Efficient Mapping of Hierarchical Trees on Coarse-Grain Reconfigurable Architectures

F. Rivera, M. Sanchez-Elez, M. Fernandez, R. Hermida, N. Bagherzadeh[§]

Depto. de Arquitectura de Computadores y Automática, Universidad Complutense, Madrid, 28040, SPAIN
e-mail: farivera@fis.ucm.es

[§]Dept. of Electrical and Computing Engineering, University of California, Irvine, CA 92697, USA

ABSTRACT

Reconfigurable architectures have become increasingly important in recent years. In this paper we present an approach to the problem of executing 3D graphics interactive applications onto these architectures. The hierarchical trees are usually implemented to reduce the data processed, thereby diminishing the execution time. We have developed a mapping scheme that parallelizes the tree execution onto a SIMD reconfigurable architecture. This mapping scheme considerably reduces the time penalty caused by the possibility of executing different tree nodes in SIMD fashion. We have developed a technique that achieves an efficient hierarchical tree execution taking decisions at execution time. It also promotes the possibility of data coherence in order to reduce the execution time. The experimental results show high performance and efficient resource utilization on tested applications.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors) – *Single-instruction-stream, multiple-data-stream processors (SIMD)*.

General Terms: Algorithms, Design.

Keywords: Hierarchical trees, reconfigurable architectures, computer graphics, SIMD, multimedia.

1. INTRODUCTION

The emergence of high capacity reconfigurable devices is starting a revolution in general-purpose processors. Many coarse-grain reconfigurable architectures have appeared as reconfigurable coprocessors, considerably relieving the burden from the main processor in many multimedia applications due to their very high degree of parallelism. In addition, they generally have wider flexibility than an application specific circuit. These facts contribute to making them better alternatives to traditionally used DSPs or ASICs.

CODES+ISSS'04, September 8–10, 2004, Stockholm, Sweden.

Coarse-grain reconfigurable architectures are an example of reconfigurable systems. They have identical Processing Elements (PEs) richly connected through programmable interconnections. The PE functionality and connection is configured through context words stored in an internal memory to allow dynamic reconfiguration. Examples of such architectures are MorphoSys [1], REMARC [2] and MATRIX [3].

Multimedia applications are fast becoming one of the dominating workloads for reconfigurable systems. Many interactive virtual reality applications such as 3D games, virtual museum or virtual shop applications have become feasible on reconfigurable systems [1, 4].

Multimedia applications deal with large sets of data that have to be processed in a little amount of time in order to accomplish interactive results. One of the most used algorithms in 3D image processing is the hierarchical trees, in particular, the Space Partitioning Trees (SPTs). The SPTs provide a recursive hierarchical subdivision of the application's domain. They provide a computational representation of the application including a search structure and a representation of geometry. Common applications that employ SPTs are ray tracing [5] and rendering [6].

Mapping applications based on SPTs onto SIMD reconfigurable architectures is a complex task. The SPT has a fine granularity and the branch decisions must be taken locally during the execution time, but the SIMD computation model does not support this condition because it takes branch decisions globally. Therefore, mapping applications based on SPTs onto such architectures can be achieved but some time penalty is inevitable.

In this paper we propose mapping methodologies and we evaluate the time penalty of each one. Our goal is to exploit the application's data coherence in order to minimize the time penalty.

A mapping scheme study in coarse-grain reconfigurable architectures was done in [7]. In this paper the authors analyze the effects of PE interconnections in order to execute efficiently DSP applications. However, they do not deal with fine granularity loops as occurs in applications based on hierarchical trees. Another study of loop execution appears in [8], but focuses on the usage of memory operations sharing. A brief study of tree execution onto these architectures appears in [9]. However, this paper does not find an optimum mapping scheme because it focuses in ray tracing algorithm optimization.

Our work goes more deeply than previous efforts by making a detailed analysis of different mapping schemes for applications based on Space Partitioning Trees onto SIMD coarse-grain reconfigurable architectures. A proper mapping solution can exploit data coherence in a parallel way, at a reasonable amount of time and a feasible hardware cost.

This paper is organized as follows. In Section 2 we describe the target architecture. Section 3 presents the problem overview focused on applications based on hierarchical trees and Section 4 complements it with relevant aspects on data coherence. Our proposed mapping methodologies are given in Section 5 and the hardware support required is treated in Section 6. We evaluate the effectiveness of the mapping schemes in Section 7 and conclude the paper in Section 8.

2. GENERIC RECONFIGURABLE ARCHITECTURE

Reconfigurable fabrics provide massive parallelism, high computational capability and their behaviour can be configured dynamically. These coarse-grain architectures typically consist of a set of Processing Elements (PEs) connected in a 2D array, a high-speed memory interface and a main processor that controls overall operation. Based on previous features, a generic reconfigurable architecture template has been proposed [8]. This architecture is not designed for a specific application and reconfiguration can be used to cover a range of different applications.

All the PEs in a generic coarse-grain architecture are considered to be identical (it supports the same functionalities and latency remains constant) to guarantee architecture regularity. The PE array has a SIMD execution model, so the entire array has to execute the same configuration (context). Each PE is the basic unit of reconfiguration and is similar to a simple microprocessor having a data-path made up by some type and number of functional units (ALUs, multipliers, shifters) and storage units (RAM, register files). A configuration register stores its functionality and contains the operation-code and the control signals to define operation type and to select source and destination operands.

The interconnection network in the architecture usually exists among the PEs of either the same row or the same column, as well as, every PE has communication links with its nearest neighbours.

Memory interface must guarantee equally distributed memory access resources along the rows or columns. Fast transfer of computation parameters and results between the on-chip data memory and the PEs must be provided through a direct data transfer path.

Our target architecture, MG, is the implementation of MorphoSys for 3D Graphics and has the same features of any coarse-grain architecture. MorphoSys (Figure 1) is composed of a reconfigurable cell array (PE Array), a high-bandwidth memory interface and a RISC processor. The PE Array is the

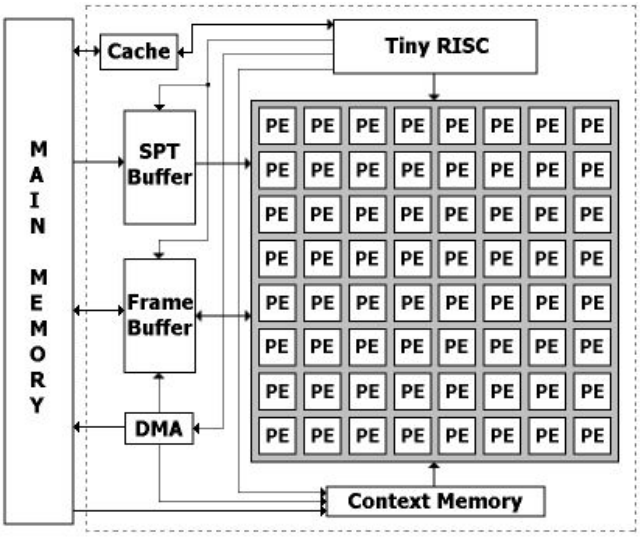

Figure 1. MorphoSys Architecture (MG)

programmable core of MorphoSys and it consists of an 8 × 8 array of PEs. A context word stored in the context register defines the functionality of the PE and the control bits for proper operation.

Morphosys high-bandwidth memory interface is implemented through the Frame Buffer, the Context Memory and the DMA controller. Frame Buffer is located between the PE Array and the main memory and it is analogous to a data cache. This buffer is organized in two sets and makes memory access transparent to the PE Array by overlapping data transfers with computation. The Context Memory broadcasts context words to the PE Array. Context words are loaded into the context register of each PE to configure it. All eight PEs in a row or column share the same context and perform the same operations.

Therefore, Morphosys supports only SIMD operations. Context Memory can be updated concurrently with the PE Array operation. The DMA controller enables fast transfers between the main memory and the Frame Buffer or the Context Memory. Simultaneous transfers of data and contexts are not possible.

3. APPLICATIONS BASED ON HIERARCHICAL TREES

In most applications involving computation on large sets of symbolic data or manipulation of 3D geometric models, partitioning trees are extremely useful. For example, in the field of interactive 3D graphics (representation and interaction over a 3D scene), the generation of every frame of an animation requires computing spatial relations like intersection and occlusion between objects. To compute spatial relations between n objects (whose number may be anything from 10^2 to 10^6) would require many operations ($O(n^2)$). The number of operations can be considerably reduced by using a data search algorithm. Space Partitioning Trees (SPTs) represent the most used search algorithms. They provide a computational representation of the space including a search structure and a representation of geometry. The SPT corresponds to a recursive hierarchical subdivision of 3D space into two or eight convex subspaces making a binary tree [10] or an octree [11], respectively. The goal behind SPTs is to quickly find the proper object instead of

checking all objects. Common applications that employ this search algorithm are ray tracing [5] and rendering [6].

The implementation of this useful search algorithm over a SIMD reconfigurable architecture is not an easy task. This is because the SPT algorithm has a fine control granularity. It implies that the branch decisions can not be taken globally for the entire PE Array; they must be taken locally inside each PE. However the PEs do not have program counter and the entire PE Array has to execute the same context. Therefore, the problem could be defined as:

"Given a SPT based application find the mapping scheme that allows its SIMD execution with the less time penalty".

A SPT has two kinds of nodes: the non-leaf nodes that contain the partitioning information such as a partitioning boundary and the pointers to their children; and the leaf nodes which are associated to a set of data, i.e. objects in the case of 3D graphics (Figure 2). The SPT search algorithm finds the data to process after searching the tree. This process takes lower computation time than searching all the application data, because only a subset of nodes is processed. In order to execute an application based on hierarchical trees, the search algorithm begins at the root node to find the next visited nodes until it reaches a leaf. The root node is loaded in the PE Array and in each PE is loaded different input data of the algorithm. The next node to process is obtained as a result of the SPT algorithm execution over those input data. If the next node is non-leaf, the SPT search algorithm continues. In the case of the leaf node the application is executed over the leaf node data. When a leaf is found, its type is identified and the corresponding process is executed. This means that the application is made up by as many different processes as types of leafs the tree has, besides the tree search process.

Reconfigurable fabrics, as explained above, are arranged in a SIMD style. This means that the entire reconfigurable array has to execute the same context on different data. In the particular case of the Space Partitioning Tree search algorithm, it could imply that different PEs need to execute different contexts at the same time, for example, leaf nodes and non-leaf nodes contexts. Since this is not possible in SIMD way, it is convenient that the data processed at the same time are data coherent to minimize this overhead.

4. DATA COHERENCE

Coherence means that several PEs may traverse the same nodes and test the same set of leaves. Therefore, a coherent group of PEs requests the same data set. If we take this into consideration the execution model can be simplified. In fact, a mapping scheme that promotes the coherence can achieve the best performance results with coherent data. However, when a set of PEs that is supposed to be data coherent turns out to be incoherent a special scheme must be applied to process different requests in different phases. These separate phases generate an overhead that has to be minimized.

For a typical $N\times N$ reconfigurable array there are several possible parallel schemes. They are: (1) all $N\times N$ PEs are assumed to be data coherent; (2) G groups $K\times N$ are coherent, wherein N is a multiple of K; (3) N groups $1\times N$ are coherent. Option (1) requires the smallest memory bandwidth and option (3) requires the largest

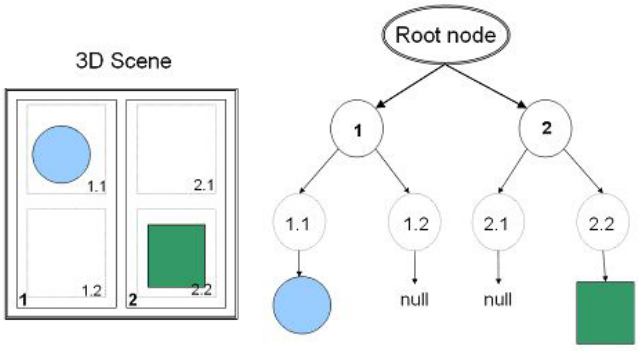

Figure 2. SPT Algorithm

one. So whenever it is possible, option (1) should be chosen. However, finding $N\times N$ PEs data coherent is really difficult. The number of coherent PEs is usually small. In this paper we suppose that our application has $1\times N$ coherence which, for example, is the case of ray tracing in MorphoSys [9], all the PEs in the same row (column) are supposed to be coherent. Moreover, a study in $1\times N$ can be easily extrapolated to the $K\times N$ and the $N\times N$ cases. We call this a 2-level SIMD mapping scheme, since it is organized into two levels: each group of N PEs are assumed to be data coherent, and the N groups of N PEs are allowed to process different nodes. This means that in the MorphoSys case all the PEs in the same row or column are supposed to be data coherent, but different rows or columns can process different data.

The level of data coherence the application has, depends directly on how many times the tree is traversed to complete the application. This means that with a high number of times it is easier to find groups that require the same data. For example, in the case of 3D image processing the SPT search algorithm is executed as many times as the number of screen pixels. Therefore, an increase in the screen size ($M\times M$) entails more data coherence. Clearly, the data coherence depends also on the number of PEs that are supposed to be coherent ($1\times N$): a small N value makes coherence easier. In order to evaluate how a mapping scheme promotes the coherence, we calculate the PE Unbalance Factor (t_{PE}). This factor shows idleness of PE.

$$t_{PE} = t\frac{N\times N}{M\times M}$$

Wherein t stands for the application execution time; $N\times N$ stands for the PE Array size; and the product $M\times M$ stands for the number of times that the SPT algorithm is executed to complete the application.

The t_{PE} value indirectly indicates the coherence whilst maintaining the application data size; an increase in t_{PE} means that with that mapping scheme the PE is more time idle because there is less coherence as will be explained in the next section. This t_{PE} value can be used to check if changes in hardware resources improve their usage. For example, an increase in the PE Array size usually implies a reduction in the application execution time. But only when there is a decrease in t_{PE}, a better exploitation of the hardware resources is achieved. This is because a reasonable amount of data coherence is maintained. On the other hand, this factor is independent of the number of times that the tree is traversed, so an increase in the $M\times M$ factor that always means a

rise in the execution time (t) entails a reduction in the t_{PE} when it increases data coherence.

5. SIMD COHERENT MAPPING SCHEME

Programming frameworks for coarse-grain reconfigurable architectures include many tasks like kernel extraction and mapping onto the architecture, bandwidth requirements and additional overhead estimation, all of them having great influence on the performance of the implementation. In this section we analyze some mapping solutions. As was explained above, the goal is to find one solution that minimize the execution time, and therefore promotes the data coherence.

In the MIMD solution PEs can perform different operations on different data. MIMD style leads to the minimum execution time for the application because on the same execution step all the PEs are performing different and useful computations. Memory overhead, network topology and communication delays among PEs must be carefully considered in order to obtain optimum performance. However, in almost all coarse-grain reconfigurable architectures that have been proposed, the PEs are typically arranged in SIMD style to exploit spatial and temporal mapping. Therefore, SIMD promotes inherent application's parallelism, and reduces memory overhead and control hardware. Our 2-level SIMD mapping scheme mentioned previously is focused on getting a balance between spatial and temporal mapping as we shall see.

In the SIMD computation model, the entire PE Array will perform the same operations on different data. However, in many applications it is very unlikely that all the PEs execute the same operation at the same time. This depends on how much coherence the application has. If data coherence is high enough we should have the execution of identical processes on nearby data. This means that PEs operating on a set of nearby data that are supposed to be coherent, will traverse the same nodes and execute the same processes associated with the found leaves. The execution model which encourages coherence achieves the best performance.

When an application based on hierarchical trees starts running in MorphoSys, the entire PE Array operates on the root node data. As we have assumed, N rows (columns) of N coherent PEs traverse the tree in a parallel way. The N PEs forming a row will operate on the same node, but different rows can process different nodes. When executing ray tracing, each PE computes one pixel over node data. Therefore, during an execution step $N \times N$ pixels will be processed in the PE Array against N nodes. In every search algorithm step the next visited nodes are found. As a row is computing N different pixels against the same node the following cases can occurs: (1) All N found nodes are non-leaf nodes; and (2) At least one of the N found nodes is a leaf node. If all N rows are in case (1), search algorithm keeps on executing. If the second case appears in any row a problem arises because on the next execution step some PEs should be processing non-leaf nodes and some of them should be operating on leaf nodes: SIMD style does not allow execute search algorithm and at the same time to process a leaf on the PE Array. Mapping solutions for this kind of applications onto SIMD architectures must define when to pass from the tree search algorithm, to start executing processes associated with leaves reached by several PEs. Also, the mapping solution must determine how to continue traversing the tree when leaf processing are completed. This must be achieved with the lowest time penalty and the mapping solutions have to promote that all the PEs perform useful work.

There are three possible choices for mapping (Table 1):

1. Execute tree search algorithm on the N rows of N PEs until some PE reaches a leaf (some PEs can reach a leaf simultaneously). In this moment leaf processing starts, but only on the PEs that have reached a leaf, keeping idle all the others to reduce power consumption. When leaf processing has been executed, the tree search is continued.

2. Execute tree search algorithm on the N rows of N PEs. When a PE reaches a leaf, the process associated with the leaf found starts, but over the entire PE Array. That is, all the $N \times N$ PEs will process the same leaf node. Some PEs can reach a leaf simultaneously, but only the different and previously unprocessed leaves are stacked. Once all stacked leaves have been executed, the tree search is continued.

3. Execute tree search algorithm on the N rows of N PEs, stalling those PEs that have reached a leaf. The leaf reached must be stacked and the corresponding PE continues executing the search algorithm. Each row has its own stack. Every time all the N stacks have leaves stored, tree search algorithm stops and leaf processing starts. This means that leaf processing is executed only when all rows found a leaf. For each row only the different and previously unprocessed leaves are stacked. Tree search is continued while there are no leaves stacked in all the row stacks. When tree search finishes, stacked leaves that have been postponed are processed.

Table 1. Mapping solutions

Mapping Scheme	Non-leaf execution	Leaf execution
1st Solution	Execute non-leaf process until one leaf is found	Execute the leaves over the PEs that found them
2nd Solution	Execute non-leaf process until one leaf is found	Execute every leaf over the entire PE Array
3rd Solution	Execute non-leaf process until one leaf is found per group (column-row)	Execute the leaves over the corresponding group (column-row)

In the first choice some PEs can be idle while the others are processing reached leaves. Typically, algorithms associated with leaves are more complex and therefore time demanding. This means that in the first option some PEs can be idle for a significant amount of time. The number of idle PEs in leaf processing depends inversely on the degree of data coherence. A large number of idle PEs is shown by a higher t_{PE}. In the second alternative all the entire PE Array executes the process associated with a found leaf. This means that the same process is executed in the array, using the idle time of some PEs to perform a computation that can be useful in the future. The performance of this scheme depends on the degree of coherence since the leaf results are stored for subsequent use. A higher degree of data coherence implies that the speculative computations are useful.

This condition means a lower value for t_{PE}. If these computations are useless they will generate a power overhead. The third mapping solution avoids speculative execution of leaves trying to reduce time and power overhead. In this case, leaf processing uses simultaneously all the PE Array resources when the number of found leaves is enough to fill it. All the leaves processed represent useful computations because they are required for the algorithm. Only when the stacked leaves that have been postponed are processed can appear some idle PEs. The number of postponed leaves and, therefore, the number of idle PEs depends on the degree of data coherence. A higher degree of data coherence results into a reduced number of idle PEs and a lower value for t_{PE}.

In the case of 3D graphics the data coherence is high, at least locally. However, sometimes there can be incoherence inside a row (column). This means that it does not evolve synchronously and requires very different data to complete the algorithm. It is for this reason that a multi-pass strategy must be employed to process different requests on different passes to guarantee the correctness of the algorithm. A lack of coherence produces time overhead because of the execution of irrelevant process on some PEs or waste of hardware resources because of the presence of idle PEs. Hence, the performance of any mapping solution depends on application's data coherence.

6. HARDWARE SUPPORT REQUIRED

The mapping of SPT algorithm onto a reconfigurable fabric is difficult and inefficient without special hardware support units. Of course we have to be sure that the cost of these hardware supports is small and will not compromise the original architecture.

The possibility of incoherence sometimes results in several PEs not having to process data. We include pseudo-branch instructions [12] and extra hardware in the PE to make possible that any PE can be idle when required.

We also add a new memory, the SPT Buffer to transfers data from/to the external memory to/from the PE Array in a non-streaming pattern with pointer-jumping behavior. The internal memory (FB) enables streaming data transfers in the reconfigurable fabrics. However, if the data are not consecutive in the FB, there is a time penalty. In the case of the SPT buffer, if there is no coherence, the PEs can traverse different nodes and the data could be in different addresses in the memory. Since our mapping scheme supposes that each column is data coherent, the SPT Buffer is organized as eight banks, each of which provides data to one column of eight PEs. The eight banks of the SPT Buffer are addressed separately. The access to SPT Buffer can be either on eight banks concurrently or only on one of them. Each bank provides a 32-bit data element. When eight banks are accessed a total of eight data elements are broadcast to the eight columns.

7. EXPERIMENTAL RESULTS

In this section we present the experimental results for different experiments. We have used the MG architecture to test these experiments and developed an interactive ray tracing algorithm for it [9]. Ray tracing involves projecting rays into the computational model of 3D space and resolving intersections and occlusions to define what color to display at each point of the screen. The computational representation of 3D space used by the algorithm is based on a space partitioning octree; recursive hierarchical subdivision of 3D space into eight convex subspaces. Our goal is to explore the performance of the mapping solutions previously proposed.

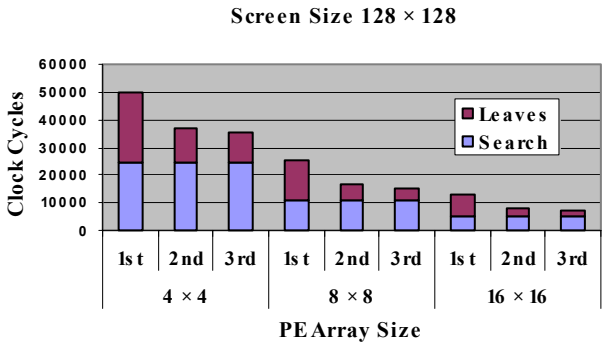

Figure 3. Execution time for different mapping schemes

We have modelled different experimental setups by varying hardware and algorithm parameters like screen and PE Array sizes. In order to evaluate the mapping solutions we have implemented the ray tracing algorithm in a profiler framework. This profiler framework executes ray tracing on the target architecture and delivers the number of clock cycles consumed by the search algorithm and the leaf processing.

In Figure 3 we compare the three mapping solutions based on their execution times, corresponding to one specific screen size. The first mapping solution takes the maximum number of cycles for all the experimental configurations. The highest number of cycles is necessary for leaf processing. This is because the same leaf is usually processed several times. The number of idle PEs grows inversely proportional with the degree of data coherence, so the first mapping solution leads to a poor use of hardware resources that negatively impact on the execution time. The second mapping solution reduces the number of leaf processing cycles in comparison to the first one, because this mapping scheme only processes different leaves and just one at a time over the entire PE Array. The number of cycles employed on leaf processing in every SPT algorithm execution is proportional to the number of different leaves reached by the N rows (columns) of $1 \times N$ PEs that make up the array. On its way, the third mapping solution performs more efficiently the leaf processing. In this case, the number of cycles employed on leaf processing in every SPT algorithm execution is proportional to the maximum number of different leaves reached by one of the N rows (columns) of $1 \times N$ PEs that comprise the array. In the worst case this number is equal to that of the second mapping solution. Typically, leaf processing are complex and time consuming. Therefore, the third mapping solution represents the best of the three mapping solutions, from those proposed, in terms of execution time.

In Figure 4 we compare the three mapping solutions based on the t_{PE} value. For all the experimental configurations the third mapping solution shows the lower value for t_{PE}. The third mapping has the lower number of idle PEs and each PE spends, on average, less time searching and processing the tree and, therefore, exploits the data coherence in a better way compared

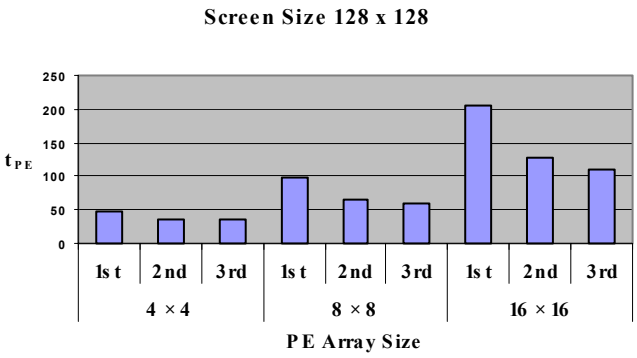

Figure 4. PE Unbalance Factor

with the other methods. In general, for a fixed PE Array size, an increase in the number of times that the SPT algorithm is executed to complete the application yields a lower t_{PE}. This means an increase in the data coherence. Also, for a fixed number of times that the SPT algorithm is executed to complete the application, an increase in the PE Array size does not mean an increase in data coherence but a decrease as shown by a higher t_{PE}. Table 2 shows these aspects for the third mapping solution.

Table 2. Effects of PE Array and screen sizes on t_{PE}

t_{PE} Value (Third Mapping Solution)			
	PE Array Size		
Screen Size	**4 × 4**	**8 × 8**	**16 × 16**
16 × 16	65.81	133.25	259.00
64 × 64	42.76	80.09	172.00
128 × 128	34.72	59.79	108.75

8. CONCLUSIONS AND FUTURE WORK

We have explored three different mapping schemes of hierarchical trees on coarse-grain reconfigurable architectures: (1) Execute non-leaf process until one leaf is found and then execute the leaves over the PEs that found them; (2) Execute non-leaf process until one leaf is found and later execute every leaf over the entire PE Array; and (3) Execute non-leaf process until one leaf is found per group (column-row) and then execute the leaves over the corresponding group (column-row). All of these have been evaluated according to their time penalty caused by the possibility of executing different tree nodes in SIMD style. Our experimental results show that the third mapping solution is more efficient with data coherence, achieves the highest performance because of its effective leaf processing and makes better use of hardware resources by avoiding the growth of idle PEs.

In order to execute efficiently applications based on hierarchical trees and any interactive applications, the scheduling should be done at execution time. Therefore, future work will address a new task and data management.

9. REFERENCES

[1] H. Singh, M. Lee, G. Lu et al., "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications," *IEEE Transactions on Computers*, Vol. 49, No. 5, May 2000.

[2] T. Miyamori and K. Olukoton, "REMARC: Reconfigurable Multimedia Array Coprocessor," *Proc. ACM/SIGDA International Symp. FPGAs*, Feb. 1998.

[3] E. Mirsky, A. DeHon, et al. "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," *Proc. IEEE Symposium FCCM*, Apr. 1996.

[4] M. Meissner, S. Grimm, W. Strasser, J. Packer and D. Latimer, "Parallel Volume Rendering on a Single-Chip SIMD Architecture," *Proc. IEEE Symp. Parallel and Large-Data Visualization and Graphics*, pp. 107-157, Oct. 2001.

[5] A, Glassner, "An Introduction to Ray Tracing," Academic Press, 1989.

[6] J. Arvo (Ed.), *Graphics Gems II*, Academic Press, 1991.

[7] N. Bansal, S. Gupta, N. Dutt et al., "Network Topology Exploration of Mesh-Based Coarse-Grain Reconfigurable Architectures," *Proc. of Design, Automation and Test in Europe Conference (DATE04)*, Feb. 2004.

[8] J. Lee, K. Choi et al., "Mapping Loops on Coarse-Grain Reconfigurable Architectures Using Memory Operation Sharing," TR 02-34, Center for Embedded Computer Systems, University of California, Irvine, Sep. 2002.

[9] M. Sanchez-Elez, H. Du, N. Tabrizi et al., "Algorithm Optimizations and Mapping Schemes for Interactive Ray Tracing on a Reconfigurable Architecture," *Computer & Graphics* (27), pp. 701-713, Elsevier, 2003.

[10] K. Sung, P. Shirley, "Ray Tracing with the BSP Tree", *Graphics Gems III*, pp. 271-274, Academic Press, 1992.

[11] J. Revelles, C. Urena and M. Lastra, "An Efficient Parametric Algorithm for Octree Traversal," *Proc. WSCG*, pp. 212-219, 2000.

[12] M. Anido, A. Paar et al., "Improving the Operation Autonomy of SIMD Processing Elements by Using Guarded Instructions and Pseudo-Branches," *Proc. of EUROMICRO DSD*, pp. 148-155, Sep. 2002.