AN OPTIMIZATION METHOD OF DMA TRANSFER FOR A GENERAL PURPOSE RECONFIGURABLE MACHINE

Sayaka Shida, Yuichiro Shibata, Kiyoshi Oguri

Dept. of Computer and Information Sciences,
Nagasaki University, Japan
{shida,shibata,oguri}@pca.cis.nagasaki-u.ac.jp

Duncan A. Buell

Dept. of Computer Science and Engineering,
University of South Caroline, USA
buell@cse.sc.edu

ABSTRACT

DMA transfer between a CPU and an FPGA often becomes a bottleneck of current reconfigurable machines. The DMA transfer of the machines like SRC-6 supports streaming processing with on-board memory interleaving, but as a preprocessing of the interleaving, the CPU must reorder the data for applications with severe FPGA resource constraints. This paper empirically evaluates this overhead to reveal the trade-off point. The results show that a speedup is achieved by interleaved streaming DMA when 150KB or lower data strings are transferred.

1. INTRODUCTION

The rapid progress in speed and integration density of field programmable gate arrays (FPGAs) has made FPGA-based high-performance general computing systems possible. Many commercial high-end machines equipped with a high-speed CPU and FPGAs are now available and have been used in performance-centric application fields including scientific simulation and cryptography[1][2][3].

While vast flexibility of FPGAs provides opportunities of efficient application processing, there are also pitfalls that may lead to inefficient results since performance of reconfigurable machines tends to depend strongly on skills of application developers[4]. In many cases, a performance bottleneck for these machines seem to be DMA transfer between a CPU and FPGAs[5][6]. To mitigate this problem, various architectural techniques to reduce and hide the transfer overhead are utilized, but another performance concern is memory access between FPGAs and their local memory.

To cope with this problem, we propose to reorder the data on the main memory using a CPU in advance of DMA transfer, so that the FPGAs take advantage of data parallelism on the local memory banks. Although the reordering of the data is obviously a waste of the CPU time, it will pay if enough parallelism is extracted on the FPGAs. In this paper, we empirically evaluate this idea on the SRC-6[7] and analyze a trade-off point.

2. SRC-6 ARCHITECTURE

2.1. System Organization

As Fig. 1 shows, the SRC-6 is composed of Xeon 2.8 GHz microprocessors with 1 MB L2 cache, reconfigurable processors called MAP, and Common Memory. They are connected each other with a crossbar switching fabric called Hi-Bar Switch. The MAP is composed of three Xilinx Virtex II XC2V6000 FPGAs. Among the three FPGAs in the MAP, one is dedicated to controller and the others (FPGA1 and FPGA2) are used for applications. The MAP has six banks of 4 MB On Board Memory (OBM), where data operated by FPGAs are stored. An FPGA can access multiple banks at the same time because they are connected with individual interactive ports. OBM is shared by the two FPGAs and a DMA controller and is used for data exchange between them. Arbitration of access to the OBM can be explicitly controlled by users.

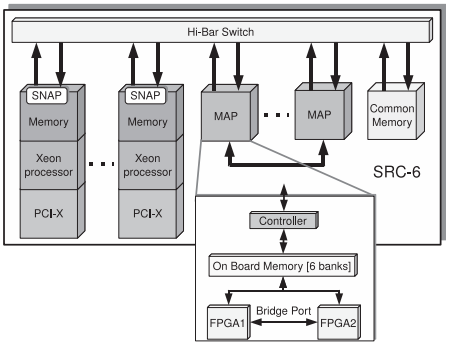

Fig. 1. Structure of the SRC-6 and MAP.

The SRC-6 has a unique compiler which generates a relocatable object file from C and FORTRAN source files. The compiler translates the loops specified by users into hardware circuits to be configured on FPGAs. The compiler automatically analyzes structure of the loops and generates hardware that processes the repetition in a pipelined manner.

2.2. Streaming DMA

SRC-6 supports two styles of DMA transfer; regular DMA and streaming DMA. In the regular DMA, FPGAs have to wait for the completion of the transfer before starting the operation. On the other hand, transfer and operation can be parallelized with the streaming DMA. In addition, the streaming DMA has two modes; single streaming DMA and dual streaming DMA.

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

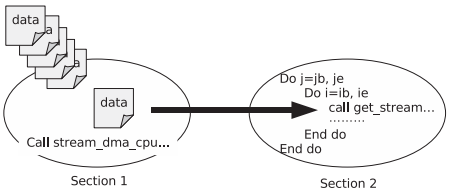

Fig. 2. Single streaming DMA.

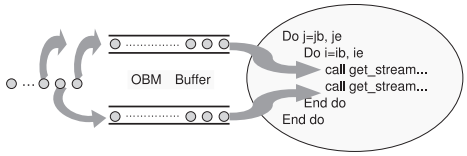

Fig. 3. Dual streaming DMA.

Fig. 2 shows the concept of the inbound single streaming DMA. In SRC-6, a pair of system functions is available for streaming DMA. These functions create separated *sections*, which are operated in parallel with independent control on the FPGA. In Section 1, data is continuously transferred through an OBM bank. In Section 2, data are fetched from the stream and used for calculation. Thus, the OBM bank is used as a buffer that absorbs a speed gap between the DMA transfer and the FPGA operation.

In the dual streaming DMA transfer, two banks of the OBM are used as buffers as shown in Fig. 3. A contiguous series of data on the main memory is alternately transferred to the specified two OBM banks, so that the FPGA can fetch two elements of the transferred data at the same time. This automatically interleaved transfer facility effectively enables the FPGA to extract more data parallelism in applications.

3. DMA OPTIMIZATION STRATEGIES

Fig. 4 conceptually shows how execution time of a function implemented on an FPGA is changed by DMA transfer methods. Here, we assume the function needs N inbound data strings and one outbound data string to be transferred. For the sake of simplicity, we will only focus on a DMA method for the inbound data transfer.

The most basic and simplest strategy is to use regular DMA only as shown in Fig. 4 (a). The execution on the FPGA starts after N strings of inbound data are transferred. Then, the outbound data can be transferred after the execution is completed. By using the single streaming DMA for the last inbound data string, the total time can be reduced as shown in Fig. 4 (b). Since the time required to transfer the last data string is overlapped with the FPGA execution, the FPGA can start execution after $(N-1)$ data strings are transferred. Note that the SRC-6 DMA controller does not allow to initiate multiple DMA transfer at the same time, since only one port is provided between the CPU and the FPGA. That is, streaming DMA transfer can not be overlapped with other DMA transfer.

Using the dual streaming DMA makes an impact on the

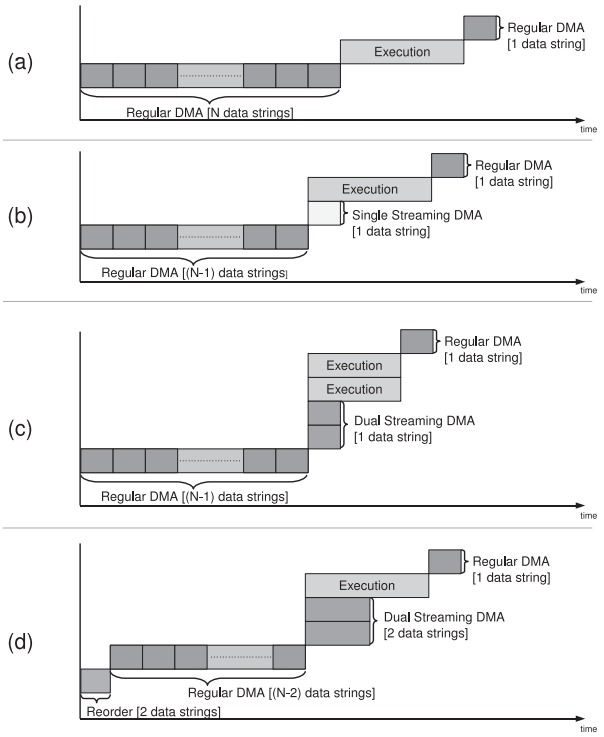

Fig. 4. DMA optimization strategies.

FPGA execution time as shown in Fig. 4 (c). Since the DMA controller interleaves the last data string between two OBM banks, the FPGA can boost the operation by dual fetch of the data. The time that the FPGA must wait before launching operation remains to be $(N-1)$-times transfer time. This method is obviously effective to improve total performance, but the circuit size on the FPGA is also increased so as to extract data parallelism. Therefore, this method is often difficult to adopt for functions that require relatively large amount of hardware.

Our proposed method shown in Fig. 4 (d) is able to reduce the DMA transfer time even under a sever FPGA resource constraint. The basic idea is to transfer two individual data string to two OBM banks respectively using dual streaming DMA in stead of interleaving a single data string into the two banks. Thus, the circuit required for the FPGA operation is almost the same as that for Fig. 4 (b). Moreover, the FPGA can start the operation after $(N-2)$ data strings are transferred, since the last two inbound strings are transferred overlapped with the FPGA operation. However, unlike ordinary dual streaming DMA, the two data strings must be interleaved on the main memory before the transfer. Therefore, the CPU have to reorder the two strings in advance and this could be a considerable overhead especially when the string size is large. We call this method "dual streaming DMA with reordering" in the following.

Here, we explain the methods, taking the barotropic operator function in an ocean circulation model called the Parallel Ocean Program (POP) as an example. The main al-

```
Initial values AX(:,:,bid) = 0.0_8
do j=jb, je
  do i=ib, ie
    AX(i,j,bid) = A0 (i  ,j  ,bid) * X(i  ,j  ,bid) + &
                  AN (i  ,j  ,bid) * X(i  ,j+1,bid) + &
                  AN (i  ,j-1,bid) * X(i  ,j-1,bid) + &
                  AE (i  ,j  ,bid) * X(i+1,j  ,bid) + &
                  AE (i-1,j  ,bid) * X(i-1,j  ,bid) + &
                  ANE(i  ,j  ,bid) * X(i+1,j+1,bid) + &
                  ANE(i  ,j-1,bid) * X(i+1,j-1,bid) + &
                  ANE(i-1,j  ,bid) * X(i-1,j+1,bid) + &
                  ANE(i-1,j-1,bid) * X(i-1,j-1,bid)
  end do
end do
```

Fig. 5. Algorithm of the barotropic operator function.

gorithm of the function is shown in Fig. 5. This function requires a total of five inbound arrays ($A0$, AN, AE, ANE, and X), and one outbound array (AX). Our previous work has revealed that the function requires more than 60% of the FPGA slices and thus use of simple interleaved streaming DMA is infeasible due to the resource constraint. One optimization strategy for inbound DMA transfer is to use the single streaming DMA for the array $A0$. The other strategy is to use the dual streaming DMA with reordering for the arrays $A0$ and AE. Before transferring them, the CPU alternates them on the main memory to form a double sized single array. Then transferring the double size array with the dual streaming DMA isolates again $A0$ and AE in individual two OBM banks. Since use of streaming DMA hardly affect the FPGA operation time, an essential trade-off will arise between the following two strategies; (1) use of the regular DMA and the single streaming DMA, and (2) use of the dual streaming DMA after reordering on the CPU.

4. EVALUATION

4.1. Streaming DMA and reordering

Fig. 6 shows the execution time of the regular DMA and reordering on SRC-6, and Fig. 7 gives a closeup view of Fig. 6. Here, we changed the string size from 1 Byte to 4 MB, which is the size of the OBM bank. The time required for the regular DMA is almost proportional to the amount of data transferred. On the other hand, the time required for reordering is not proportional until the string size reaches around 500 KB. This is attributed to the fact that the two strings of data to be reordered can not be stored in the 1 MB L2 cache when the string size exceeds 512 KB.

As Fig. 7 shows, the streaming DMA with reordering is superior to the regular DMA when the string size is smaller than 150 KB. In many scientific applications including POP, users can chose desired computation granularity of function calls, that is, how much data is processed by a single call of the function. Although this does not make a large impact on execution time on ordinary computers, choosing the string size of 150 KB or smaller would bring about a positive effect of the dual streaming DMA with reordering on SRC-6.

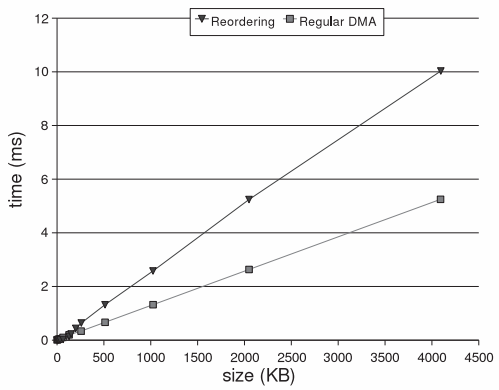

Fig. 6. DMA transfer and reordering (1).

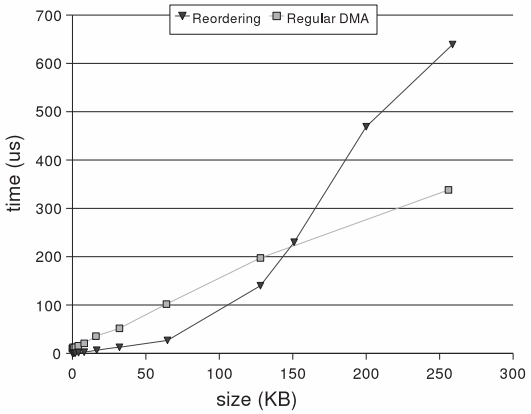

Fig. 7. DMA transfer and reordering (2).

4.2. Barotropic operator function

Based on the results of Sec. 4.1, we implemented the barotropic operator function using dual streaming DMA with reordering. The six implementation patterns summarized in Table 1 were compared to evaluate the effect of the DMA methods. The measured values of execution time on SRC-6 and software execution time on a 2.8 GHz Xeon processor are presented in Fig. 8. The figure also shows a breakdown of the execution time on SRC-6. Note that the overlapped time of FPGA execution and DMA transfer is counted as "execution time" in this breakdown. The evaluation results show that Arch. 5 and Arch. 6, which use the dual streaming DMA with reordering, effectively reduced the DMA transfer time and boosted the total execution time. Arch. 6, which uses the dual streaming with reordering for both inbound and outbound transfer, marked the best performance, achieving 1.37 times performance improvement compared to the Xeon processor. The time required to reorder strings accounted for only 2.23 % of the total execution time.

4.3. Grad function

Next we evaluated streaming DMA with the grad function in POP, which calculates the gradient of an energy field at each

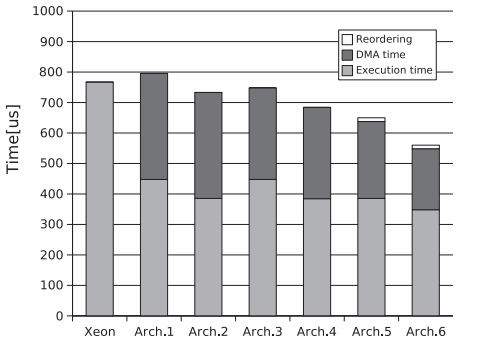

Fig. 8. Performance results of using Dual streaming DMA.

Table 1. Evaluated patterns.

	Internal MEM	Streaming DMA		
		Single		Dual with reordering
		Inbound	Outbound	Inbound
Arch. 1	not used	not used	not used	not used
Arch. 2	used	not used	not used	not used
Arch. 3	not used	used	not used	not used
Arch. 4	used	used	not used	not used
Arch. 5	used	not used	not used	used
Arch. 6	used	not used	used	used

time step. In this implementation, two inbound arrays are transferred with streaming DMA while the others are transferred with regular DMA. The grad function mainly consists of three loops of processing and the SRC compiler converts each loop description into individual pipelined arithmetic hardware circuits. The streaming DMA was used for the second loop. Unlike the barotropic operator mentioned in the previous section, the grad function uses only one FPGA since the required hardware amount is not so large.

Fig. 9 shows the execution results of the grad function on the SRC-6 and the Xeon processor. While the grad function utilizes some 32-bit integer arrays, SRC-6 only supports 64-bit data transfer for DMA. Therefore, we implemented data conversion function on the CPU which extends 32-bit integer data to 64 bits in advance of DMA transfer. Fig. 9 also shows this process as "extension" in the breakdown, but the actual time was less than a microsecond and was almost negligible. Also, the data reordering accounted for only up to 1.82 % of the total execution time. Compared to software execution on the Xeon, the regular DMA implementation, the single streaming DMA implementation, and the dual streaming DMA with reordering implementation achieve 1.31, 1.37, and 1.46 times speedup, respectively. Focusing only on FPGA execution time, slight degradation arose as the traffic of the streaming DMA was increased. However, the effect of the streaming DMA overcame this overhead, resulting in the performance improvement.

Since the implementation in Fig. 9 uses the streaming DMA only for inbound transfer, the performance would be further improved by using streaming DMA for outbound

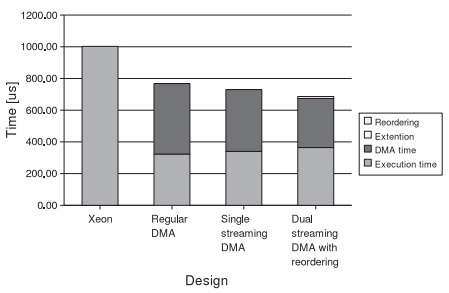

Fig. 9. Performance results of the grad function.

transfer as it was in the barotropic operator function. Moreover, this implementation utilized approximately only 50 % slices of single FPGA. Therefore another resource-consuming techniques such as circuit parallelization and use of more FPGA internal memory could also be applied to reduce the number of OBM access.

5. CONCLUSION AND FUTURE WORK

Aiming at making the best use of the interleaving streaming DMA facility even under a rigid FPGA resource constraint, we proposed to make the CPU reorder the data strings to be transferred on the main memory and empirically evaluated the overhead using two functions in the POP. The evaluation results showed the proposed method was superior to single streaming DMA when the size of transferred data string does not exceed approximately 150 KB. As our future work, we will try to lay an automatic optimization framework for DMA transfer. Scheduling of DMA traffic and communication between the two FPGAs is also our challenging task.

6. REFERENCES

[1] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The promise of high-performance reconfigurable computing," *IEEE Computer*, vol. 41, no. 2, pp. 69–76, Feb. 2008.

[2] R. Scrofano, M. Gokhale, F. Trouw, and V. K. Prasanna, "A hardware/software approach to molecular dynamics on reconfigurable computers," *Proc. FCCM*, pp. 23–34, 2006.

[3] M. C. Smith, J. S. Vetter, and S. R. Alam, "Scientific computing beyond CPUs: FPGA implementations of common scientific kernels," *Proc. MAPLD*, 2005.

[4] O. Mencer, "Computing with FPGAs," *Proc. COOL Chips*, pp. 327–343, 2006.

[5] M. B. Gokhale, C. D. Rickett, J. L. Tripp, and C. H. Hsu, "Promises and pitfalls of reconfigurable supercomputing," *Proc. ERSA*, pp. 11–20, June 2006.

[6] S. Shida, Y. Shibata, K. Oguri, and D. A. Buell, "Implementation of a barotropic operator for ocean model simulation us ing a reconfigurable machine," *Proc. FPL*, pp. 589–592, Aug. 2007.

[7] SRC Computers, Inc., *MAPstation*, 2005.