

A Register File with Transposed Access Mode

Yoochang Jung, Stefan G. Berg, Donglok Kim, and Yongmin Kim
Image Computing Systems Laboratory (ICSL)
University of Washington
Box 352500 Seattle, WA 98195-2500
{mrchang, sgberg, dong, ykim}@icsl.ee.washington.edu

Abstract

We introduce a new register file architecture that provides both row-wise and column-wise accesses, thus allowing partitioned instructions to be used in column-wise processing without transposition overhead. This feature can accelerate 2D separable image and video processing algorithms, such as 2D convolution and 2D discrete cosine transform (DCT), by eliminating the transposition steps.

1. Introduction

2D convolution and 2D transforms, such as wavelet transform and discrete cosine transform (DCT), are widely used in image and video processing. To reduce the computational complexity, these algorithms are often implemented in two separable passes of 1D processing (e.g., row-wise processing followed by column-wise processing). Many image and video processing algorithms handle data elements that are smaller than a register size. Mediaprocessors take advantage of this property by employing partitioned instructions that can simultaneously process multiple data elements packed into one register [1]. In performing the two separable passes of 1D processing, these partitioned instructions can be useful for row-wise processing. However, since the data are not stored in consecutive memory locations for column-wise processing, a transposition on the row-wise processing result before entering the column-wise processing and another transposition on the column-wise processing result have to be applied to utilize the partitioned instructions. Our new register file architecture can eliminate the transposition steps by providing both row-wise and column-wise accesses.

2. A transposable register file

Figure 1 shows a transposable register file with 8-bit partitions. Each register consists of n 8 bits of data and therefore has a total width of $n \times 8$ bits. This register file contains n registers to provide $n \times n$ transposition. There are two access modes: normal and transposed.

The register blocks are accessed row-wise in the normal access mode, while column-wise access is used in the transposed access mode. Note that we need n registers to transpose an $n \times n$ 8-bit block of data. In image and video processing, 16-bit data are also frequently used as well as 8-bit data. Figure 2 shows a transposable register file that allows 16-bit column-wise access. In this case, we need only $n/2$ registers (n should be even) compared to n registers in the 8-bit transposable register file since there are $n/2$ 16 bits of data in a register.

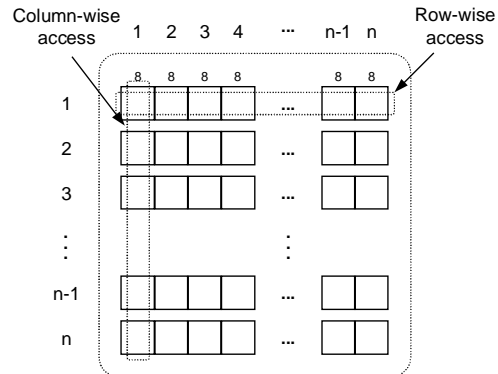


Figure 1. A transposable register file with 8-bit partitions.

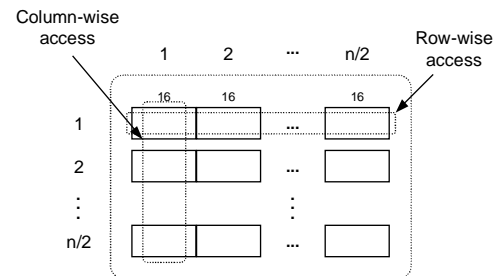


Figure 2. A transposable register file with 16-bit partitions.

Transposition can be done by either a normal write followed by a transposed read or a transposed write

followed by a normal read. In this paper, we use the transposed write – normal read mode since it generally requires less hardware overhead, i.e., the transposition requires additional hardware for each port that supports it and there are fewer write ports than read ports on typical register files, (e.g., there is one destination operand field and two or three source operand fields in typical instruction sets).

3. Discussion

We have implemented three examples using MAP1000 [2] instructions to see the effect of the transposable register file. Image transposition is a common example that can be used in many 2D separable algorithms. An 8 x 8 16-bit block DCT example shows that the two separate transposition steps can be completely hidden when all the data fit in the register file. A 2D separable convolution example shows that this architecture is useful for the algorithms that can be processed in a block-wise fashion. Table 1 compares the number of instructions taken in the examples. The reduction in the number of instructions due to the transposable register file is significant with the factors ranging 1.4 to 2.5 compared to the regular methods.

Table 1. Comparison of the number of instructions

Examples	Without transposable register file	With transposable register file	Ratio
8x8 8-bit transpose	40	16	2.50 : 1
8x8 16-bit DCT	200	136	1.47 : 1
42x42 16-bit 2D separable convolution with a kernel length of 3	5922	4242	1.40 : 1

There are three aspects to the hardware cost for our technique of transposing registers. First, each transposable access port requires its own decoder and additional wiring within the register file structure. Second, the ability to transpose on different data widths, such as 8 bits and 16 bits, requires separate transposable access ports for each data width. Finally, the instruction set architecture will need to be able to address additional registers in its instruction to include the newly added transposed registers.

A transposable register file can hide the time-consuming transposition steps in many fundamental algorithms and allow us to explore various implementation methods. However, it increases the

need for more number of registers when the data width increases since the required number of registers for transposition is proportional to the number of partitions in a register. For example, in a 64-bit architecture, we need 8 registers to transpose an 8 x 8 8-bit data block. For 128-bit architecture, we need 16 registers to transpose a 16 x 16 data block. These 16 registers cannot be used for other purposes until the transposition is completed. In addition, when there is a long latency between writing and reading the register file caused by the processor pipeline, we cannot start reading the transposed data right after issuing an instruction that writes a result to the register file. These issues and implementation details are further discussed in the complete paper [3].

4. Conclusion

We have presented a transposable register file architecture for reducing the number of instructions for transposing a block of data. This technique allows us to access the data in a register file in both row-wise and column-wise directions. It requires no extra instructions in accessing the transposed data in the register file. For example, the results of row-wise processing can be stored in a transposed fashion and directly used as source operands in the subsequent column-wise processing. The transposable register file requires many registers during transpositions especially on wide data path architecture, thus could increase the register pressure. We have shown three examples that demonstrate the effect of the transposable register file in image and video computing algorithms, which resulted in the reduced number of instructions by factors ranging 1.4 to 2.5 compared to the regular implementation methods.

5. References

- [1] S. Rathnam and G. Slavenburg, "Processing the new world of interactive media," *IEEE Signal Processing Magazine*, vol. 15, no. 2, pp. 108-117, 1998.
- [2] C. Basoglu, R. Gove, K. Kojima, and J. O'Donnell, "A single-chip processor for media applications: the MAP1000," *International Journal of Imaging Systems and Technology*, vol. 10, pp. 96-106, 1999.
- [3] ICSL, <http://icsl.ee.washington.edu/papers>