Low Energy Data Management for Different On-chip Memory Levels in Multi-Context Reconfigurable Architectures*

M. Sánchez-Élez, M. Fernández, M. Anido†, H. Du§, N. Bagherzadeh§, R. Hermida

Dept. de Arquitectura de Computadores y Automática – Universidad Complutense de Madrid – 28040 SPAIN
†Nucleo de Computacion Electronica – Federal University do Rio de Janeiro – Brazil
§Dept. of Electrical and Computer Engineering – University of California, Irvine – CA 92697 USA

Abstract:
This paper presents a new technique to improve the efficiency of data scheduling for multi-context reconfigurable architectures targeting multimedia and DSP applications. The main goal is to improve application energy consumption. Two levels of on-chip data storage are assumed in the reconfigurable architecture. The Data Scheduler attempts to optimally exploit this storage, by deciding in which on-chip memory the data have to be stored in order to reduce energy consumption. We also show that a suitable data scheduling could decrease the energy required to implement the dynamic reconfiguration of the system.

1. Introduction

The emergence of high capacity reconfigurable devices is igniting a revolution in general-purpose processors. It is now possible to tailor and dedicate reconfigurable units to take advantage of application dependent dataflow. These reconfigurable systems combine reconfigurable hardware units with a software programmable processor. They generally have wider applicability than an application specific circuit. In addition, they attain a better performance than a general-purpose processor for a wide range of computationally intensive applications.

FPGAs [1] are the most common fine-grained devices used for reconfigurable computing. Dynamic reconfiguration [2] has emerged as a particularly attractive technique for minimizing the reconfiguration time, which has a negative effect on FPGA performance. Multi-context architectures are an example of dynamic reconfiguration, which can store a set of different configurations (contexts) in an internal memory. When a new configuration is needed, it is loaded from this internal memory which is faster than reconfiguration from the external memory. Examples of such chips are MorphoSys SIMD system-on-chip [3] or FUZION architecture [4].

Multimedia applications are fast becoming one of the dominating workloads for reconfigurable systems. For these applications the system energy is clearly dominated by memory access. Most of these applications apply block-partitioning algorithms to the input data. Within these blocks there is a significant data reuse as well as spatial locality [5]. These blocks are relatively small and can easily fit into reasonably sized on-chip memories. Therefore, a data scheduling at this block level significantly reduces the system energy consumption.

Previous work [6][7][8] discussed scheduling for multi-context architectures, in particular using MorphoSys. It achieves high performance compared with other approaches, on several DSP and multimedia algorithms. These applications are typically composed of a group of macro-tasks (kernels), which are characterized by their contexts and their data.

A kernel scheduling technique is proposed in [6] to generate a kernel sequence that estimates the execution time through tentative context and data schedules. Context scheduling is discussed in [7] and its goal is to minimize the number of context loads. A first approach to data scheduling was discussed in [8] but it does not deal with energy minimization and different on-chip memory levels. The method presented in this paper optimizes data storage in different on-chip memory levels, reducing the energy consumption.

Although previous work has been done related to energy optimizations, this kind of data management for reconfigurable systems has not been discussed in detail by other authors. A review of data memory design for embedded systems is discussed in [9]; but it does not take into account the reconfiguration energy consumption. A method to decrease the power budget by transforming the initial signal or data processing specifications is suggested in [10], but it deals with fine-granularity problems. In [11] a data scheduler for dynamic architectures is proposed, though it does not optimize memory management and energy consumption. A method to minimize memory traffic is suggested in [12], but it does not deal with memory allocation in detail. However, regarding memory organization most effort has been made in cache organization for general-purpose computers, see [13][14], and not on more custom memory organizations, as needed for example by multimedia and DSP applications.

* This work was sponsored by the New Del Amo program.

1530-1591/03 $17.00 © 2003 IEEE

The paper begins with a brief overview of MorphoSys and its compilation framework. Section 3 describes the problem. Section 4 analyzes data management in different on-chip memory levels. A data management in FB to reduce energy consumption is discussed in Section 5. Experimental results are presented in Section 6. Finally we present some conclusions from our research in Section 7.

2. Architecture and framework overview

This section describes the target system M2 (second implementation of MorphoSys). We also present the development framework that integrates the data scheduler with other compilation tasks.

MorphoSys [3] (Figure 1) consists of an 8x8 array of reconfigurable cells (RC). Its functionality and interconnection network are configured through 32-bit context words. The context word controls the ALU functions, the usage of registers, the internal RAM, etc, of each RC. These context words are stored in a context memory (CM).

The frame buffer (FB) serves as a data cache (level 1) for the RC Array; it is logically organized into two sets. This arrangement helps in overlapping computation with data load/store. Data from one set is used for current computation, while the other set stores results in the external memory and loads data for the next round of computation. Moreover, each cell has an internal RAM (level 0 cache) that can be used to store the most frequently accessed data and results. The Data Scheduler reduces these data and results transfers and optimizes their storage to minimize energy consumption.

The DMA controller establishes the bridge that connects the external memory to either the FB or the CM. Thus, simultaneous transfers of data and contexts are not possible. A RISC processor controls MorphoSys operation.

An overview of the proposed compilation framework is shown in Figure 2. The application assembly code and contexts are written in terms of kernels that are available in a kernel library. The information extractor generates the information needed by the compilation tasks that follow it,

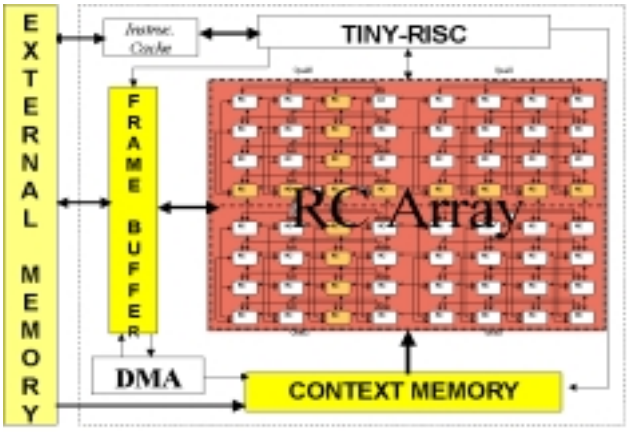

Figure 1: M2 MorphoSys Chip

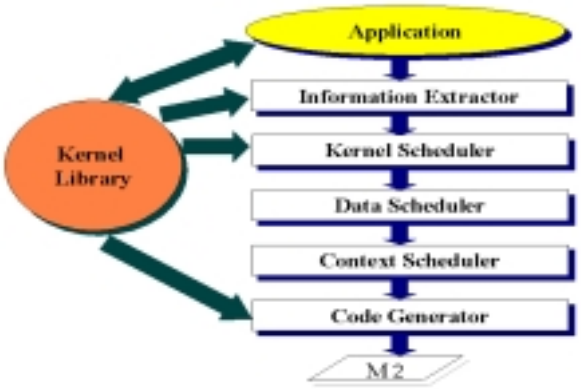

Figure 2: Compilation Framework

including kernel execution time, data size and number of contexts for each kernel.

The kernel scheduler [6] explores the design space to find a sequence of kernels that minimizes the execution time. It decides which is the best sequence of kernels and creates clusters. The term cluster is used here to refer to a set of kernels that are assigned to the same FB set and whose components are consecutively executed. For example, an application is composed of kernels k_1, k_2, k_3, k_4; the kernel scheduler, estimating context and data transfers, could assign k_1 and k_2 to one FB set, and k_3 and k_4 to the other set. This implies the existence of two clusters $c_1=\{k_1, k_2\}$ and $c_2=\{k_3, k_4\}$. The first cluster is being executed using data of one FB set and/or internal RAMs, meanwhile the contexts and data of the other cluster kernels are being transferred to CM and to the other FB set respectively.

The kernel scheduler generates one possible kernel sequence that minimizes the overall execution time. The context and data schedulers specify when and how each transfer must be performed to reduce the energy budget.

The code generator builds the optimized code that is going to be implemented in M2.

3. Problem Overview

We propose in this paper a new methodology to perform data scheduling on a given set of clusters.

The problem could be defined as: "*Given an ordered set of clusters with their data and results sizes, and a memory hierarchy, find the data scheduling that minimizes the energy consumption*".

As [5] explains, multimedia applications have a significant data reuse as well as spatial locality at block level (kernel level). The Data Scheduler makes good use of this characteristic to reduce energy. If several data have a reasonable reuse, these can be stored in the low energy memory to minimize energy consumption as is explained below. Energy is calculated by multiplying time and power.

The MorphoSys execution model allows overlapping of context and data transfers with system computation, but it does not allow simultaneous transfers of data and contexts. Therefore the execution time [6] for a cluster c is the

greater of either the sum of data, results and context transfer times or the sum of kernel computation times. For multimedia applications the data and results transfers are more time consuming than kernel computation time.

The Data Scheduler reduces data and results transfers to reduce application execution time. It keeps the cluster data or results in FB, which are later used by other clusters, in order to avoid unnecessary transfers.

The Data Scheduler uses FB free space to store data in executed kernels RF consecutive times (Context Reuse Factor [8]) to reduce contexts transfers.

On the other hand, we consider that the memory access is a major contributor to the overall power consumption [14]. The main sources of memory power are: the power dissipated in the bit-lines, the power dissipated in word-lines, and the power used to drive the external buses. Though memory access has the higher power contribution, we also take into account the power dissipated by reconfigurable hardware elements and RISC processor. We are not considering the energy dissipated in the address decoders, since we found this value to be negligible compared to the other components [15].

The power dissipated in the bit and word lines depends on memory size, in smaller memories a smaller capacity is switched. The Data Scheduler reduces power consumption by storing the most frequently used data in the smallest on-chip data memories. In addition, the Data Scheduler allows consecutive kernel executions, which reduces context transfers by a factor RF, the power consumption of context memory is also diminished. The Data Scheduler reduces data and context transfers, which implies that the power used to drive the external buses is also reduced.

MorphoSys' speed is limited by the access time to FB. Each RC has a small internal RAM tightly coupled with it, only used to store executed square root, sine or cosine algorithm data. But these memories could be used to store any other data and results since algorithm data do not occupy the whole RC-RAM and they are not used in many applications. In addition, Tiny RISC makes the instructions, which do not depend on FB data running faster than the dependent ones. As RC-RAMs are within RC, they are quicker and also smaller than FB. Therefore, if the Data Scheduler uses these RC-RAMs to store the most accessed data, it reduces energy consumption, as we discuss in section 4.

Although our approach to solving this problem targets one particular reconfigurable system, MorphoSys, this approach is quite general in nature and may be easily applied to other reconfigurable and embedded systems.

4. Low Energy RC-RAM Management.

In this section we discuss a low energy RC Data Management.

MorphoSys has a hierarchical memory design wherein RC-RAM is the lowest hierarchy level. This RC memory was only used to store data for special algorithms because data transfers to/from RC-RAM were very time consuming in the M1 chip (7 cycles to transfer one word). These algorithm data were loaded previous to application execution and there was no transfer between FB and RC-RAM during the execution.

In M2 we have developed new instructions and RC contexts, which allow 1 cycle data transfers to/from RC-RAM. These transfers are carried out through DMA. A new Tiny RISC instruction loads DMA with the FB start address, number of words, transfer type (load/store) and RC cell. RC-RAM access is controlled by new contexts, with the address register and the transfer type. Within the M2 reconfigurable cell we add a counter, which increases the RC-RAM access address in each clock cycle until DMA generates a stop signal. Moreover, a multiplexer at the RC-RAM port allows both transfers to/from FB and to/from RC registers. This method makes one cycle transfers between FB and RC-RAM possible, and lets the Data Scheduler optimize data storage in on-chip memories.

The execution time of instructions that work on RC-RAM data is less than that of instructions working on FB data. Power consumption is also smaller in RC-RAM than in FB, therefore load data in RC-RAM should reduce energy consumption. However, the data storage in RC-RAM does not always reduce energy consumption. There is an increase in execution time due to the data transfers from FB to RC-RAMs having to be done within their cluster computation time, instead of overlapping with previous cluster computation, as occurs in transfers between external memory and FB. And there is an increase in power because data must be loaded into the FB before loading them into RC-RAM. Therefore, if several data are transferred from FB to RC-RAM and they are read just once, the improvement in energy is negative. However, if they are read n_{min} times, the decrease in energy from using data in RC-RAM is larger than the energy wasted in transferring data from FB to RC-RAM (Figure 3). If the energy consumption of the data when stored in RC-RAM is less than the energy consumption when stored in FB, the Data

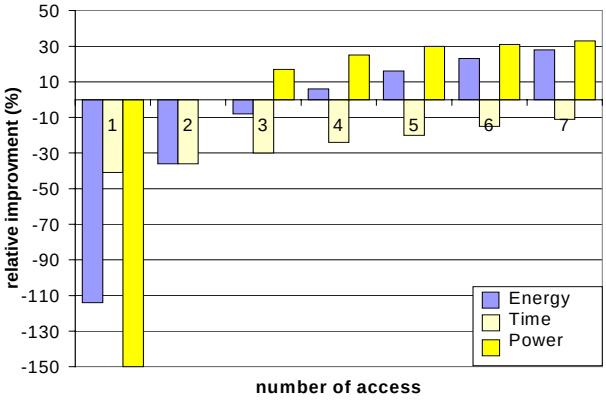

Figure 3: Relative Improvement when data are stored in RCRAM (RCRAM size = $^1/_4$ FB size)

Scheduler must store them in the RC-RAM. (i.e. more than 3 instructions that read these data in a "RC-RAM = FB/4" case). In addition, there are several data shared among clusters or several results used as data by some clusters. The Data Scheduler attempts to keep these data or results in RC-RAM instead of reloading them in each cluster execution, to increase energy reduction.

EF_{RC} reflects the energy reduced if these data or results are stored in RC-RAM:

$$EF_{RC}(D) = (n_{acc} - n_{\min}(d)) \cdot D(u,v)^2 \cdot (N)^2$$

$$EF_{RC}(R) = (n_{acc} - n_{\min}(r)) \cdot R(u,v)^2 \cdot (N)^2$$

n_{acc}: the number of instructions that read these data or results

$n_{min}()$: the minimum number of instructions to improve energy consumption. (d: data; r: results).

D(u,v): the input shared data size of data stored in the RC-RAM for cluster u execution and not used after cluster v execution.

R(u,v): an intermediate shared results size produced in cluster u execution and not used as data after cluster v execution.

N: the number of clusters that read these data or results.

The Data Scheduler stores data and results following the energy factor until no more data fit into RC-RAM. The data size stored in the RC memories can be obtained by the sum of all data and results sizes (RCDS). It takes into account that data have to be stored before the first cluster that uses them is executed. The results are stored while the kernel that produces them is executed and the space occupied by the results or data that are not going to be used again is used to store new data or results.

$$RCDS = \underset{c\in\{1,\dots N\}}{MAX}\left\{ \sum_{u=1,v=c}^{c,N}\left(\underset{i\in\{1,\dots n\}}{MAX}\left[\sum_{j=i}^{n} d_j(u,c) + \sum_{j=1,t=i}^{i,n} r_{jt}(c,v) \right] \right) + \right.$$
$$\left. + \sum_{u=1,v=c+1}^{c-1,N} \left(D(u,v) + R(u,v) \right) \right\}$$

$d_j(u,c)$: the input data size for kernel j of cluster c, except those shared with kernels executed after k_j $\{k_{j+1}, \dots, k_n\}$; these data were stored in the RC-RAM for cluster u execution.

$r_{jt}(c,v)$: an intermediate results size of k_j of cluster c, which are data for k_t and not for any kernel executed after k_t of cluster c; these results were produced in cluster u execution.

The Data Scheduler sorts the data and results to store in RC-RAM according to EF_{RC}. It starts checking that the data with the highest EF_{RC} fit in the RC-RAM (RCDS $\leq$ RAM_size). Scheduling continues with data or results with less EF_{RC}. If RCDS > RAM_size for some data or results, these are not kept. The Data Scheduler keeps the highest possible amount of data or results that minimizes energy consumption. The algorithm results is RC_list(c,i), a list of data and results stored in RC_RAM for cluster c and cell i, ordered by energy improvement.

The Data Scheduler repeats this algorithm for all the 64 RC cells. However, in most cases, this is not necessary because the 64 RC cells have the same execution behavior, though on different data.

5. Low Energy FB Management

The Data Scheduler has to take the RC-RAMs into account to calculate the FB management. These new memories complicate scheduling and could increase energy consumption. In this section we discuss the changes made to the data scheduler developed in [8] to handle RC-RAMs and to reduce energy consumption.

Cluster data and results have to be stored in the FB even though they are transferred to RC-RAM or to external memory. Therefore, the data size stored in FB can be obtained by the sum of all data and results used and generated by all kernels. The data transferred to RC-RAM have to be stored initially in FB, before execution of the first kernel in the cluster. The space occupied by these data is released when they are stored in RC-RAMS (before execution of the first kernel in the cluster), and their space can be used to store the results of cluster kernels. On the other hand, the results stored in RC-RAM have to be stored in FB before they are transferred to the external memory. But this is done after the last kernel in the cluster is executed.

Multimedia applications are composed of a sequence of kernels that are consecutively executed over a part of the input data, until all data are processed. This part of the input data depends on application scheduling and FB size, for example *n* times to process the total amount of data. In this case their contexts may be loaded into CM *n* times. However, sometimes loop fission can be applied to execute a kernel *RF* consecutive times before executing the next one (loading kernel data for RF iterations in FB). In this case

```
c = cluster with the maximum RFFB;
(c',i) = RC-cell and its cluster with the maximum RFFB;
RC_list(c,i);
fin=0;
while (fin==0)        {
      RF_FB = calculate_RFFB(c);
      RF_RC = calculate_RFRC(c',i);
      if RF_FB<= RF_RC then {
            RF = RF_FB;
            fin=1;
      } else              {
            E_FB = e_improvement_FB(RF_FB);
            E_RC = e_improvement_RC(RF_RC);
            if    E_FB>E_RC then {
                  reduce_RC(RC_list(c,i));
                  (c',i) = recalculate(RC_list);
            } else RFFB = RFFB – 1;
}           }
```

Figure 4: Algorithm to calculate RF.

kernel contexts are reused because they have to be loaded only *n/RF* times, reducing context transfers from the external memory and thus minimizing energy consumption. The number of consecutive kernel executions *RF* (Reuse Factor) is limited by the internal memory sizes.

The Data Scheduler finds the maximum common *RF* value in FB (RF_{FB}) and also the maximum RF value for each cell (RF_{RC}). However, RF may be the same for all clusters and cells, on account of data and results dependencies. The Figure 4 algorithm is applied to decide which is going to be the application RF value.

The procedure *calculate_RFFB(c)* finds the maximum RF value for cluster c. The procedure *calculate_RFRC(c,i)* finds the maximum RF value for RC cell i RCRAM for cluster c. The procedure *e_improvement_FB(RFFB)* calculates the energy improvement if kernels are executed consecutively RF_FB times (RF_RC = RF_FB). The procedure *e_improvement_RC(RF_RC*) calculates the energy improvement if kernels are executed consecutively RF_RC times (RF_FB = RF_RC). The procedure *reduce_RC(RC_list(c,i))* extracts from the list the data or result with least energy improvement, where if some data have the same energy the procedure extracts the largest one. The procedure *recalculate(RC_list)* calculates the new clusters and cell with the maximum data size.

If *RF_FB* < *RF_RC* then *RF* = *RF_FB* because all data have to be stored beforehand in FB. However, if *RF_FB* > *RF_RC* the algorithm finds the RF that minimizes energy. This can be achieved by decreasing RF_{FB} or increasing RF_{RC}. The later is the result of reducing the data stored in RC-RAM.

Context and data transfers are very energy consuming. In order to reduce context transfers, the Data Scheduler achieves the highest common RF value to all clusters. However, in most cases, though context transfers are minimized, there is wasted energy on account of unnecessary data transfers between FB and the external memory.

Data transfer reduction between FB and external memory was discussed in [8]. The difference between that scheduling and this scheduling is the RCRAM memory management. The Data Scheduler finds the external data and intermediate results shared among clusters but not stored in any RC-RAM. The Data Scheduler attempts to keep these in FB instead of reloading them to increase energy reduction. EF_{FB} reflects the energy reduced if these data or results are kept in FB:

$$EF_{FB}(D) = D(u,v)^2 \cdot (N-1)^2$$

$$EF_{FB}(R) = R(u,v)^2 \cdot (N+1)^2$$

The Data Scheduler stores data and results following the energy factor until no more data fit in FB.

The Data Scheduler reduces energy because data and results used by cluster kernels are loaded only once even though more than one kernel uses them. Memory write and read operations are executed mainly in the RC-RAMs which consume less energy than FB. The Data Scheduler minimizes write and read to/from the external memory due to the fact that it tries to transfer only the input data and the output results of the application, keeping the intermediate results, when possible, in the FB and RC-RAMs.

Although the number of memory accesses could not be minimized, there are other techniques to minimize energy consumption. For example, reducing switching activity on the memory address and data buses produces a decrease in energy consumption. These energy reductions can be brought about by the appropriate reorganization of memory data, thus consecutive memory references exhibit spatial locality. This locality, if correctly exploited, results in a power-efficient implementation because, in general, the Hamming distance between nearby addresses is less than that between those that are far apart [16]. During the cycles for which the data-path is idle, all power consumption can then be easily avoided by any power-down strategy. A simple way to achieve this is, for example, the cheap gated-clock approach [17].

6. Experimental Results

In this section we present the experimental results for a group of synthetic and real experiments, in order to demonstrate the quality of the proposed methodology. As a real experiment we have developed a ray-tracing algorithm for MorphoSys. It projects rays into the computer's model of the world to determine what color to display at each point in the image [18].

The data scheduling depends on kernel scheduling, data size and available internal memory. We analyze different kernel schedules for different memory sizes as shown in Table 1. We compare the previous data scheduling [8] and the data scheduling proposed in this paper with the Basic Scheduler [6]. E1 stands for the relative energy improvement on the previous data scheduler, E2 stands for the relative energy improvement of the current Data Scheduler and E1-2 compares the current Data scheduler with the previous data scheduler.

We have tested the same kernel schedules for different memory

	N	n	DS	RAM	FB	RF	E1	E2	E1-2
A1	5	4	1.1	0.125	0.5	2	45%	53%	18%
A2	5	4	1.1	0.06	0.25	1	5%	20%	16%
B1	5	5	2.7	0.25	1	5	60%	65%	15%
B2	5	5	2.7	0.5	1	5	60%	69%	28%
C	4	3	2.5	0.25	1	5	55%	68%	30%
RT1	4	2	2	0.25	1	1	6%	26%	22%
RT2	4	2	2	0.5	2	3	55%	68%	37%
RT3	7	2	4	0.5	2	2	55%	58%	10%

***N**: total number of clusters; **n**: maximum number of kernels per cluster; **DS**: total data size per iteration (input data + intermediate results + final results) in KB ; **RF**: reuse context factor; **FB**: One frame buffer set size in KB;**RAM**: RC-RAM size in KB. **E1, E2, E1-2**: Data Schedulers relative energy improvement*

Table 1. Experimental Results

sizes as shown, A1-A2 or B1-B2 or RT1-RT2. RC-RAM size is always smaller than FB size because RC-RAM is hierarchically the lowest memory. A1 and A2 have few data stored in RC-RAM since the majority of data are not read many times. The increase in FB size achieves a better reduction in energy budget by avoiding context transfers. B1 and B2 have many data stored in RC-RAM.

For B1 all the most accessed data cannot be stored in RC-RAM. Therefore an increase in RC-RAM size, as B2 shows, achieves a better E2 result. RT1 and RT2 represent simple image ray-tracing, and for this case an increase in RC-RAM does not improve energy performance because all the most accessed data fit into RC-RAM. However an increase in FB and RC-RAMs allows context reuse. Although the increase in memory is more energy consuming this effect is worthwhile due to greater data and context reuse. C stands for an intermediate example context reuse between A and B. RT3 stands for a more complicated image ray-tracing, which increases data size and number of kernels. In all cases the Data Scheduler achieves better energy results than the previous version, as E2 and E1-2 show, due to the current Data Scheduler improving FB and RC-RAMs usage, minimizing energy consumption.

7. Conclusions

In this paper we have presented a new technique to improve data scheduling for multi-context reconfigurable architectures. It stores the most frequently accessed data in the on-chip memories (FB or RC-RAMs) to minimize data and context transfers, reducing the energy budget.

We have developed a method to efficiently store data and results in RC-RAMs. The Data Scheduler decides which data or results have to be loaded into these internal memories to reduce energy consumption.

The Data Scheduler allows data and results reuse within a cluster, minimizing the memory space required by cluster execution. It allows the reuse of data and results among clusters if the FB has sufficient free space. It chooses the shared data or results to be kept within FB, allowing further reductions in transfers to/from the external memory.

The Data Scheduler maximizes the available free space in the FB. This allows the number of consecutive iterations (RF) to increase and as a consequence, kernel contexts are reused during these iterations, reducing context transfers.

The experimental results demonstrate the effectiveness of this technique in reducing the execution time and power budget compared to previous data schedulers.

Future work will address data management within a kernel, as well as a detailed study of optimal memory sizes for applications.

References:

[1] S. Brown, J. Rose. “Architecture of FPGAs and CLPDs: A Tutorial,” IEEE Design and Test of Computer, Vol. 13, No 2, pp. 42-57, 1996.

[2] E. Tau, D. Chen, I. Eslick, J. Brown and A. De Hon, “A First Generation DPGA Implementation”, FDP’95, Canadian Workshop of Field-Programmable Devices, May 29-Jun 1, 1995.

[3] H. Singh, M. Lee, G. Lu, F. Kurdahi, et al, - “MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications”, IEEE Transactions on Computers, pp. 465-481, Vol. 49, No. 5, May, 2000.

[4] M. Meiβner, S.Grimm, W. Straβer, et al, “Parallel Volume Rendering on a Single-Chip SIMD Architercture”, IEEE Symposium on Parallel and Large-Data Visuallization and Graphics, San Diego, Ca, October, 2001.

[5] S. Sohoni, R. Min, Z. Xu, Y. Hu. “A study of memory system performance of multimedia applications”, SIGMETRICS Performance 2001, pp. 206-215.

[6] R. Maestre, F. Kurdahi, et al. “Kernel Scheduling in Reconfigurables Architectures”, DATE Proceedings pp 90-96, 1999.

[7] R. Maestre, F. J. Kurdahi, M. Fernandez, et al. "Configuration Management in Multi-Context Reconfigurable Systems for Simultaneous Performance and Power Optimizations", ISSS Proceedings, pp. 107-113, 20-22 September 2000.

[8] M. Sanchez-Elez, M. Fernández, et al. “A Complete Data Scheduler for Multi-Context Reconfigurable Architectures” DATE Proceedings, Paris, France, pp. 547-552 March 2002.

[9] Panda, P. R., Catthoor, F., Dutt, N. D., Danckaert et al. “Data and Memory Optimization Techniques for Embedded Systems”, ACM Transactions on Design Automation of Electronic Systems. Vol. 6, Iss. 2. Apr. 2001

[10] S. Wuytack, F. Catthoor, et al. “Global communication and memory optimizing transformations for low power design,” in Proc. IWLPD-94: Int. Workshop on Low Power Design, Napa Valley, CA, Apr. 1994, pp. 203–208.

[11] M. Kaul, R. Vemuri, et al. “An Automated Temporal Partitioning and Loop Fission approach for FPGA based reconfigurable synthesis of DSP applications” Proc. 36th Design automation conference, 1999, Pages 616 - 622

[12] A. Jantsch, P. Ellervee, A. Hemani, et al. “Hardaware-Software Partitioning and Minimizing Memory Interface Traffic”, DATE Proceedings, 1994, Pages 226 – 231.

[13] P.R. Wilson, M.S. Johnstone, M Neely, and D Boles “Dynamic Storage Application A Survey and Critical Review” IWMM 1995: 1-116

[14] M. B. Kamble and K. Ghose, “Analytical Energy Dissipation Models for Low Power Caches”, ACM/IEEE International Symposium on Microarchitecture, pp 184-193, Research triangle Park, NC, December 1997.

[15] J. Kin, M. Gupta and W. H. Mangione-Smith, “The Filter Cache: An Energy Efficient Memory Structure”, MICRO-97: International Symposium on Microarchitecture, pp 184-193, Research Triangle Park, NC, December 1997.

[16] W. Cheng, M. Pedram, “Low Power Techniques for Address Encoding and Memory Allocation”, Procc. of Asia and South Pacific DAC, Jan. 2001, pp. 245-250.

[17] P. Van Oostende, G. Van Wauve, “Low Power design: a gated-clock strategy” Low Power Workshop, Ulm, Germany, Sep.’94

[18] A. Glassner. “An Introduction to Ray Tracing”. Academic Press, 1989