A HIERARCHICALLY-CONTROLLED SIMD MACHINE FOR 2D DCT ON FPGAs*

Xizhen Xu and Sotirios G. Ziavras

Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102, USA
Email: ziavras@njit.edu

ABSTRACT

Platform FPGA devices are an attractive option for implementing parallel systems on a single chip that can be used as a coprocessor. However, the substantial communication overhead between the host workstation and the FPGAs is a major performance bottleneck. Also, mapping an application to FPGAs still remains a daunting job. To address these problems, this paper describes a Hierarchical SIMD (H-SIMD) machine design with its codesign of a Hierarchical Instruction Set Architecture (HISA). Our proposed H-SIMD design uses an FPGA controller to facilitate ease of program development. It also employs a memory switching scheme to overlap communications with computations as much as possible. The 2-dimensional Discrete Cosine Transform (DCT2 or 2D-DCT) is enlisted to show the effectiveness of the H-SIMD machine.

I. INTRODUCTION

Platform FPGAs have emerged as a new powerful SOC (System-On-Chip) computing paradigm. They are usually customized to implement computationally expensive datapaths when coupled with a host [1] [2]. Usually, the host may be a workstation while the FPGA resources form a coprocessor that communicates with the host via an I/O interface. This mechanism yields greater parallelism at run time but at the expense of higher communication overheads and a daunting algorithm mapping process.

The SIMD mode of computation is suitable for data-intensive applications. Assuming a host-FPGA system, we configure the FPGA chip as an application-specific SIMD coprocessor controlled by the host. Based on HISA, application tasks are partitioned into three layers: host, FPGA, and nano-processor layers, in decreasing order of task granularity. The host layer, at the top of the H-SIMD machine, is implemented in the workstation by a high-level language. More specifically, similar to the approach for PC clusters in [5] we suggest that an effective ISA be developed at the host layer for each application domain. Frequently used instructions in that domain should belong to this ISA. The invocation of these instructions is actually implemented as function calls at the host layer. These functions can take advantage of available IP cores so that the design time can be significantly reduced for the complex platform FPGA system. Existing design methodologies focus on logic synthesis without considering the algorithm structure [2] [8]. Our H-SIMD machine approach takes into account algorithm synthesis at the host layer and logic synthesis at the nano-processor layer, respectively. Another major advantage of the H-SIMD machine is the employment of a memory switching scheme for data loads/stores involving the host and the FPGAs. Switching between pairs of data memory banks overlaps operand communications with computations, thus hiding communication overheads to improve performance.

The remainder of this paper is organized as follows. Section II presents our H-SIMD machine architecture. Section III includes a detailed design of HISA and its workload balancing scheme for DCT2. Section IV contains implementation results on a Xilinx Virtex II 6000 FPGA and a comparative study with a 2.0GHz Pentium processor. Section V concludes our work.

II. MULTI-LAYERED H-SIMD MACHINE

A. H-SIMD Architecture

The H-SIMD control hierarchy is composed of three layers, as shown in Fig. 1. It comprises the host controller (HC), the FPGA controller (FC) and the nano-processor controllers (NPCs). HC lies in the host machine and controls the FPGA chip. Inside the FPGA chip, an FC is designed not only to run all the on-chip NPCs in the SIMD mode but also to decompose the coarse-grain host instructions

* This work was supported in part by the US Department of Energy under grant DE-FG02-03CH11171.

into fine-grain nano-processor instructions which run in parallel. The NPCs control the execution of nano-processor-level code. HISA logically partitions the application into the HC, FC and NPC layers that can be handled efficiently at run time to balance the pipeline flow from the host down to the on-chip NPCs. The FC converts application program functions represented by coarse-grain host SIMD instructions (HSIs) into sequences of medium-grain FPGA SIMD instructions (FSIs). Then, the NPCs decode the received FSIs into machine-level nano-processor instructions (NPIs) for final code execution. This way, task-level HSIs are executed in parallel by the FPGA-level SIMD machine and the applications fully exploit the FPGA parallel datapath resources. The HISA instruction set is tailored to the specific application in an effort to yield high performance. The developed HISA ISA for DCT2 is presented in Section III.

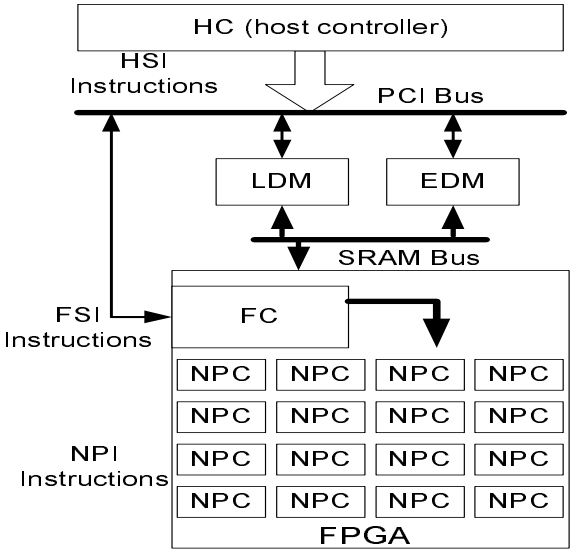

Fig. 1. The H-SIMD architecture and its HISA hierarchies.

B. Memory Switching Schemes

The communication overhead between the host and the FPGA chip can be substantial primarily due to the non-preemptive nature of the operating system on the host. Based on tests in our laboratory, the one-time interrupt latency for a Windows-XP installed Dell Precision 650 host workstation running the PCI bus at 133MHz is no less than 1.5 ms. This penalty is intolerable in high-performance computing because, for example, the 64x64-point DCT2 takes about 350 us on a single arithmetic unit running at 100 MHz (which is within the range of current FPGA technology) for the Feig & Winograd algorithm [3]. If the host frequently intervenes in FPGA operations, the speedup benefits gained from the parallel FPGA implementation can be significantly reduced or even removed. Our data prefetching scheme involving memory switching is designed for the H-SIMD machine to delicately overlap host communications with FPGA computations as much as possible. Data flowing from the HC is directed into the high-speed SRAM banks on the FPGA board. The HC-level memory switching scheme is shown in Fig. 2. The SRAM banks are organized into two functional memory units: the execution data memory (EDM) and the loaded data memory (LDM). They alternate between HSI execution and HSI data load/retrieval. The nano-processors form the execution units in the H-SIMD machine datapath. Their functionality can be customized to applications, such as DCT2, matrix multiplication (MM) or FFT. The register files of each nano-processor include the load register file (LRF) and the execution register file (ERF) shown in Fig. 3. Both register files work in a "memory" switching capacity, similarly to LDM and EDM. The interrupt request/response latency between the FCs and their own nano-processors is one cycle only as opposed to the tens of thousands of cycles between the host and the FPGAs, thus gaining a performance boost.

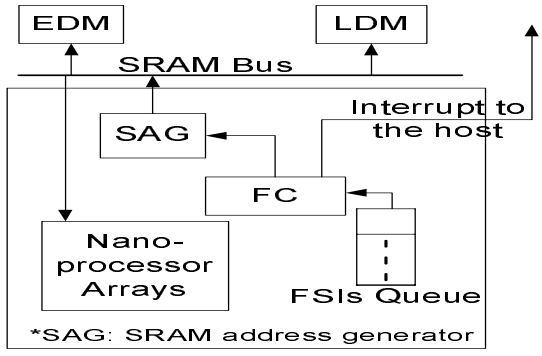

Fig. 2. HC-level memory switching in H-SIMD.

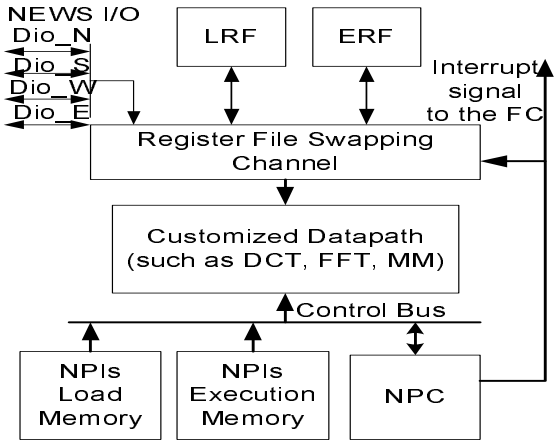

Fig. 3. Nano-processor datapath and control unit.

III. HISA DESIGN AND TASK PARTITIONING FOR DCT2

A. HISA ISA for DCT2

DCT2 is a technique widely used in image processing and adopted by several compression standards such as H.261, H.263 and MPEG-4. The basis vectors for DCT2 can be pre-computed and stored in the on-chip memory of the FPGA for better performance. We assume large images divided into blocks of size 8x8. These blocks are transformed via an 8x8 forward DCT2. Here, we show the HISA tailoring for DCT2 with the H-SIMD design methodology.

Three HSIs are needed for DCT2. They are programmed with the software API library for our target Annapolis FPGA Wildstar II board [4]. 1) ***host_dct2_load(LDM, N_h)***: This HSI loads the FPGA's LDM memory with an input matrix of size N_hxN_h via the host PCI bus; 2) ***host_dct2(H_A, H_B, N_h)***: H_A is the input image of size N_hxN_h and H_B = dct2(H_A); 3) ***host_dct2_retrieve(LDM, N_h)***: The final results are stored in the switched-out LDM and can be retrieved by the host through the PCI bus. The FSIs in the middle of the HISA hierarchy run on the FPGA hardware. Assume that there are p nano-processors in each FPGA chip. There are two FSIs: 1) ***FPGA_dct2_load(F_a, LRFi, N_f)***: FC will execute this instruction by loading each nano-processor's LRF with a block matrix of size N_f x N_f from matrix F_a of size N_f^2 x p. F_a is the starting address of the input; 2) ***FPGA_dct2(F_a, F_b, N_f)***: matrix F_b of size N_f^2 x p is stored into EDM and produced by applying DCT2 to the input matrix F_a. The NPIs constitute only one customized instruction: 8x8 point DCT2, ***NP_dct2(R_a, N_f)***. Its datapath is produced by the Xilinx CoreGenerator [6] and targets image blocks of size 8x8, i.e., N_f = 8. The input data in our experiments is stored in a vector R_a of size 64 represented as 8-bit signed integer numbers. R_a is the starting address of the vector residing in the ERF memory of each nano-processor.

B. Analysis of Task Partitioning for DCT2

Basically, there are two interfaces in H-SIMD: the PCI bus with a bandwidth B_{pci} =133MHz x 64 bits and the SRAM bus with a bandwidth B_{sram} =960Mbytes/s. In the case of DCT2 on an N_hxN_h matrix, let us assume that $N_h^2/64$ block matrices of size 8x8 are uniformly distributed among p nano-processors. Our HDL simulation results show the CoreGenerator-produced 8x8-point DCT2 with latency L_{dct2}=117cycles. For the total computation time $T_{comp\_hosts\_dct2}$ of ***host_dct2(H_A, H_B, N_h)***,

$$T_{comp\_hosts\_dct2} > N_h^2/(N_f^2*p)*T_{NP\_dct2\_8x8}*N_{frame} \quad (1)$$

where $T_{NP\_dct2\_8x8}$ is the nano-processor execution time of DCT2 on a block matrix of size 8 x 8 and N_{frame} is the number of input image frames. On the other hand, the communication time T_{i/o_pci} of ***host_dct2_load/retrieve*** depends on the available PCI I/O bandwidth and the interrupt latency T_{host_int},

$$T_{i/o\_pci} = 2*b*N_h^2/B_{pci}*N_{frame} + T_{host\_int} \quad (2)$$

where b is the width in bits of each transaction. Pipeline balancing is guaranteed if $T_{comp\_hosts\_dct2}$ is greater than or equal to T_{i/o_pci}, and the computations fully overlap I/O communications. For p=2, 4 or 8, B_{pci}=133MHz x 64bits, T_{host_int} =1.8ms, b=8, N_{frame}=6 and τ=8ns for 8-bit signed integer numbers, the simulation results in Fig. 4 show that the computation time varies with two parameters: N_h and p. By tweaking these parameters, the computation time grows faster than the I/O communication time.

For effective nano-processor-level memory switching, the execution time $T_{NP\_dct2\_8x8}$ of an 8 x 8-point DCT2 ***NP_dct2(R_a, 8)*** must be greater than p times the register file reference time T_{NP_load}. In fact, this condition can be easily met when B_{sram}=960Mbytes/s, p=8, τ=8ns.

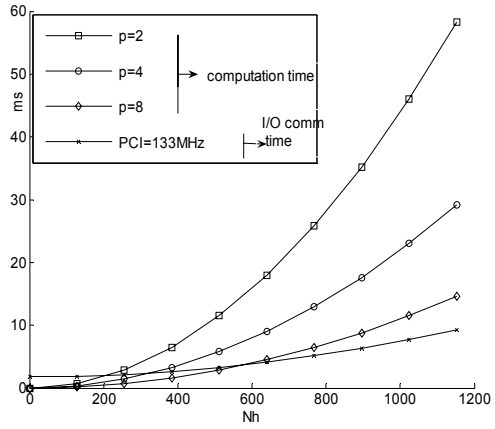

Fig. 4. Computation vs I/O communication times as a function of N_h and p.

IV. DCT2: IMPLEMENTATION AND TEST RESULTS

We implemented H-SIMD on a host PC workstation and an Annapolis FPGA board [4]. The latter contains two Xilinx Virtex-II 6000 FPGAs. There are six Samsung 512kx36 SRAM banks around each FPGA. The host is assigned the DCT2 transform $H_{dct2}(N_h,N_h)$=dct2(H_A), where H_A and H_{dct2} are matrices of size N_hxN_h.

The DCT2 cores were generated by the Xilinx CoreGenerator6.2 [6] and were configured for 8-bit input data, 16-bit coefficients, internal data and output results. After Xilinx ISE6.2 Place&Route, eight nano-processors running at 126MHz fit in

each FPGA. DCT2 was tested on matrices of size 128x128, 256x256, 512x512 and 1024x1024. The timing results break down into the interrupt overhead, PCI reference time and DCT2 computation time, as shown in Fig. 5. When N_h is set to 128, the frequent interrupts to the host contribute excessively to the performance penalty. When N_h is 256 or 512, the computation time does not increase long enough to fully overlap the sum of the host interrupt and PCI-SRAM sequential reference overheads. If N_h =1024, the designed overlap scheme is so good that the interrupt and communication overheads are hidden and all the nano-processors work in parallel.

Let us denote by t_B the DCT2 time for one HSI-level N_h xN_h matrix. If the frame size is *NxN*, then the time for DCT2 on a frame is $t_{frame}=N^2/N_h^2*t_B$ and the frame rate is $R = 1/t_{frame}$. We show frame rates for various frames and HSI matrix blocks in Table I. N_h=1024 scores the highest frame rate. This is due to the combination of HISA and the memory overlap scheme.

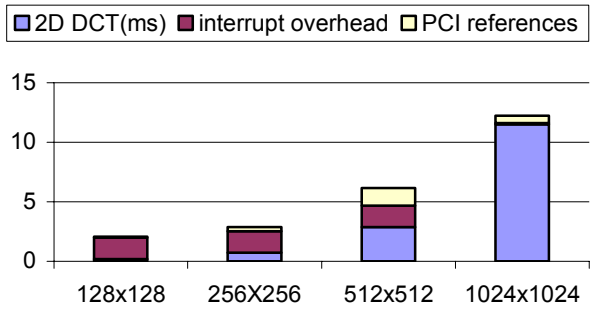

Fig. 5. Execution time breakdown of DCT2 for six input frames.

Table II compares the performance of our implementation on one Virtex II 6000 FPGA to the performance of a 2GHz Pentium processor as presented in [7]. If MMX and streaming SIMD instructions are enabled on the latter, H-SIMD yields a speedup of about 6%~18%; otherwise, the speedup is about four. This shows the effectiveness of the H-SIMD architecture on data-intensive applications.

Table I. Frame rates for various frame and matrix block sizes.

HSI matrix size / Frame Size	128	256	512	1024
1024x1024	45	129	243	516
2048x2048	11	32	60	129
4096x4096	3	8	15	32
8192x8192	0.7	2	3	8

Table II. Performance comparison for the 1024x1024-point DCT2.

		Execution time (ms)
2GHz Pentium	**Integer implementation**	7.9
	MMX instructions	2.29
	MMX and streaming SIMD	2.05
H-SIMD machine		1.93

V. CONCLUSIONS

Our multi-layered H-SIMD machine paired with an appropriate multi-layered HISA software codesign yields high performance on platform FPGAs for data-intensive applications. Our current implementation of DCT2 demonstrates the effectiveness of H-SIMD on applications having enough parallelism. Our multi-layered design approach is not limited to DCT2. It can be applied to other applications as well which are rich in data parallelism.

REFERENCES

[1] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: a survey," *Journal of VLSI Signal Processing*, Vol. 28, 2001, pp. 7-27.

[2] M. J. Wirthlin, B.L. Hutchings and K.L. Gilson, "The nano processor: a low resource reconfigurable processor," *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, April 1994, pp. 23 – 30.

[3] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform," *IEEE Trans. Signal Processing*, Vol. 40, No. 9, Sep. 1992, pp. 2174-2193.

[4] Wildstar II Reference Manual, *Annapolis Micro Systems*, Inc.. Annapolis, MD, 2004.

[5] D. Jin and S. Ziavras, "A super-programming approach for mining association rules in parallel on PC clusters," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 9, Sept. 2004, pp. 783-794.

[6] http://www.xilinx.com/ipcenter.

[7] A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX™ Instructions. *Intel Corp.* Order #742474-001, Application Note AP-528.

[8] K. Bondalapati and V.K. Prasanna, "Reconfigurable computing systems," *Proceedings of the IEEE,*Vol 90, Issue 7, July 2002, pp:1201-1217.