TRANSACTIONS ON ELECTRICAL AND ELECTRONIC MATERIALS
Vol. 13, No. 6, pp. 278-282, December 25, 2012

Regular Paper

pISSN: 1229-7607 eISSN: 2092-7592
DOI: http://dx.doi.org/10.4313/TEEM.2012.13.6.278

A Vector Instruction-based RISC Architecture for a Photovoltaic System Monitoring Camera

Youngho Choi and Hyungkeun Ahn†
Department of Electrical Engineering, Konkuk University, Seoul 143-701, Korea

Received September 4, 2012; Accepted September 12, 2012

Photovoltaic systems have emerged to be one of the cleanest energy systems. Therefore, many large scale solar parks and PV farms have been built to prepare for the post fossil fuel ages. However, due to their large scale, to efficiently manage and operate PV systems, they need to be visually monitored within the range of infrared ray through the Internet. To satisfy this need, the efficient implementation of a high performance video compression standard is required. This paper therefore presents an implementation of H.264 motion estimation, which is one of the most data-intensive and complicated functions in H.264. To achieve this, this work implements vector instructions in hardware and incorporates them in a generic RISC processor architecture, thus increasing the processing speed while minimizing hardware and software design efforts. Extensive simulation results show that this proposed implementation can process motion estimations up to 13 times faster.

Keywords: Photovoltaic, Monitoring, Infrared ray images, RISC, Vector instruction

1. INTRODUCTION

Photovoltaic systems have emerged to be one of the cleanest energy systems. Therefore, many large scale solar parks and PV farms have been built to prepare for the post fossil fuel age. However, due to their large scale, in order to trace the degradation process of the system, and thus to efficiently manage it, PV systems need to be monitored visually through the Internet [1-3].

In such a visual monitoring system, a camera takes infrared ray images of PV modules and cells to identify damages in real time and transmits these images to monitoring centers. To do this effectively and economically, efficient video compression algorithms are required, which reduce the amount of data to be transmitted and stored. For this purpose, many video compression standards such as MPEG2, MPEG4, and H.264 [4-6] have been proposed and used over the last few decades. Among them, H.264 has the highest compression efficiency as it was developed most recently. However, since its high compression efficiency is mainly due to its high computation complexity, it is very difficult to implement it in software for real-time applications such as photovoltaic monitoring systems. To resolve this problem, and thus to provide real-time video compression encoding with minimal quality degradation, full hardware design solutions have been proposed [7,8]. However, these dedicated hardware architectures have less flexibility and a long time-to-market period.

In order to increase design flexibility, a co-design approach based on a reconfigurable platform has recently been proposed [9,10]. One of these works [9] proposes an algorithm which automatically partitions and schedules tasks for hardware and software. Because the run-time reconfigurable processing elements (PEs) of this algorithm can contain any specific data-dominated tasks, various combinations of PEs provide flexibility and reusability of hardware design efforts. However, a sub-optimized partitioning algorithm using automated tools degrades the performance of a co-design system compared to hand-made designs. In addition, its task manager algorithm which performs scheduling in the task unit means the system is limited in the parallel processing of the instruction unit and thus performance degradation occurs.

† Author to whom all correspondence should be addressed:
E-mail: hkahn@konkuk.ac.kr

To efficiently implement the H.264 compression algorithm, this paper presents an implementation of H.264 motion estimation, which is one of the most complicated and data-intensive tools of H.264. To minimize hardware design efforts, this work first identifies the necessary vector instructions which are data-intensive and frequently executed in the motion estimation of H.264, and are thus required to be implemented in the hardware. This allows us to limit our hardware design efforts only in these vector instructions, eliminating the need for a full hardware design. After the vector instructions are implemented in the hardware, they are incorporated into a generic RISC architecture with an out-of-order execution scheduler. Because the generic RISC processor can support pre-existing software development environments and instruction sets as well as newly-added vector instructions, the software programmers then only need to replace the complicated and data-intensive functions with the newly-added vector instructions. This enables to minimize additional software design efforts and thus, exploits the benefits of both software and hardware design solutions, i.e., flexibility and performance.

In Section 2, this paper describes the specifications of H.264 motion estimations, and in Section 3 the implementation of the H.264 motion estimations is described. Section 4 evaluates the performance of the implementation and Section 5 concludes this work.

2. DESIGN SPECIFICATIONS OF H.264 MOTION ESTIMATION

As a state-of-the-art video coding standard, H.264 provides a maximum increase in compression efficiency of 50% over a wide range of bit rates and video quality compared to previous standards. However, the H.264 is about 10 times more complex than a corresponding MPEG-4 standard [11]. Specifically, to reduce the temporal redundancy between successive pictures, the motion estimation algorithm is improved by adopting a variable block size for motion compensation and higher motion vector resolutions.

Compared to previous video coding standards, H.264 has a variable block size motion estimation scheme to represent one macroblock. Fig. 1 shows the candidate macroblock and sub-macroblock partitioning from the Inter16×16 mode to the Inter4×4 mode. A macroblock is composed of 16×16 pixels, and it can be divided into two 16×8 partitions, two 8×16 partitions or four 8×8 partitions. If the 8×8 partitions are selected, each of the four 8×8 sub-macroblocks within the macroblock may be split a further 4 ways as shown in Fig. 1(b). The results of the iterative partition motion search to find the best mode showed that the complexity and computation load of motion estimation increased, and consumed 60%~80% of the total H.264 encoding time [11].

3. IMPLEMENTATION OF H.264 MOTION ESTIMATION MODULE

To efficiently implement an H.264 motion estimation module, which is one of the most complicated tools in H.264, this study employs a vector instruction-based RISC architecture. For this design, this study first defines the vector instructions that are very complicated and consume a lot of execution time, and thus need to be implemented in hardware. To identify the vector instructions, the Intel Vtune performance analyzer 8.0 is utilized, which selects the most frequently called and complicated functions. Once such vector instructions are determined, only these

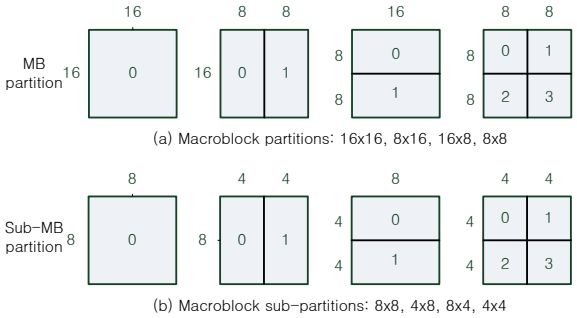

Fig. 1. Macroblock (MB) and Sub-MB partitions for motion estimation.

instructions are implemented in the hardware and incorporated into a generic out-of-order executable RISC. Since this processor provides a programmer with vector instructions, corresponding to the complicated functions of H.264, the programmer does not need to change the previously-open H.264 algorithm code, and instead replaces the most frequently-called and complicated functions of H.264 with implemented vector instructions. Consequently, this co-design methodology enables hardware and software design efforts to be minimized, this reducing the cost and time-to-market period. More details about this design process are given in the following subsections.

3.1 Identification of vector instructions

To define vector instructions for H.264 motion estimation, this work profiles the JM 10.1 reference software using the Intel Vtune performance analyzer 8.0. From the profiling result of H.264 motion estimations, 6 vector instructions are selected which are the most data-intensive and the most frequently used, as shown in Fig. 2: (1) 4×4SAD, (2) SATD, (3) 4×4RD, (4) 4×4RD2, (5) NAC, and (6) NAC2.

4×4SAD is a vector instruction which calculates the sum of absolute differences on two 4×4 matrixes. This instruction is used for integer-pixel searches to perform motion estimations. Since the 4×4SAD instruction is executed on the smallest pixel matrixes, i.e., 4×4 matrixes, this instruction can support any kind of H.264 variable block size such as 16×16, 16×8, 8×16, …, 4×4. This eliminates the need for additional vector instructions and makes the design of an H.264 motion estimation efficient.

SATD is a vector instruction which calculates the sum of the absolute transformed differences, and is frequently used for sub-pixel searches of motion estimations. Additionally, this vector instruction can be used to implement another important H.264 function, i.e., DCT4×4.

Since the data size of the 4×4SAD and SATD instructions selected above is in the form of a 4×4 block, data from the main memory to the vector registers should be 4×4 vectors. Furthermore, before the 4×4SAD and SATD are calculated, their source data should be loaded into a cache as soon as possible. Therefore, the 4×4RD and 4×4RD2 vector instructions need to be supported. 4×4RD reads the 4×4 vector data from memory into both a lower level cache and a vector register, which are frequently used for integer-pixel searches of motion estimations, while 4×4RD2 is very similar to 4×4RD but is used for sub-pixel searches of motion estimations. Furthermore, 4×4RD and 4×4RD2 can be used to implement other functions in motion estimation processes.

Many types of memory address pointers can be used to calculate the image blocks in H.264 motion estimations. Since the

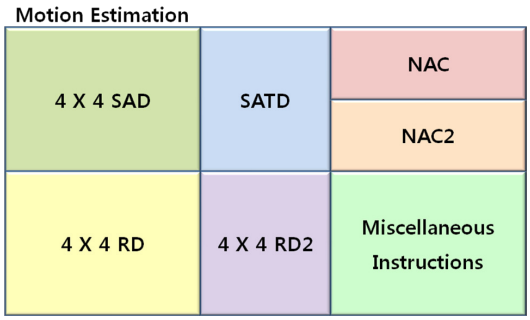

Fig. 2. Profiling results for motion estimation by Intel Vtune Analyzer.

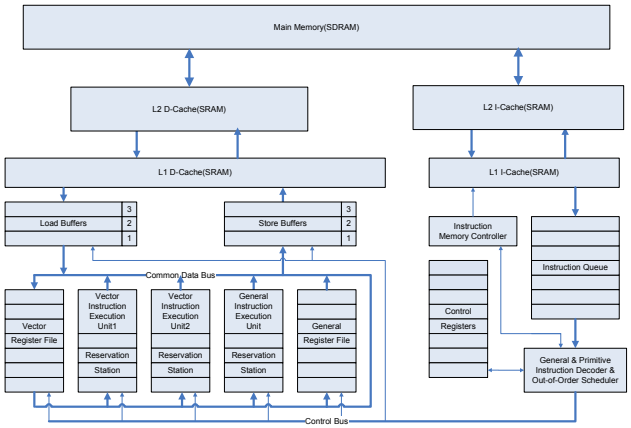

Fig. 3. An implementation of the out-of-order executable RISC architecture with vector instructions.

address of macroblocks or sub-macroblocks in one picture is frequently operated and moved to the next address in motion estimations, this work also defines NAC and NAC2 as vector instructions. NAC calculates the addresses for the next memory pointer and is used for integer-pixel searches while NAC2 is used for sub-pixel searches.

3.2 HW/SW co-design of H.264 motion estimation

To present an implementation of H.264 motion estimations, this work employs a general purpose RISC (Reduced Instruction Sets Computer) processor, out-of-order scheduler, main memory (SDRAM), L1/L2 Cache (SRAM), common data and control bus, hardware modules, instruction buffers (Queue), and register files to support vector or normal scalar instructions. Additionally, all the vector instructions identified in the previous subsection are incorporated in the out-of-order executable RISC processor as shown in Fig. 3. To support an out-of-order execution for vector instructions, the Tomasulo's algorithm [12], with reservation stations, is used.

As shown in Fig. 3, instructions are sent from L1 I-Cache into the instruction queue, where instructions are issued in a FIFO order by an instruction memory controller. Reservation stations include information used for detecting data dependencies among predefined vector instructions (hardware tasks) as well as general instructions (software tasks). This information enables both vector instructions and normal scalar instructions to be executed in an out-of-order fashion. The load buffers and the store buffers shown in Fig. 3 enable memory data accesses to reorder, reducing memory blocking. The data path and the control path shown in Fig. 3 are built to communicate through the common data and control bus.

The architecture presented in this work can be easily extended and customized for a wide range of video applications by reforming the vector instruction sets shown in Fig. 4. Moreover, because this architecture supports not only newly defined vector instructions but also general scalar instructions, software engineers do not need to fully understand the specific hardware architecture using MMX or VLIW technology to optimize the software implementation for a target specification. The software engineer only needs to replace the identified data-intensive functions with the associated vector instructions, minimizing the effort of programming and optimizing. Fig. 4 shows an example code of a partial motion estimation function.

```
for (y=0, abort_search=0; y<blocksize_y && !abort_search; y+=4)
{
  for(x=0; x<blocksize_x; x+=4)
  {
    // NAC
    __asm__ __volatile__ (
            "add/15:0(11)  $10, $8, $9\n\t"          // nac.i $vr10, $vr8, $vr9
            :"=r"(pOrig_pic), "=r"(pRef_pic)
            :"r"(img_width), "r"(img_height),"r"(cand_x),"r"(cand_y),
              "r"(x),"r"(y),"r"(orig_pic[y]),"r"(ref_pic)
            :"8", "9", "10"
    );

    // 4x4SAD
    __asm__ __volatile__ (
            "add/15:0(3)   $3, %1, %3\n\t" // ll4x4.pi $pr3, pOrig_pic, 16
            "add/15:0(3)   $4, %2, %4\n\t" // ll4x4.pi $pr4, pRef_pic, img_width
            "add/15:0(1)   %0, $3, $4\n\t"  // sad.pi   t_mcost, $pr3, $pr4
            :"=r"(t_mcost)
            :"p"(pOrig_pic), "p"(pRef_pic), "r"(16), "r"(img_width)
            :"3", "4"
    );
    mCost += t_mcost;
  }
}
```

Fig. 4. An example code of H.264 motion estimation using vector instructions.

4. PERFORMANCE EVALUATION

To verify and evaluate the performance of the presented implementation of the H.264 motion estimation function, this work employs the SimpleScalar simulator, which is developed and supported by T. Austin at SimpleScalar LLC [13].

SimpleScalar is an execution-driven simulator and supports an out-of-order executable and user-extensible instruction format [14]. Therefore, our identified vector instructions and the out-of-order execution scheduler are well incorporated into this simulator. As shown in Fig. 4, JM 10.1 [15] is modified by using newly added vector instructions. Additionally, to verify the performance of the presented implementation, this paper defines three test conditions and performs them in the SimpleScalar simulator. Under the first condition, motion estimations are performed using an in-order execution scheduler without vector instruction sets. Under the second condition, motion estimations are performed using an in-order execution scheduler with a vector instruction set. The third condition uses an out-of-order execution scheduler with a vector instruction set. In a simulation

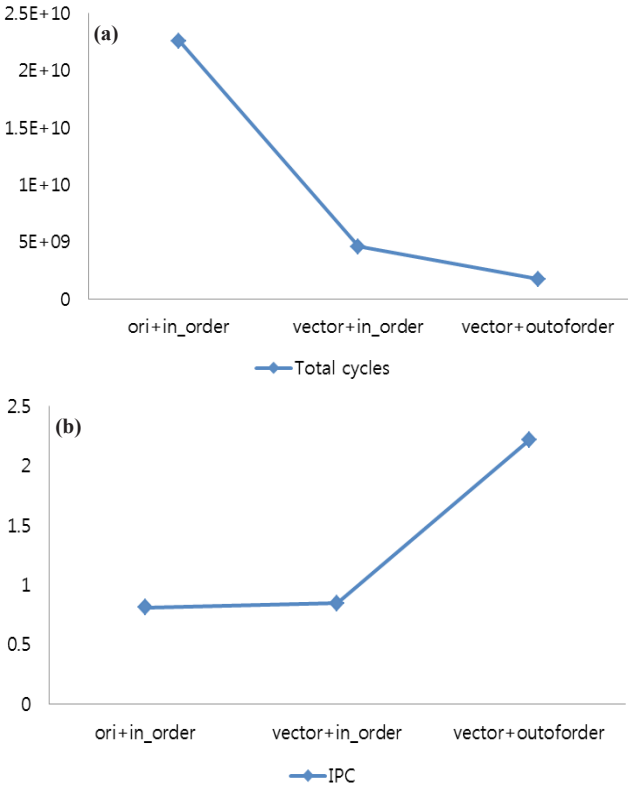

Fig. 5. Simulation results of H.264 motion estimation function (a) Total cycles of motion estimations under three conditions and (b) IPCs of motion estimations under three conditions.

process, this work assumes that the vector instructions defined in the previous chapter are executed in one clock cycle because of their dedicated hardware units.

Additionally, the following features are assumed: (1) 4:2:0 YCb-Cr format and QCIF (176×144) resolution, (2) full search algorithm, (3) 16 pixel motion vector search range, (4) 1/4 pixel motion vector resolution per one macroblock, (5) matching criteria: SAD for integer-pixel search, and SATD for sub-pixel search, and (6) 5 reference frames.

The simulation results of the presented implementation of H.264 motion estimation are given in Fig. 5. In the original H.264 with an in-order issue structure, the total cycles are 22.57 billion cycles, but after the partial function blocks of motion estimation are replaced by the proposed vector instructions, the performance is increased by 4.93 times under the same in-order execution scheduler. The main reason for this performance improvement is that the hardware supported vector instructions can process complicated and data-intensive modules effectively.

Additionally, as shown in Fig. 5(a), the out-of-order executable RISC with vector instruction sets is approximately 2.62 times faster than the in-order executable RISC with vector instruction sets even though both architectures use vector instructions (vector+outoforder: 1.7 billion cycles, vector+in_order: 4.5 billion cycles). This is because the out-of-order scheduler boosts up the performance of the system based on vector instructions by increasing exploited parallelisms in motion estimations. This is verified in Fig. 5(b). In Fig. 5(b), the IPCs of motion estimations under the three conditions are measured. Instruction Per Cycle (IPC) is a measure used to determine the amount of parallelism among instructions that is exploited. A higher IPC value means that the number of instructions (including vector instructions) performed per one cycle is higher, thus better exploiting the parallelism of the given algorithm. As can be seen, the out-of-order RISC with vector instruction sets increases IPC significantly compared to the in-order RISC processors (more than 2 times). This means that even processors with vector instruction sets still need to exploit out-of-order executions in order to increase performance.

5. CONCLUSION

To enable an efficient photovoltaic monitoring system for tracing a degradation process of the system, this paper presents a vector instruction-based RISC architecture and implements a H.264 motion estimation tool which is a key tool of the H.264 compression standard. This implementation is based on a vector instruction set and an out-of-order executable RISC architecture, which can take advantages of software and hardware design features, i.e., flexibility, low development cost, compactness, and adequately high performance.

By appropriately defining vector instruction sets, this work can increase the speed of H.264 motion estimations by up to 13 times while minimizing design efforts of both hardware and software. Additionally, since the vector instruction sets comprise a powerful codec function library, software engineers can easily use the vector instructions without fully understanding their specific hardware architecture to optimize the software implementation of a target application.

Additionally, this work can be extended to various compression algorithms efficiently and economically by properly identifying vector instructions and incorporating them into a generic RISC architecture with an out-of-order execution scheduler.

REFERENCES

[1] D. Sera, R. Teodorescu, P. Rodriguez, Partial Shadowing Detection based on Equivalent Thermal Voltage Monitoring for PV Module Diagnostics, IECON, 2009: 708-713 [DOI:http://dx.doi.org/10.1109/IECON.2009.5415006]

[2] L. Cristaldi, M. Faifer, M. Rossi, F. Ponci, Monitoring of a PV System, The role of the Panel Model, (IEEE International Workshop on Applied Measurements for Power Systems, 2011: 90-95[DOI:http://dx.doi.org/10.1109/AMPS.2011.6090437]

[3] G. Notton, V. Lazarov, L. Stoyanov, Optimal Sizing of a Grid-connected PV System for Various PV Module Technologies and Inclinations, Inverter Efficiency Characteristics and Locations, (Renewable Energy 35, 2010:.541-554 [DOI:http://dx.doi.org/10.1016/j.renene.2009.07.013]

[4] ISO/IEC 13818-2: “Information technology ? Generic coding of moving pictures and associated audio information: video,” 1996.

[5] ISO/IEC 14496-2: “Information technology ? Coding of audio-visual objects- part2: Visual,” 1999

[6] ISO/IEC 14496-10: “Coding of Audiovisual Objects-Part 10: Advanced Video Coding,” Dec. 2003.

[7] M. Irfan, A. K. Khan, and H. Jamal, FPGA based implementation of MPEG-2 compression algorithm, (IEEE 17th Int. Conf. On Microelectronics, Dec. 2005:204-244 [DOI: http://dx.doi.org/10.1109.ICM.2005.1590075]

[8] K. Denolf, C. D. Vleeschouwer, R. Turney, G. Lafruit, and J. Bormans, IEEE Trans. On Circuits and Systems for Video Technology, 2005 15(5): 609-619 [DOI: http://dx.doi.org/10.1109/ TCSVT.2005.846430].

[9] T. Wiangtong, P. Y. K. Cheung, and Wayne Luk, IEEE Signal Processing Magazine, 2005, 22(3): 14-22 [DOI: http://dx.doi.org/10.1109/MSP.2005.1425894].

[10] S. D. Haynes, H. G. Epsom, R. J. Cooper, and P. L. McAlpine, UltraSONIC: A reconfigurable architecture for video image processing, (Proc. Field-Programmable Logic and Applications, 2002) pp. 482-491.
[11] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, IEEE Circuits and Systems Magazine 2004 (4): 7-28 [DOI: http://dx.doi.org/10.1109/MCAS.2004.1286980].
[12] R. M. Tomasulo, IBM Journal of Research and Development 1967 11(1): 25-33 [DOI: http://dx.doi.org/10.1147/rd.111.0025].
[13] SimpleScalar LLC web Site at http://www.simplescalar.com/
[14] T. Austin, E. Larson, and D. Ernst, IEEE Computer Society Magazine, 2002 35(2): 59-67[DOI: http://dx.doi.org/10.1109/2.982917].
[15] Reference Software: available at http://iphome.hhi.de/suehring/tml/