Improve Memory Access for Achieving Both Performance and Energy Efficiencies on Heterogeneous Systems

Hongyuan Ding and MiaoqingHuang
Department of Computer Science and Computer Engineering
University of Arkansas
Fayetteville, AR 72701, USA
Email: {hyding,mqhuang}@uark.edu

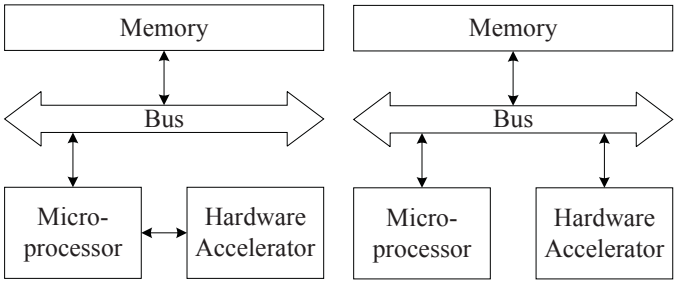
***Abstract*—Hardware accelerators are capable of achieving significant performance improvement for many applications. In this work we demonstrate that it is critical to provide sufficient memory access bandwidth for accelerators to improve the performance and reduce energy consumption. We use the scale-invariant feature transform (SIFT) algorithm as a case study in which three bottleneck stages are accelerated on hardware logic. Based on different memory access patterns of SIFT algorithms, two different approaches are designed to accelerate different functions in SIFT on the Xilinx Zynq-7045 device. In the first approach, convolution is accelerated by designing fully customized hardware accelerator. On top of it, three interfacing methods are analyzed. In the second approach, a distributed multi-processor hardware system with its programming model is built to handle inconsecutive memory accesses. Furthermore, the last level cache (LLC) on the host processor is shared by all slaves to achieve better performance. Experiment results on the Zynq-7045 device show that the hybrid design in which two approaches are combined can achieve ∼10 times and better improvement for both performance improvement and energy reduction compared with the pure software implementation for the convolution stage and the SIFT algorithm, respectively.**

I. Introduction

The tradeoff between performance and power consumption has been carefully evaluated for many embedded systems and hand-held devices [1]–[3]. Typically, the more powerful the microprocessor is, the more power it needs to consume. In order to achieve both the performance and energy efficiency, hardware accelerators are frequently adopted in systems with limited power budget [4], [5].

Hardware accelerators are typically used as co-processors in computer systems. There are two configurations for hardware accelerators to access data as shown in Figure 1. In the first configuration shown in Figure 1(a), the hardware accelerator does not access the memory directly; instead, it performs as a slave of the microprocessor. The microprocessor reads the source data from the memory and feeds them to the accelerator, which processes them and returns the result back to the microprocessor [6], [7]. In the second configuration shown in Figure 1(b), the hardware accelerator is directly connected to the bus to access the memory. Typically a DMA module needs to be integrated to facilitate the accelerator for reading source data from and writing the result data back to the memory [8].

Memory has been playing an increasing important role in computer systems. Due to the growing disparity of speed between the processor and the memory, i.e., the memory wall, it is proposed to use Concurrent Average Memory Access Time (C-AMAT) as an accurate metric for modern memory systems [9]. Putting the data closer to the processor or accelerator, or vice versa, is a general strategy to mitigate the memory wall issue. Scratchpad memory (SPM) is typically used in multiprocessor System-on-Chip (MPSoC) to improve the performance and reduce the power consumption [10], [11]. The SPMs can be considered as local memories to processor and are typically implemented using SRAM for embedded systems on FPGAs. Data and/or instructions are typically moved from the off-chip memory to the local SPMs for faster access. However, the capacity of the SPMs is typically small compared with the off-chip DRAMs. For applications such as image processing and computer vision algorithms in which the operations need to be performed on large amounts of data, the use of SPM with limited capacity may bring a large communicate overhead between the SPM and the off-chip memory in addition to the extra programming complexity. A direct access to the off-chip memory for the hardware accelerator should be superior than the SPM when dealing with large amounts of data.



(a) Access data through microprocessor. (b) Directly connect to memory.

Fig. 1. Two different configurations for hardware accelerators to access data.

In this work, we use the SIFT implementation on Xilinx Zynq-7045 device as a case study to demonstrate how to improve both performance and energy efficiencies on heterogeneous systems by designing different hardware architecture and increasing memory bandwidth for hardware accelerators. We designed two different approaches to accelerating SIFT on the Xilinx Zynq-7045 device. In the first approach, the hardware accelerator is fully customized and is connected to the ARM CPU as a slave through AXI general-purpose (GP) interface, high-performance (HP) interface, and accelerator coherency port (ACP) interface, respectively. It turns out that by using AXI GP interface the hardware accelerator actually will bring

down the performance of the convolution stage [12] in the SIFT algorithm on an image of 4,288×2,848 pixels. This performance reduction is mainly due to a couple of reasons. First, the ARM core on the Zynq-7045 is quite powerful and is very efficient dealing with the 1-D convolution operation when the cache is enabled. Second, the general-purpose AXI interface between the ARM core and the accelerator is very slow with a bandwidth $<$ 25 MB/s. Further the CPU needs to spend extra clock cycles on sending data to and receiving data from the accelerator. In order to further improve the performance, the hardware accelerator is directly connected to the off-chip memory through the AXI HP interface. A DMA module, which is implemented using the programmable logic, is added to handle the data movement between the accelerator and the off-chip memory. Results show that the convolution involving both the hardware accelerator and the CPU is 6.22 times faster than the pure software implementation while reducing the energy consumption by 5.97 times. On top of the second method, we further connect the hardware accelerator to the AXI ACP port, which connects to the L2 caches through the snoop cache controller on the ARM cores, instead of the high-performance AXI port. Under this connection, the hardware accelerator accesses the data through the CPU cache instead of directly connecting to the off-chip memory. Experiment results show that the last method can achieve ∼10 times for both performance improvement and energy reduction compared with the pure software implementation for the convolution stage of the SIFT algorithm.

In the second approach, a distributed multi-processor hardware system with its programming model is built. Pre-analysis of the SIFT algorithm shows that the two stages of octave gradient and key description generation contain thousands of core function calls, and each one consumes little time to execute. Every function call accesses a few inconsecutive addresses. Through our multi-processor system, thousands of data independent computation is distributed among up to 46 processing elements (PEs). Experiment results show that this design can achieve a better performance than the pure software implementation with a lower power consumption.

In the remainder of this paper, we first introduce the background and the related work in Section II. The SIFT algorithm is briefly discussed in Section III, the convolution stage in particular. In the same section, we discuss the design of the hardware accelerator and the two methods for connecting it to the memory. We also present the multi-processor system to accelerate the last two stages of the SIFT algorithm. We present the experiment results in Section IV. Finally, we conclude this work and give some remarks regarding the future work in Section V.

II. Background and Related Work

In order to achieve better energy-efficiency and performance, algorithm-specific custom accelerators are usually used in mobile devices equipped with ARM cores [13], [14]. Compared with general-purpose processors, hardware accelerators perform operations of data transferring and computation, reducing energy consumption by two orders of magnitude [15]. Significant research has been focusing on high level description of software algorithms with the generation of hardware circuits [16], [17]. Some algorithms take advantages of data-level parallelism, and can be accelerated by using GPGPUs that are built for massive data parallel computation. However, GPGPUs still consume much more energy than custom hardwares [12].

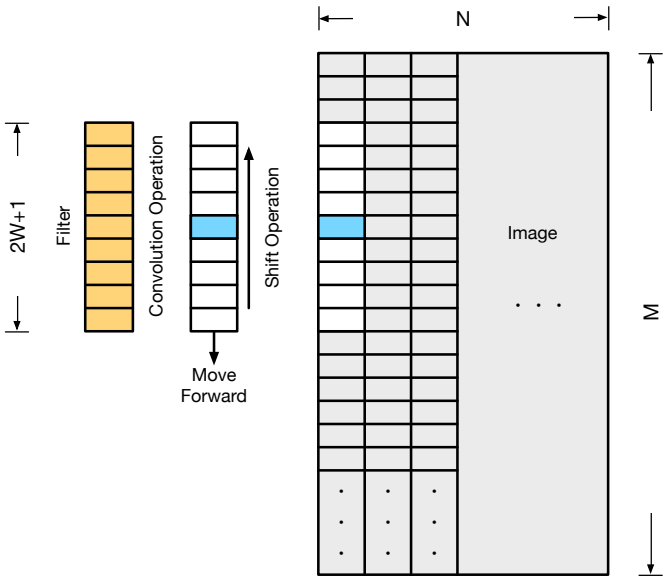


Fig. 2. Principle of the 1-D convolution between the filter and the image.

A domain-specific convolution model is discussed in [12] that can be programmed to adapt different applications. Our convolution hardware is designed for SIFT algorithm. Therefore, our accelerator is deeply optimized for the data-flow and data-locality features inside SIFT to reduce resource utilization and to save energy consumption. In order to utilize data parallel features, previous research tried to build multi-core systems along with their programming models by using hybrid computation unites [18], [19]. Our multi-processor system is designed on hybrid reconfigurable devices. Both hard core and reconfigurable resources are well utilized for better performance and more flexibility. In addition, slaves and host are configured to share the same address spaces, reducing the overhead of transferring data. To accomplish even better performance, a shared cache and its cache controller are connected to both the slaves and the host.

The purpose of this work is not purely on accelerating the SIFT algorithm. Nevertheless, there are a couple of previous work on the SIFT algorithm itself. Mizuno *et al.* presented a hardware implementation of SIFT on Altera FPGA [20] to deal with VGA frames at 640×480 resolution. They divided the image to much smaller regions of interest (ROIs) and then processed those ROIs in sequence. Huang *et al.* presented another hardware implementation of SIFT targeting the VGA frames [21]. Their work could efficiently deal with images with fewer than 890 feature points. Deng *et al.* revised the SIFT algorithm using the polar sampled descriptor [22]. They reduced the descriptor dimension from 128 to 72, which was also used in [23]. More recently, a complete image matching design targeting 1,280×720 video frames was presented in [24]. SIFT algorithm was implemented for detecting features. The original SIFT descriptor was replaced with BRIEF (binary robust independent elementary feature) descriptor.

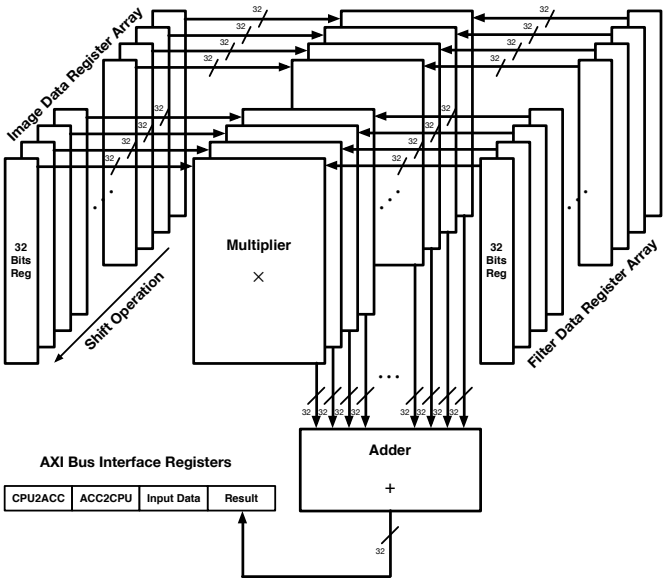


Fig. 3. The design of the accelerator and its integration through the general-purpose AXI ports (Acc-1).

TABLE I. ANALYSIS OF SIFT ALGORITHMS.

Stage	Execution Time	Core Function Calls
Down sample	0.61%	–
Up sample	0.18%	–
Convolution	**37.49%**	**72**
DoG	0.44%	–
Find & refine key	0.38%	–
Octave gradient	**19.34%**	**29,944**
Key description generation	**41.56%**	**34,873**
Total	100%	–

*The size of the test image: 4,288×2,848.

III. ACCELERATING THE SIFT ALGORITHM

The application we used in this work is scale-invariant feature transform (SIFT), which is an algorithm in computer vision to detect and describe local features in images [25]. As shown in Table I, the whole SIFT process consists of multiple stages. Three stages, *convolution*, *octave gradient* and *key description generation*, take the most time to carry out. We studied the memory access patterns for these three stages. It was found that the two stages of *octave gradient* and *key description generation* contain thousands of short memory access periods, in each of which a few inconsecutive addresses are accessed. On the other hand, it was found that the *convolution* stage consists of dozens of long memory access periods, each of which contains consecutive memory accesses. Therefore, we decided to accelerate the three stages using two different approaches in this work.

Figure 2 illustrates the principle of the convolution operation between the filter and the image. The 1-D convolution between the filter and the image will be carried out on all pixels in the image. The image will be processed for 72 times as 36 pairs of convolutions using 5 different 1-D filters, i.e., $W = 5, 7, 8, 10, 13$, respectively. In order to calculate the convolution value at the position of the blue pixel as shown in Figure 2, a vertical window centered at the blue pixel is mapped to the source image. Then the 1-D convolution is carried out between the values of the filter and the pixel values within the window. In order to perform the convolution on the next pixel, the window moves down by one pixel such that the center of the window maps to the next pixel.

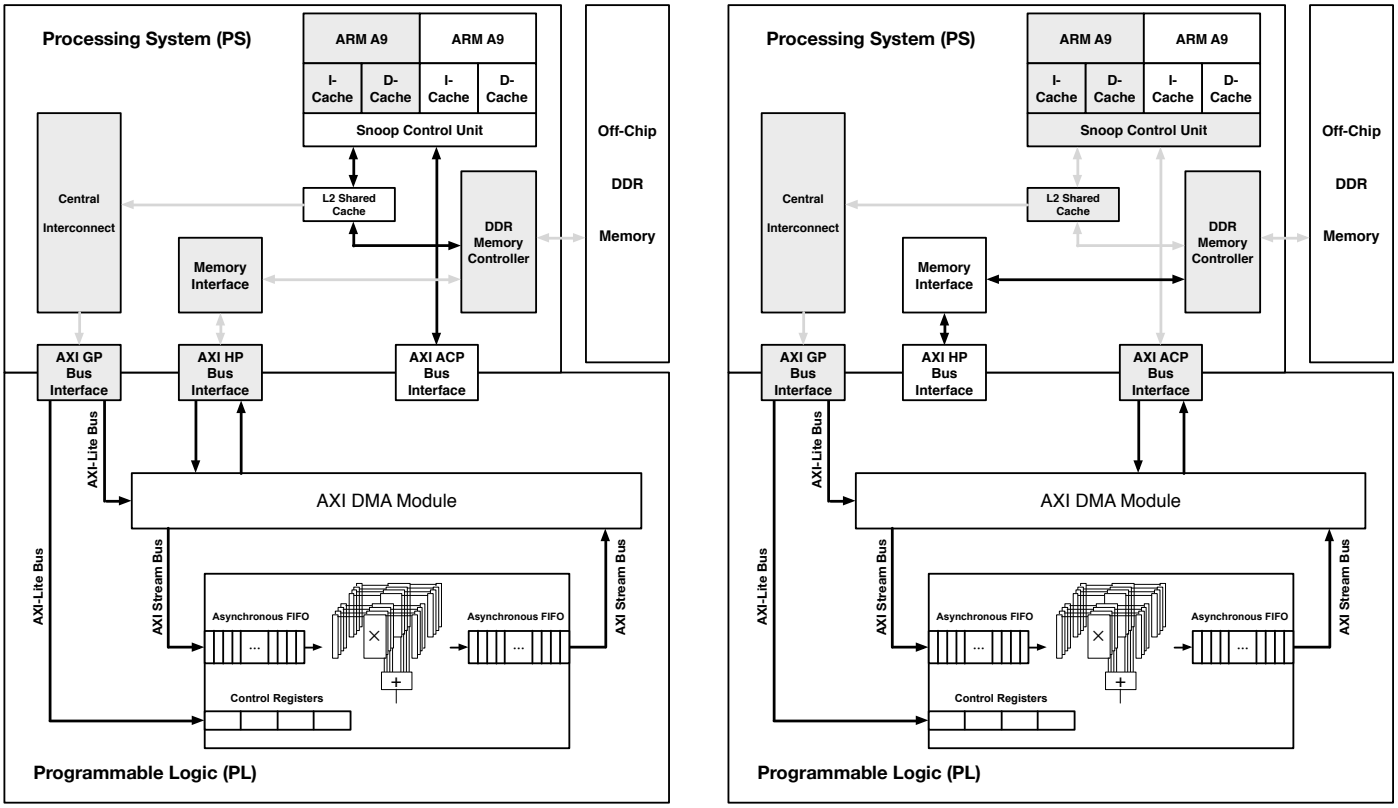
There is a total of 36 convolution pairs in this stage. For each pair of convolution, there are two perpendicular one-dimensional convolutions, i.e., $A \xrightarrow{convolution} B$ and $B \xrightarrow{convolution} C$. In the first convolution, the source image A is stored in row order, i.e., all pixels in the same row are saved in consecutive memory locations. The filter goes through all rows by performing 1-D convolution operation on each pixel. The result of the first convolution ($A \longrightarrow B$) will be saved into the memory as B in column order. In other words, two pixels of A from two consecutive memory locations will be saved in two inconsecutive memory locations as B. After the first convolution is finished, the $B \longrightarrow C$ convolution is carried out in column order on image B and the image C will be saved in row order.

Because these 72 convolutions between the filters and the image need to be performed one by one, we designed a single hardware accelerator that can carry out a convolution between a filter and a window when $W = 13$. As shown in Figure 3, there are 27 multipliers that can carry out 27 multiplications in parallel. All 27 multipliers are fully pipelined such that they can perform the 1-D convolution between the filter and the window every clock cycle. The outputs of the 27 multipliers are fed into a fully-pipelined adder chain of 5 layers for generating the sum. For the convolution of smaller filters and windows, the contends of certain items at both ends of the filter are set to zeros. In the implementation of this work, both multiplication and addition are single-precision floating-point operations. Running at 50 MHz, the depths of the multiplier and the adder are 6 and 11, respectively.

The implementation platform is Xilinx ZC706 Evaluation Board, which contains an XC7Z045 FFG900-2 AP SoC and 1 GB off-chip DDR3 memory. The ZC-7045 device consists of a Dual ARM Cortex-A9 MPCore processor and 350K logic cells. Each ARM core has separate L1 data and instruction caches. The 512 KB unified L2 cache is shared by two cores. For the hardware accelerators developed using the programmable logic (PL), there are three types of ports to connect to the CPU and the off-chip memory, i.e., the general-purpose ports, the high-performance ports, and the accelerator coherency port (ACP). All the three types of ports use Advanced eXtensible Interface 4 (AXI4).

We designed the hardware accelerator for carrying out the convolution and integrated it to the system using two methods as described in Figure 1.

In the first method, the hardware accelerator is connected to the CPU through the general-purpose AXI port, as shown in Figure 3. Each general-purpose AXI port consists of four registers, i.e., CPU2ACC, ACC2CPU, Input Data, and Result. CPU2ACC and ACC2CPU are used for control signals. The values of the filter and the pixels of the image are sent from CPU to the hardware accelerator through the Input Data register. The convolution results are returned through the Result register. The main SIFT application running on the CPU is a single-thread implementation. Therefore, only one ARM core is used in this work. Since the CPU can send



(a) Through high-performance AXI port (Acc-2.1).

(b) Through ACP AXI port (Acc-2.2).

Fig. 4. Connect the hardware accelerator directly to the off-chip memory (Acc-2).

at most one pixel to the hardware accelerator in one cycle, only one hardware accelerator is instantiated. Based on Xilinx document [26], the throughput of the general-purpose AXI port is < 25 MB/s

Furthermore, we connect the hardware accelerator directly to the memory through either the high-performance AXI port or the ACP AXI port, both having a bandwidth of 1.2 GB/s.

In Figure 4(a), the high-performance (HP) AXI ports are connected to the DDR memory controller through the memory interface inside the PS part. By using the AXI-Stream bus to transfer datasets to the user custom IP logic via the DMA module, the HP AXI port can provide the maximum bandwidth, which can benefit data-hungry applications such as image processing algorithms. The control signals of the DMA module, along with the memory addresses are sent by the ARM processors via the general purpose AXI-lite bus, which is connected to the central interconnect. With the help of high speed DMA module and bus interface, the ARM processors are free from data transaction. Once the transaction is done, the processors will be noticed by interrupts or with polling.

Figure 4(b) shows the connectivity of ACP AXI port, where the IP topology is similar to the HP AXI port. In terms of throughput, both HP and ACP interfaces can provide the highest throughput capability, which is up to 1.2 GB/s per port interface. The differences lie in the connectivity inside the PS part. When using DMA module via ACP AXI interface, the ACP AXI port is connected to the snoop control unit, which is connected with private L1 caches and shared L2 cache of the dual ARM cores to provide cache coherence. When the DMA module intents to use the ACP AXI port to move data between the off-chip DDR memory and the programmable logic (PL), all memory access requests are handled by the snoop control unit. In this way, the average memory access time is reduced, especially for data with obviously spatial and temporal localities. And meanwhile, data access latency from processor side is potentially reduced if the processors write and read data within the memory blocks that are just transferred by the DMA module. In addition, since the Zynq ARM caches can only support write-back and write-allocate policy, the need to invalidate and flush cache blocks is prevented compared with the HP AXI mode. However, the drawbacks are that large datasets have chances to cause cache thrashing. Therefore, in this design, large data transfer is carefully divided into small sets.

In the custom IP accelerator, slave and master ports are connected to the AXI DMA module via the AXI stream bus, respectively. Asynchronous FIFOs are used to provide redundancy for data flows in to and out from specific convolution hardware. Similar as the previous design via the general purpose AXI bus, the convolution hardware is designed to be fully pipelined. But the input data and output data registers are replaced by FIFOs. FIFOs are configured with the depth of only 128 since transfer exceptions rarely happen. Control signals are provided through the general purpose AXI-lite bus from ARM processor to coordinate the behavior of accelerator.

In the method for accelerating octave gradient and key

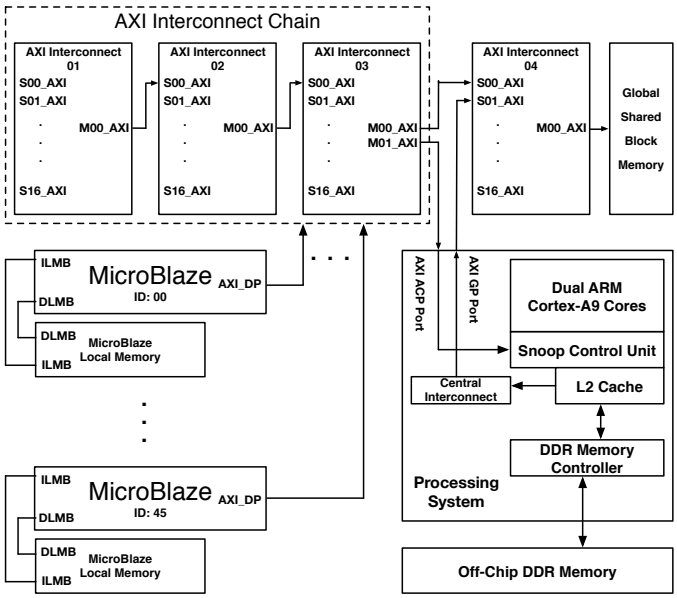


Fig. 5. The design of multi-processor system and its integration into Zynq.

```
//Host:
int main () {
  const int numSlaves = TotalMicroBlazes;
  writeGlobalData();
  //WRITE global data to shared block memory
  DCacheFlush();
  for (i = 0; i < numSlaves; i++) {
    globalStatus[i] = SignalStart;
  }
  DCacheFlush();
  for (i = 0; i < numSlaves; i++) {
    while (globalStatus[i]) != SignalFinish)
      wait();
  }
  return 0;
}

//Slaves:
int main () {
  const int ID = MicroBlazeID;
  const int numSlaves = TotalMicroBlazes;
  while (true) {
    while (globalStatus[ID] != SingalStart)
      wait();
    readGlobalData();
    //READ global data from shared block memory
    for (j = ID; j < numThreads; j += numSlaves) {
      threadExecution(j);
    }
    globalStatus[ID] = SignalFinish;
  } //Continue to next program
  return 0;
}
```

Fig. 6. Pseudocodes of the host and the slaves.

description generation, as Figure 5 shows, up to 46 MicroBlazes as slaves are bundled together to be connected with AXI interconnect chain. Processing system (PS) as the host and the AXI interconnect chain are connected to the global shared block memory (GSBMem) using another separate AXI interconnect. Another output port of the AXI interconnect chain is connected to AXI ACP interface insides PS. In this

(a) The source image (4,288×2,848).

(b) The image augmented with key points.

Fig. 7. Apply the SIFT algorithm on a grayscale image.

way, the GSBMem is shared by all slaves and host. In addition, all slaves are able to access the off-chip DDR memory, which is shared by PS via the AXI ACP interface. In other words, through AXI ACP interface, all slaves share the same data cache (i.e., the last level cache in PS).

Figure 6 demonstrates the template codes for both host and slave sides, which are synchronized by the global status variables stored in GSBMem. At the beginning of the host program, all global data (including memory addresses of input data for slaves, and global variables) will be sent to GSBMem. *SignalStart* will be written to the associate locations in GSBMem to trigger slaves. Two *DCacheFlush* functions are used to guarantee that all newly-modified data have been written back to the lower memory so that slaves can access the newest data. Then, the host will wait for all slaves to finish by fetching *SignalFinish* from all slaves. In slave side, each slave is required to perform independent computation by assigning a unique ID number at the compilation time. After every computation, this ID number will be added by the number of total slaves. In this way, all computation threads are distributed among slaves. After all computation threads finish, *SignalFinish* will be written back to GSBMem to notify the host.

TABLE II. PERFORMANCES OF SIFT IMPLEMENTATION UNDER VARIOUS CONFIGURATIONS (UNIT: s).

Stage	D-Cache Disabled		D-Cache Enabled				
	w/o Acc	w/ Acc-1*	w/o Acc	w/ Acc-1*	w/ Acc-2.1†	w/ Acc-2.2‡	w/ Acc-3§
1	36.56	35.96	1.66	1.66	1.67	1.66	1.66
2	10.98	10.78	0.15	0.17	0.15	0.16	0.15
3	**2,248.98**	**436.22**	**121.42**	**217.34**	**19.52**	**12.57**	**12.61**
4	26.34	26.56	3.49	3.51	3.52	3.51	3.52
5	22.80	22.83	5.43	5.39	5.42	5.44	5.43
6	**1,148.30**	**1,150.21**	**70.92**	**70.87**	**70.90**	**70.88**	**58.42**
7	**2,492.48**	**2,490.65**	**175.51**	**175.50**	**175.47**	**175.48**	**142.65**
Total	5,997.30	4,180.26	380.36	476.01	280.78	272.94	227.58

*Acc is connected to the CPU as a slave as in Figure 1(a). On Zynq7045, the Acc is connected to the general-purpose AXI ports.
†Acc is directly connected to the memory as in Figure 1(b). On Zynq7045, the Acc is connected to the high-performance AXI ports.
‡Acc is directly connected to the memory as in Figure 1(b). On Zynq7045, the Acc is connected to the ACP AXI port.
§Acc is combined with both Acc-2.2 and multi-processor system with shared cache as in Figure 5.

TABLE III. POWER AND ENERGY CONSUMPTION OF PERFORMING CONVOLUTION UNDER VARIOUS CONFIGURATIONS.

	D-Cache Disabled		D-Cache Enabled			
	w/o Acc	w/ Acc-1	w/o Acc	w/ Acc-1	w/ Acc-2.1	w/ Acc-2.2
CPU Active (W)	0.50					
Logic Static (W)	0.125					
Logic Active (W)	–	0.157	–	0.157	0.164*	0.164†
Logic Idle (W)	–				0.139	0.139
Energy (J)	1,405.61‡	286.60§	75.89¶	142.79‖	12.55**	8.10††

*Accelerator is active for 3.23 seconds and idle for 16.29 seconds.
†Accelerator is active for 2.81 seconds and idle for 9.76 seconds.
‡1405.61=(0.50+0.125)×2248.98.
§286.60=(0.50+0.157)×436.22.
¶75.89=(0.50+0.125)×121.42.
‖142.79=(0.50+0.157)×217.34.
**12.55=(0.50+0.164)×3.23+(0.50+0.139)×16.29.
††8.10=(0.50+0.164)×2.81+(0.50+0.139)×9.76.

IV. EXPERIMENTS AND RESULTS

We used the image shown in Figure 7(a) for the benchmark in this work. The ∼180 thousand generated key points are illustrated in Figure 7(b). The development software is Xilinx Vivado 2014.2 on Windows 7. The ARM cores run at 667 MHz. The off-chip 1-GB DDR3 memory runs at 533 MHz. The frequency of the hardware accelerator in the first approach is 50 MHz. In the second approach, all PEs are configured as 200 MHz.

We have 7 different implementations of SIFT listed in Table II. For all 7 implementations, the I-Caches of the ARM cores are always enabled, and only one ARM core is used. We list the timing results of all stages of the SIFT implementation in Table II. Except *w/ Acc-3* that combines two different approaches to accelerating the whole SIFT algorithm, others focus on the convolution part.

We first disable the D-Cache on the CPU and run the pure software version. The convolution stage alone takes more than 2,000 seconds. When the hardware accelerator is used to perform the convolution, the time is reduced to 436.22 seconds, achieving a 5.15× speedup. The data access volume to the off-chip memory is exactly the same for both the pure software version and the accelerated version when the D-Cache of the CPU is disabled. Although it brings extra overhead to send the data to the hardware, it only takes one clock cycle (of 50 MHz frequency) to finish one convolution operation when the pipeline becomes full.

Once the D-Cache is enabled, the performance of the pure software version is significantly improved by 18.52 times, i.e., 2248.98 seconds → 121.42 seconds. The performance of the hybrid implementation (i.e., Acc-1) is improved also by 2 times, i.e., 436.22 seconds → 217.34 seconds. However, its performance is even worse than the pure software implementation. The main reason is due to the low bandwidth of the general-purpose AXI port, which is < 25 MB/s. Apparently the general-purpose AXI port becomes the bottleneck in the hybrid implementation. In addition, the CPU needs to spend extra clock cycles for communicating with the hardware accelerator.

In order to release the performance potential of the hardware accelerator, it is essential to deliver the data in a much more efficient way from the memory to the accelerator. As mentioned before, we designed two methods to connect the hardware accelerator directly to the memory. The first method is to connect the hardware accelerator to the memory using the AXI HP port, which has a bandwidth of 1.2 GB/s, through a DMA module. Now the computation time is significantly reduced to 19.52 seconds, achieving a 6.22× speedup compared with the pure software implementation. In the second method, we further add the L2 cache and the snoop cache controller, which are parts of the PS, into the data path by connecting the hardware accelerator to the AXI ACP interface instead of the AXI HP interface. Now thanks to the advanced features of the cache, such as pre-fetching and write-back policy, the computation time is further reduced to 12.57 seconds, achieving a 9.66× speedup.

After convolution part is accelerated, the last two stages, *octave gradient* and *key description generation*, become the bottlenecks of the whole algorithm. Approach *w/ Acc-3* is proposed to combine both convolution accelerator with multi-processor system together. Dozens of PEs are assigned to run the same program with multiple input data. With the help of shared L2 cache and cache controller in PS, the last two stages are also accelerated by 17.6% and 18.7%, respectively. However, according to Amdahl's law, compared with ARM single core execution, multi-processor system is supposed to achieve a 13× speedup. Three reasons affect the performance of the multi-processor system. (1) In terms of architecture, ARM Cortex-A9 core is more advanced than MicroBlaze processor. Therefore ARM will have higher IPC under the same frequency. (2) Our multi-processor system only takes advantage of the last level cache (LLC). Instead, ARM core utilizes both levels of caches (i.e., L1 and L2). (3) AXI interconnect becomes inefficient when memory requests and the number of master and slave devices increase.

We also measured the power consumptions of various components during both convolution stage and the whole SIFT algorithm. We have the following observations. (1) The CPU is always consuming 0.50 watts no matter it is actively computing or simply waiting for the hardware accelerator. (2) The programmable logic consumes 0.125 watts even no hardware accelerator is implemented on it. (3) When the hardware accelerator is implemented on the programmable

TABLE IV. POWER AND ENERGY CONSUMPTION OF PERFORMING SIFT UNDER VARIOUS CONFIGURATIONS.

	w/o Acc-1*	w/o Acc-2†	w/ Acc-3‡
CPU Active (W)	0.50		
Logic Static (W)	0.125		
Logic Active (W)	–	–	0.525
Logic Idle (W)	–		0.313
Energy (J)	3,748.31§	237.73¶	224.87‖

*D-Cache is disabled in ARM.
†D-Cache is enabled in ARM.
‡Acc is combined with both Acc-2.2 and multi-processor system with shared cache as in Figure 5.
§3748.31=(0.50+0.125)×5997.30.
¶237.73=(0.50+0.125)×380.36.
‖224.87=(0.50+0.125)×10.76+(8.10+12.61×0.313)+(0.5+0.525)×201.07.

TABLE V. PROGRAMMABLE RESOURCE UTILIZATIONS UNDER VARIOUS CONFIGURATIONS.

Resource	D-Cache Disabled	D-Cache Enabled			
Type	w/ Acc-1	w/ Acc-1	w/ Acc-2.1	w/ Acc-2.2	w/ Acc-3
Register	7,466 (1.7%)	7,466 (1.7%)	11,058 (2.5%)		137,783 (31.3%)
LUT	11,648 (5.3%)	11,648 (5.3%)	14,635 (6.7%)		148,044 (67.7%)
DSPs	137 (14.5%)	137 (15.2%)	137 (15.2%)		413 (45.9%)
BRAM	0 (0%)	0 (0%)	5 (0.9%)		432 (79.3%)

logic, it consumes more power when the accelerator is running than when the accelerator is idle.

When the hardware accelerator is connected to the general-purpose AXI port, both CPU and accelerator work together during the whole period of the convolution stage. When the hardware accelerator is connected to the high-performance AXI port or ACP AXI port, the accelerator first works on the convolution and writes all results back to memory in a consecutive order. The results need to be re-organized by the CPU by re-writing them back to the correct locations. During the re-ordering processing, the accelerator stays idle. Table III and Table IV list the power and energy consumptions of all implementations with the self-explaining equations demonstrating how the energy consumption is calculated for each implementation with the respective power and time results. Apparently, in convolution stage, enabling the D-Cache can reduce the energy consumption significantly. With the D-Cache enabled, it is critical to connect the hardware accelerator directly to the memory. As the results show, the direct access to memory through high-performance AXI port and ACP AXI port can reduce the energy consumption by 5.97× and 9.23×, respectively, compared with the pure software implementation.

When it comes to the whole SIFT algorithm, reduction of power consumption is not significant when D-Cache is enabled. It is resulted from different energy efficiencies for different heterogeneous components. When dealing with the same task, customized hardware (i.e., Acc-2.2) is more efficient in terms of performance and power consumption than general-purpose processors (i.e., ARM and MicroBlaze).

The hardware resource utilizations of various implementations are listed in Table V. We used DSP modules to implement the single-precision floating-point multipliers and adders. The Acc-2 implementations use some BRAM modules to implement the FIFOs for communicating with the AXI DMA module. The Acc-2 implementations also require more registers and LUTs than the Acc-1 implementation to implement the AXI DMA module. Because the Acc-3 implements large AXI interconnects and dozens of MicroBlazes in which DSPs are used to build multipliers, it utilizes much more resources than other implementations.

V. CONCLUSION

Hardware accelerators are capable of achieving significant performance improvement for many applications. In a straightforward way, the hardware accelerator is connected the microprocessor as a slave co-processor. However, as the microprocessor becomes more and more powerful as time goes, this simple integration method may not be able to improve the performance. The results in this work show that the use of hardware accelerator actually brings down the performance of the convolution stage in the SIFT algorithm on an image of 4,288×2,848 pixels. The low-bandwidth general-purpose AXI port on the Xilinx Zynq-7045 device is the main bottleneck. The extra clock cycles the CPU spends on communicating with the accelerator bring additional overheads. Therefore it is critical to provide the efficient access to the memory for the hardware accelerator when dealing with large amounts of data. In the improved method, the hardware accelerator is directly connected to the off-chip memory through the high-performance AXI interface. With the help of the DMA module, the hardware accelerator can access the memory at a bandwidth up to 1.2 GB/s. On top of the second method, we further connect the hardware accelerator to the ACP AXI port, which connects to the L2 caches through the snoop cache controller on the ARM processor instead of the high-performance AXI port. Under this connection, the hardware accelerator accesses the data through the CPU cache instead of directly connecting to the off-chip memory. Furthermore, a distributed multi-processor system is designed to handle large amount of function calls with inconsecutive memory accesses in each of them. Low efficiency of MicroBlaze and large AXI interconnects affects further performance improvement and power consumption reduction. Experiment results show that the hybrid design in which the hardware accelerator directly connects to the memory through the CPU cache can achieve ∼10 times and better improvement of using multi-processor system for both performance improvement and energy reduction compared with the pure software implementation for the convolution stage and the whole SIFT algorithm, respectively.

As the timing results shown in Table II, the last two stages still have much room to be improved. We plan to design hybrid multi-processor system with both general-purpose processors and customized hardware threads to achieve even better efficiency in terms of performance and energy consumption in the future work.

ACKNOWLEDGMENT

This work is supported in part by NSF and NASA under grant CNS-1219062 and 12-2012EPSCoR-0026, respectively. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF and NASA. The authors are grateful to Xilinx, Inc. for the donation of the FPGA boards and the software.

REFERENCES

[1] C. Su and A. M. Despain, "Cache design trade-offs for power and performance optimization: A case study," in *Proceedings of the 1995 International Symposium on Low Power Design*, ser. ISLPED '95, 1995, pp. 63–68.

[2] K. Puttaswamy, K. Choi, J. C. Park, V. J. Mooney, III, A. Chatterjee, and P. Ellervee, "System level power-performance trade-offs in embedded systems using voltage and frequency scaling of off-chip buses and memory," in *Proceedings of the 15th International Symposium on System Synthesis*, ser. ISSS '02, 2002, pp. 225–230.

[3] M. Alipour, M. E. Salehi, and K. Moshari, "Cache power and performance tradeoffs for embedded applications," in *Proc. IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE)*, Dec. 2011, pp. 26–31.

[4] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[5] 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. 78–85, Feb. 2008.

[6] L. Pozzi and P. Ienne, "Exploiting pipelining to relax register-file port constraints of instruction-set extensions," in *Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '05. New York, NY, USA: ACM, 2005, pp. 2–10.

[7] T. Kluter, P. Brisk, P. Ienne, and E. Charbon, "Speculative dma for architecturally visible storage in instruction set extensions," in *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '08. New York, NY, USA: ACM, 2008, pp. 243–248.

[8] T. Kluter, P. Brisk, E. Charbon, and P. Ienne, "Way stealing: A unified data cache and architecturally visible storage for instruction set extensions," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 1, pp. 62–75, Jan 2014.

[9] X. Sun and D. Wang, "Concurrent average memory access time," *IEEE Trans. Comput.*, no. 99.

[10] V. Suhendra, C. Raghavan, and T. Mitra, "Integrated scratchpad memory optimization and task scheduling for MPSoC architectures," in *Proc. 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'06)*, Oct. 2006, pp. 401–410.

[11] Z. Chen and A. W. Y. Su, "A hardware/software framework for instruction and data scratchpad memory allocation," *ACM Transactions on Architecture and Code Optimization*, vol. 7, no. 1, pp. 2:1–2:27, Apr. 2010.

[12] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: Balancing efficiency & flexibility in specialized computing," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 24–35.

[13] *Tegra processors*, NVIDIA Corporation.

[14] *Apple A7 SoC*, Apple Inc.

[15] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 37–47.

[16] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable domain-specific computing," *Design Test of Computers, IEEE*, vol. 28, no. 2, pp. 6–15, March 2011.

[17] G. Martin and G. Smith, "High-level synthesis: Past, present, and future," *Design Test of Computers, IEEE*, vol. 26, no. 4, pp. 18–25, July 2009.

[18] B. Senouci, A. Kouadri M, F. Rousseau, and F. Petrot, "Multi-cpu/fpga platform based heterogeneous multiprocessor prototyping: New challenges for embedded software designers," in *Rapid System Prototyping, 2008. RSP '08. The 19th IEEE/IFIP International Symposium on*, June 2008, pp. 41–47.

[19] P. H. Wang, J. D. Collins, G. N. Chinya, H. Jiang, X. Tian, M. Girkar, N. Y. Yang, G.-Y. Lueh, and H. Wang, "Exochi: Architecture and programming environment for a heterogeneous multi-core multithreaded system," in *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07. New York, NY, USA: ACM, 2007, pp. 156–166.

[20] K. Mizuno, H. Noguchi, G. He, Y. Terachi, T. Kamino, H. Kawaguchi, and M. Yoshimoto, "Fast and low-memory-bandwidth architecture of sift descriptor generation with scalability on speed and accuracy for vga video," in *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications*, ser. FPL '10, 2010, pp. 608–611.

[21] F.-C. Huang, S.-Y. Huang, J.-W. Ker, and Y.-C. Chen, "High-performance SIFT hardware accelerator for real-time image feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 3, pp. 340–351, Mar. 2012.

[22] W. Deng, Y. Zhu, H. Feng, and Z. Jiang, "An efficient hardware architecture of the optimised SIFT descriptor generation," in *Proc. 22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012, pp. 345–352.

[23] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao, and W. Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *Proc. International Conference on Field-Programmable Technology (FPT)*, Dec. 2009, pp. 30–37.

[24] J. Wang, S. Zhong, L. Yan, and Z. Cao, "An embedded system-on-chip architecture for real-time visual detection and matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 3, pp. 525–538, Mar. 2014.

[25] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[26] *Zynq-7000 All Programmable SoC Technical Reference Manual (UG585 v1.7)*, Xilinx, Inc., Feb. 2014.