 C (MANNA)

Literature

[1] K. Akeley, T. Jermoluk, “High-Performance Polygon Rendering”, SIGGRAPH ‘88

[2] S. Molnar, J. Eyles, J. Poulton, “PixelFlow: High-Speed Rendering Using Image Composition” SIGGRAPH ‘92

[3] S. Molnar, “Image Composition Architectures for Real-Time Image Generation”, PhD dissertation, UNC Computer Science Technical Report TR91-046, 1991

[4] D. Ellsworth, H. Good, B. Tebbs: “Distributing Display Lists on a Multicomputer”, Siggraph Proceedings 1990, pp. 147-154

[5] D. Ellsworth, “A New Algorithm for Interactive Graphics on Multicomputers”, IEEE Computer Graphics & Applications, July ‘94

[6] W.K. Giloi, “From Suprenum to MANNA und META - Parallel Computer Development at GMD-FIRST”, Proceedings Mannheim Supercomputing Seminar ‘94, Sauer Verlag

[7] W.K. Giloi, “Towards the Next Generation Parallel Computers: MANNA und META”, Proceedings ZEUS ‘95, Linkoeping, Sweden, May 1995

[8] W.K. Giloi, U. Brüning, W. Schroeder-Preikschat, “MANNA: Prototype of a Distributed Memory Architecture With Maximized Sustained Performance”, Proceedings Euromicro PDP ‘96 Workshop, IEEE-CS Press, 1996

[9] W.K. Giloi, D. Jackèl, H. Rüsseler, “Setting New Standards for Highly Realistic Real-Time Rendering”, Proceedings Graphicon 95, St. Petersburg, Russia, July, 1995

[10] D. Jackèl: “Grafik-Computer”, Springer Verlag, 1992

[11] D. Jackèl, H. Rüsseler, “A Real Time Rendering System with Normal Vector Shading”, 9th Eurographics Workshop on Graphics Hardware, Oslo, Norway, 1994, pp 48-57

[12] S. Montenegro, “Kommunikationsstrukturen für verteilte Rechnersysteme”, Dissertation, Technische Universität Berlin, FB Informatik 1989

[13] T. Jung, “An Algorithm with Logarithmic Time Complexity for Interactive Simulation of Hierarchically Articulated Bodies Using A Distributed-Memory Architecture”, Journal of Real Time Motion Analysis, to be published in 1996

[14] U. Brüning, W.K. Giloi, W. Schroeder-Preikschat, “Latency Hiding in Message-Passing Architectures”, Proceedings IPPS ‘94, IEEE-CS Press, 1994

[15] F. Dai, W. Felger, T. Frühauf, M. Göbel, D. Reiners, G. Zachmann, “Virtual Prototyping Examples for Automotive Industries”, Virtual Reality World, Stuttgart, Germany, February 1996

[16] A. del Pino, “MPSC - A Model of Distributed Virtual Environments”, 3rd Eurographics Workshop on Virtual Environments, Monte Carlo, Monaco, February 19-20, 1996

[17] T. Frühauf, “Interactive Visualization of Vector Data in Unstructured Volumes”, Computers & Graphics, Volume 18, No. 1, 1994

[18] A. del Pino, “A Classification Scheme for Rendering Algorithms on Parallel Computers” in: M. Chen, P. Townsend and J. A. Vince: “High Performance Computing for Computer Graphics and Visualisation”, Springer-Verlag London, 1996