ASICs for a High Performance Multi Processor System for Photo-realistic Image Synthesis

Peter De Vijt, Luc Claesen, Hugo De Man

ABSTRACT
A number of ASIC architectures are presented to build a system for fast photo-realistic rendering of complex images. Both ray tracing and radiosity algorithms can be used. The system consists of a number of custom and general purpose processors that communicate through a serial interface. The scene database is split into three disjoint data sets. Rays are passed between processors. The load is dynamically balanced by means of a load balancing processor. Fine grain and coarse grain parallelism are exploited.

1.1 Introduction

Although computer systems become more powerful, realistic image synthesis still takes a large time to compute. A number of techniques can be used to speed up the image synthesis. (classical techniques). If all these techniques still don't result in the requested performance, some parts of the algorithm can be executed on special purpose hardware accelerators.

In multiprocessor systems, the communication between the different processors and the partitioning of the data are critical issues. Performance degrades as the system is scaled upwards. A number of algorithms have been proposed to alleviate the communication and load balancing problem [2, 3, 4, 5, 8, 10]. These solutions however restrict the speed up techniques that can be used.

In this paper a number of solutions are presented for building a system containing hardware accelerators. First the System design criteria are set up. Secondly the difference between hardware and software parallelism is investigated. Some solutions to the main performance bottlenecks are presented. Next a number of Application Specific Integrated Circuit (ASIC) architectures are presented that fulfill the design criteria and some system implementations are given. A number of system architectures ranging from small size to large size are discussed. Finally some conclusions are drawn.

1.2 System Design Criteria

To be able to derive the specifications for the ASICs that will be used in a number of system configurations, a number of design criteria for a general system were set up:

- The system should be flexible, modular and expandable. Reuse of the different ASICs should be possible. It should be possible to replace some of the ASICs by off the shelf components. The interface of the ASICs should therefore be plug in compatible with commercial systems.

- The shading model and primitives should not be fixed or limited. It should be possible to use a number of different algorithms (ray tracing and radiosity) to generate the images.

- The total amount of data should be as low as possible. Replication of data in the local memory of all the processors should therefore be avoided.
- The distribution of the data should not pose too many restrictions on the distribution of the workload over the different processors. The load balancing should be determined by the load of the processors and should not be restricted by the availability of the data.
- The system should be scalable. Communication between the different processors should be kept to a minimum. Congestion of networks is to be avoided.
- The overhead for multiprocessing should be minimal.
- The use of conventional techniques for reducing intersection calculations should not be restricted. eg. the choice of the hierarchical data structure of the scene should be open and not fixed by a some load balancing method, data availability or system structure.

1.3 Software vs Hardware parallelism

Although a number of good solutions exist for image generation on general purpose multiprocessor systems, none of these can be used for a system that consists of ASICs due to the different nature of the parallelism that can be exploited. The algorithms involved need to be changed for a successful implementation in silicon.

The reason ASICs can outperform general purpose processors for a given application is that they can take into account the details of the algorithm. The hardware can be fine tuned for the algorithm. This is only possible when the algorithm is not too complex and consists of rather small functions which are repeatedly executed. In this case the hardware can be optimally used. If the algorithm is too complex, the specific aspects of this algorithm can not be taken into account due to space limitations. The functionality of the ASIC must then be general enough to be able to deal with the global algorithm. If an algorithm consists of two functions A() and B(), the hardware should be able to deal with the smallest common multiple C() of the algorithm. Such an general implementation can take less specific information into account than an implementation that has separate hardware for functions A() and B(). If an algorithm is very complex the implementation in hardware will be so general that it resembles a general purpose processor and no speedup can be obtained. Since general purpose microprocessors benefit from a larger design effort (more manpower), technological advantages (smaller transistor size) and larger flexibility (programmable), the algorithm should have enough specific aspects that can be exploited in silicon, to make an ASIC implementation a good solution.

A complex algorithm should therefore be split into smaller pieces if one is to preserve the speedup due to the specific aspects of the algorithm. The different parts can then be implemented in fine tuned ASICs. Due to cost constraints, the number of ASICs will however be limited. There is a trade off between the number of ASICs and the complexity of the partial algorithms and thus the speedup.

While a general purpose multiprocessor system will generally perform the whole algorithm for a given ray on one processor (with almost no communication), in a multiprocessor ASIC system the algorithm will be split in pieces which are executed on different processors. ASICs exploit fine grain parallelism, general purpose processors coarse grain parallelism.

The system can also be built in a hierarchical way. At a first level fine grain parallelism is used. At a second level, coarse grain parallelism as in general purpose systems can be used.

Most parallel architectures for image synthesis can be classified in one of three classes: (a) no data flow between processors, (b) ray data flow, (c) object data flow. The first alternative is limited to small scenes and can not be used for ASICs due to the spilt of the algorithm into smaller entities. Communication is necessary. Which one of (b) and (c) is better depends on the overall system design. When passing rays (b), the rays may pass through several processors if some form of database partitioning is used. If the partitioning is limited, the communication is also limited. On the other hand, the amount of data that needs to be passed is more important for (c).

The split algorithm should satisfy three constraints:

- The algorithm should be split in such a way that the different parts of the algorithm make the fine tuning of the ASICs possible.
- The partial algorithms should have a maximum amount of localized data. This way these localized data can be kept in local memories.
- The amount of data exchanged between the different parts should be minimal. This keeps the communication overhead minimal.

A natural division of the algorithm, which satisfies the above constraints, will consist of three parts: shading, the candidate set construction and the primitive intersection calculation. The database for these three tasks is disjoint, only ray and intersection information needs to be passed between them.

1.4 Performance Considerations

In general the performance of a multiprocessor system is very sensible to scaling. The performance of a large system decreases rapidly as the system is scaled. To achieve an optimal performance and a scalable system, attention must be paid to a number of issues. These constraints are heavily interrelated and often contradict one another.

1.4.1 Memory Management

Most multi processor systems fall into three classes: shared memory, duplicated memory or distributed memory. Shared memory is cheap but not well scalable. Duplicated memory is expensive and well scalable. Distributed memory is cheap but not well scalable and results in high communication.

In this system a combined approach is chosen. Memory is shared between processors as long as this does not prevent further scaling. If additional scaling is needed, the memory is distributed and further duplicated when needed.

The database is divided into disjoint sets corresponding to the different sub tasks. This reduces memory contention for processors that execute a different task (eg. construction of the candidate sets and the intersection calculation) A number of processors share a common memory. Data are loaded into the local caches of each processor under control of a MMU. Since image generation represents a great deal of coherence, this approach is useful.

If a large number of processors is used, the cache hit ratio will eventually drop. This problem can be solved by dividing the processors into clusters and divide the data into

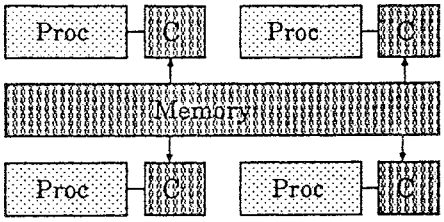

FIGURE 1.1. The processors share a common memory through caches.

different sets. This way the coherence in the partial set will rise again. A drawback of this approach is that the load balancing has more constraints and becomes thus less effective.

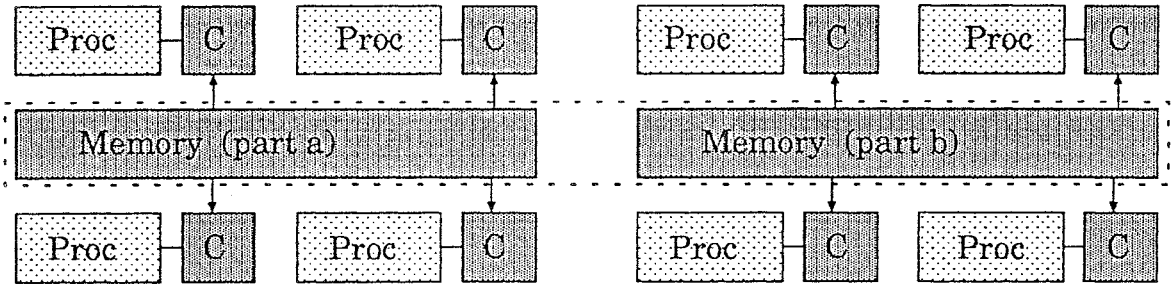

FIGURE 1.2. A cluster with partitioned data sets.

If still more processors are used, super clusters can be formed with duplicated memory.

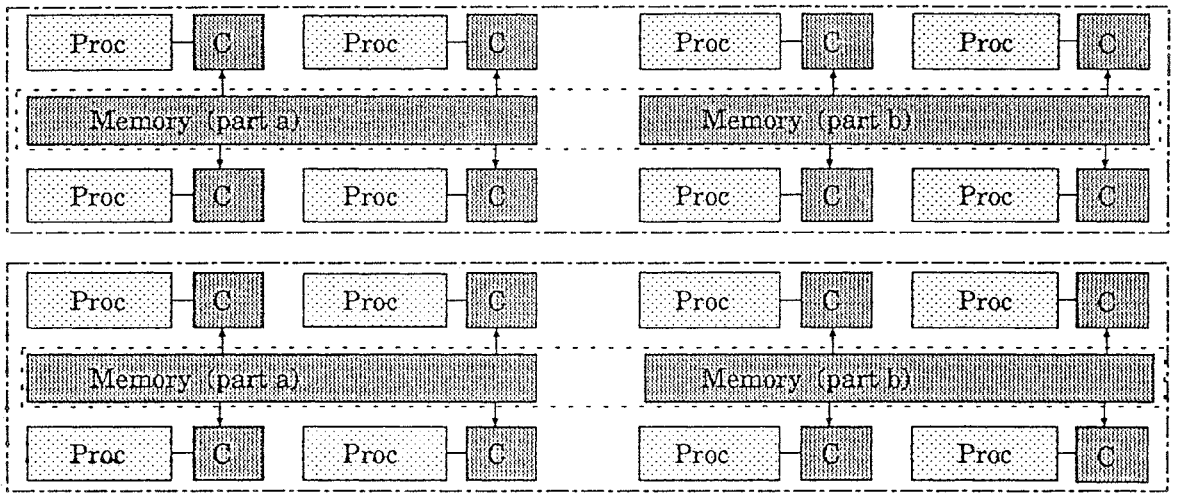

FIGURE 1.3. A super cluster consisting of two clusters.

1.4.2 The Communication Network

In a multi processor environment there is need for communication. A good communication network should be provided to manage the communication problems that usually arise in such an environment.

As a consequence of the split of the algorithm into different parts, communication between the ASICs is necessary and can not be avoided. Therefore the algorithm should

be split in such a way that this communication overhead is minimal. This communication can only be further reduced by use of higher level calls (eg. intersect an object once with a beam of rays instead of multiple single ray calls) or by keeping static data local to each processor.

Since ASICs exploit fine grain parallelism, there will be more communication than in a general purpose system exploiting coarse grain parallelism. The need for a good communication network is therefore very high.

Communication is kept local. No "hopping stones" are allowed and a point to point network is provided. In view of the system design criteria asynchronous bidirectional full duplex transputer links are used.

1.4.3 Load Balancing

Load balancing in a multiprocessors system is very critical. If the load is not spread evenly over the different processors, the slowest processor will determine the speed and the speedup will be less than linear. The system will only be moderately scalable.

In an ASIC environment the load balancing becomes even more stringent since the ratio execution time / communication time will be smaller than in a general purpose multiprocessor system. If the scheduling of the operation on one of the available resources is not done with minimal delay, the communication time will determine the total execution time. Even if these operations can be pipelined, the time spent waiting for new data will determine the overall execution time.

In an ASIC environment also fine grain parallelism is used. This means that the speed of the different processors must be matched to obtain optimal performance. Since the speed of some processors is unknown (future processors) or not fixed (eg. the number of iterations in the intersection calculation depends on the data) some actions have to be taken. The mean processing power can be matched by using the right number of processors. Variations around this mean value can be absorbed by buffering. This will however introduce some latency.

Notice that at this level the load balancing and speed up techniques are not interrelated. In a larger system, the load balancing of the higher level can influenced by the choice of the speed up technique.

1.5 The ASIC architectures

1.5.1 The Intersection Processor

In the past there have been a number of attempts to build a intersection processor for bicubic patches [6, 7, 1, 9]. To be able to take the specific aspects of the intersection algorithm into account, different primitives need to be intersected on different intersection processors (IP). Since the interface to the custom processors is a general T800 link, less used primitives can be intersected on a general purpose transputer.

Currently an implementation for an intersection processor for Bernstein patches is being investigated into. The ASIC accepts a ray and a pointer to a patch through a serial interface. The ray is then transformed from floating point into an internal data format. The patch data (which have already been transformed during a preprocessing step) are read in through a 24 bit data bus. The transformation guarantees in most cases an equivalent accuracy of a 32 bit floating point implementation (IEEE 754). The intersection is calculated with an iterative algorithm. The number of iterations can be defined by the user and is also derived from the incoming data. The chip is able to deal with three inter-

section calculations at the same time. The internal data stack is managed with an internal memory management unit. Data are stored in an internal cache according to a scheme that optimizes a cache hit for this particular algorithm. Overwritten data are recalculated when needed. The closest intersection is reported in different formats (real world space, parameter space) and the normal is calculated when needed.

1.5.2 The Candidate Set Processor

Some preliminary studies have been done for this candidate set processor (CSP). The algorithm used will be a combination of different algorithms (space partitioning and hierarchical data structure). The influence of multi processing on this algorithm has to be further studied in depth before an implementation can be made. Especially the access of the data seems to restrict the speed, more than the work involved to calculate the next set of candidates.

The different intersections reported by the intersection processors have to be compared. If there is no intersection a new set of candidates has to be calculated. The closest intersection is then reported back to the rendering processor.

1.5.3 The Load Balancing Processor

The Load Balancing Processor (LBP) has 16 bidirectional full duplex links. With each link is a link type associated depending on the processor type that is attached to this link. There can be multiple instances of a certain type of link. There are also 4 links to Memory Management Units (MMU). The LBP has a private data space to temporarily store messages before routing.

Incoming messages contain a link type to which the message can be send. The LBP puts this message in the queue for that type of link if there is still room available. When a slave processor requests data, the queue associated with that link type is checked. If the queue is non empty, a new message is fetched from memory. A message is sent to the MMU to make sure that the appropriate data are available to the slave processor. The new address is put into the message, and the message is transmitted. Since multiple slave processors can share the same queue, the load is automatically balanced over these processors.

A first version without interface to a MMU has been designed in 2.4 μ standard cell technology. There still needs to be further investigation on how a MMU interface can be implemented as a general purpose interface (incorporating a sort of data flow mechanism) so that there are no restrictions on the number of MMUs.

1.5.4 The Memory Management Unit

The memory management unit can be straightforward. A LRU scheme will be used to fill up the pages. Page size and number of different memories can be set up. Since the MMU is to work in a multi processor system not only requests for data need to be sent but also release of data needs to be reported.

It will be possible to control different memories from one MMU. Different processors can share a common MMU (four links are provided).

1.6 System Configurations

The ASICs are designed in such a way that a number of system architectures are possible (ranging from basic configuration to a large size configuration). At each level a number

of memory configurations are possible.

1.6.1 Memory Configurations

The memory of the system can be set up in many different ways:

- memory duplication: this provides good load balancing and scaling problems are low but the cost of the system becomes very high for large scale implementations
- memory distribution: this puts constraints on load balancing since not all data are available to each processor. This approach is acceptable for small size systems but the load balancing becomes poorer with scaling
- cached memory: the data are loaded from a number of global memories into local caches under control of a MMU. The load balancing is good, but the hit ratio will determine the scaling. A number of configurations exist: multiple MMUs can be attached to each LBP, multiple LBPs can share a MMU, each MMU can control multiple memories

1.6.2 Basic Configuration

The basic configuration consists of a transputer a CSP and an IP. The data are split into three disjoint sets according to the operations involved. Each ASIC has a full set of memory.

1.6.3 Medium Size Configuration

This system consists of a set of clusters of processors interconnected by LBPs. At a first level general purpose processors generate rays and calculate the rendering equations. Pixel colors are reported over a common bus. Spawned rays are collected and redistributed to a second level in a LBP. This second level the candidate sets are calculated. The requests for intersection calculation are collected in a second LBP and distributed over the different IPs. The interface between each level can consist of one or more LBPs that can have common MMUs.

At each level the data can be stored in a number of memory configurations. Since the optical information is small compared to other information, it can eg be copied into different memories. The hierarchical data structure can be shared or split depending on the number of CSPs. Clusters of processors can be formed that have common access to a part of this hierarchical data structure. Splitting the memory results in less memory contention problems, but put extra constraints on the load balancing. The ASICs are however flexible enough to implement both systems.

1.6.4 Large Size Configuration

When a very large number of processors is needed, super clusters can be formed. Each super cluster has a configuration of the medium size configuration and thus a complete copy of the entire data set. If needed some LBPs and MMUs can be shared between clusters.

1.7 Conclusions

In this paper a high performance multi processor system architecture for image synthesis has been described. The constraints that influence the performance of the system have

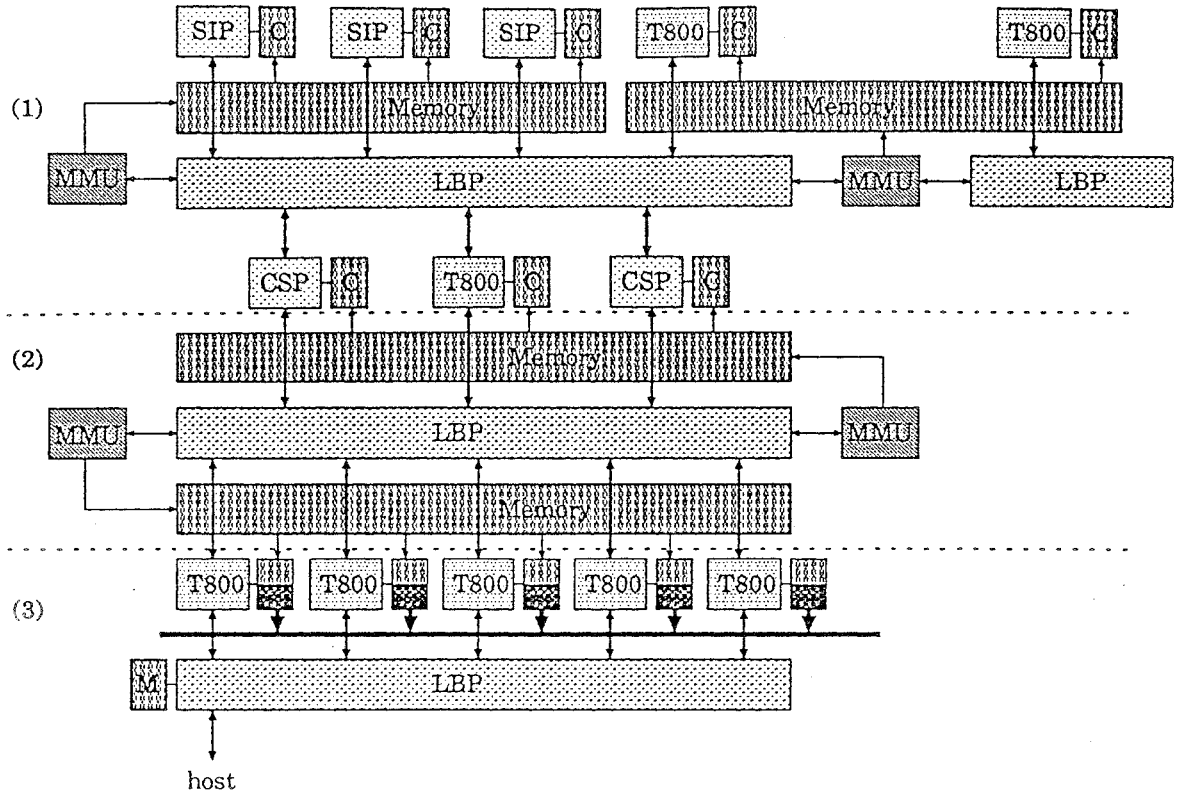

FIGURE 1.4. example of a medium size system

been investigated and a number of solutions have been proposed. A set of ASIC architectures has been presented that can be used as building blocks for a high performance system. The system is very flexible and expandable. Depending on the requested speed and available processors a full custom system can be built.

1.8 References

[1] Kadi Bouatouch, Yannick Saouter, and Jean Charles Candela. A vlsi chip for ray tracing bicubic patches. In W. Hansmann, F. R. A. Hopgood, and W. Strasser, editors, *Eurographics 89*, pages 107 - 124, 1989.

[2] John G. Cleary, Brian M. Wyvill, Graham M. Birtwistle, and Reddy Vatti. Multiprocessor ray tracing. *Computer Graphics Forum*, 5(1):3 - 12, March 1986.

[3] Hiroaki Kobayashi, Tadao Nakamura, and Yoshiharu Shigei. Parallel processing of an object space for image synthesis using ray tracing. *The Visual Computer*, 3:13 - 22, 1987.

[4] Hiroaki Kobayashi, Satoshi Nishimura, Hideyuki Kubota, Tadao Nakamura, and Yoshihara Shigei. Load balancing strategies for a parallel ray-tracing system based on constant subdivision. *The Visual Computer*, 4:197 - 209, 1988.

[5] Thierry Priol and Kadi Bouatouch. Static load balancing for a parallel ray tracing on a mimd hypercube. *The Visual Computer*, 5:109 - 119, 1989.

[6] R.W. Pulleyblank and J. Kapenga. The feasability of a a vlsi chip for ray tracing bicubic patches. *IEEE Computer Graphics and Applications*, 7(3):33 - 44, 1987.

[7] R.W. Pulleyblank and J. Kapenga. A vlsi chip for ray tracing bicubic patches. In W. Strasser, editor, *Advances in Computer Graphics Hardware I*, pages 125 – 140, 1987.

[8] Isaac D. Scherson and Elisha Caspary. Multiporcessing for ray tracing: a hierarchical self-balancing approach. *The Visual Computer*, 4:188 – 196, 1988.

[9] Bengt-Olaf Schneider. Ray tracing rational b-spline patches in vlsi. In A.A.M. Kuijk and W. Strasser, editors, *Advances in Computer Graphics Hardware II*, pages 47 – 62, 1988.

[10] Li-Sheng Shen and Ed F. Deprettere. A new space partitioning technique to support a highly pipelined parallel architecture for the radiosity method. In E.F. Deprettere and A.-J. van der Veen, editors, *Algorithms and Parallel VLSI Architectures, Volume B: Proceedings*, pages 435 – 444, 1991.