RC-004

三次元アレイプロセッサ構造を用いた効率的な 3D-DCT 計算機構の提案
Effective 3D-DCT Calculation Based on a 3D Array Processor

生垣　佑樹†　　武石　直人‡　　宮崎　敏明†　　Stanislav G. Sedukhin†
Yuki Ikegaki　Naoto Takeishi　Toshiaki Miyazaki　Stanislav G. Sedukhin

Abstract

Traditional array processors randomly access to input/ coefficient data stored in memory many times during the three dimensional discrete cosine transform (3D-DCT) calculation. Hence, it becomes a bottleneck of fast calculation. In this paper, a three dimensional array processor dedicated to 3D-DCT is proposed. The array processor tremendously reduces the data swapping or replacement during the calculation. Thus, it contributes to the performance improvement. The computational complexity of the proposed array processor is O(N) for an $N\times N\times N$ input data cube while that of the 3D-DCT direct calculation is O(N^4). A specified I/O architecture and throughput/cost-effective architectures are also discussed for practical implementation. Experimental results of an FPGA (Field Programmable Gate Array) implementation show that our architecture has enough performance for real-time 3D-DCT calculation on its scalable architecture.

1. Introduction

The advancement of digital imaging applications like high-definition television, teleconference, medical and space exploration images compactions [1], and portable video player accelerates the demands for effective image compression techniques. Prospective systems such as portable video chatting require high performance computing to realize video compression, because the transmission bandwidth is often limited. The 2D-DCT (two dimensional discrete cosine transform) is a technique for spatial information compression, in 2D images [2]. The 3D-DCT extends the DCT energy compaction properties to integral 3D images and to the spatio-temporal coding of 2D video sequences [2, 3]. The image data could be compressed to 1/10 sizes by 3D-DCT transform without dropping of its image quality [4]. This paper will discuss an array processor that performs 3D-DCT effectively on the 3D-LSI. The block diagram of the 3D-DCT/Inverse DCT (IDCT) video codec is schematically shown in Figure 1.

Compared to motion-estimation/compensation based methods, the 3D-DCT approach has three essential factors for portable video compression systems:

- No motion estimation is needed, hence, it greatly decreases the number of en/decoding operation per pixel compared to a motion-estimation/compensation based approach.
- The encoder and decoder are composed of an equivalent architecture; coefficient data stored in register are different.

† 会津大学大学院コンピュータ理工学研究科, Graduate School of Computer Science of Engineering, University of Aizu
‡ 富士通株式会社, FUJITSU LIMITED

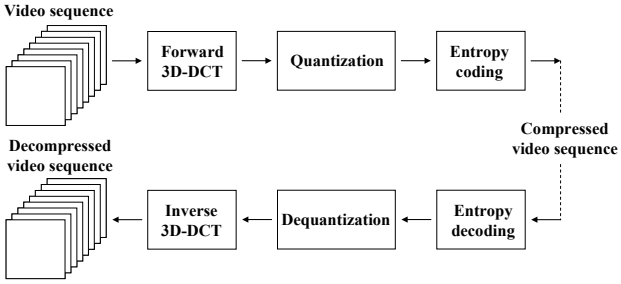

Figure 1: Block diagram of the 3D-DCT/IDCT video codec

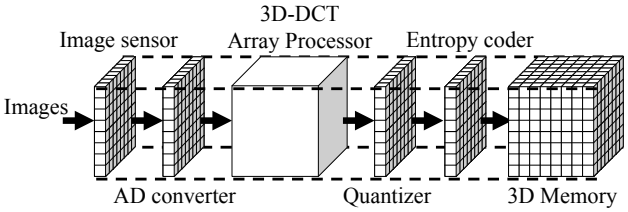

Figure 2: Array processor implementation image on a 3D-LSI

- There is no relationship between the complexity of the implementation and the compression ratio. [5]

Direct calculation of 3D-DCT requires a long time so that a real-time application cannot be used because we have to randomly access coefficients and input data stored in memories many times. Thus, researchers have proposed many algorithms and hardware architectures for rapid calculation of 3D-DCT, especially for the real-time applications. In [6] and [7], a parallel algorithm for 3D-DCT computation based on butterfly calculation is proposed. The butterfly calculation and recursive addition is carried out in log N steps [7]. However, it is difficult to design the 3D butterfly interconnection into hardware architecture, because of its complex structure, when N is large. In this research, a more practical array processor dedicated to 3D-LSI is proposed. As shown in Figure 2, by using the 3D-LSI, we will be able to connect planes of image sensor array, processors, memory module, and other devices. Our array processor has $N\times N\times N$ toroidal-cube-connected PEs (Processing Elements). Each PE has a homogeneous structure: a MAC (multiply-accumulation) unit, a register-file, and wires connected to adjacent PEs. Compared to the other array processors, our array processor can tremendously reduce the data swapping or replacement during the 3D-DCT calculation by introducing a smart data transfer scheme with a simple PE array structure. This is a key to realize a high-speed 3D-DCT. In addition, the trend of three-dimensional large scale integration (3D-LSI) technologies [8] could support to realize our 3D array processor that can directly handle cubical (3D) data streams.

Section 2 shows the definition formula of 3D-DCT. In Section 3 and 4, the array processor architecture and its I/O interfaces for 2D devices are proposed. Throughput/area improved architectures are shown in Section 5 and 6. Some experimental results and evaluations are presented in Section 7. Conclusions are discussed in Section 8.

2. Definition formula of 3D-DCT

Let $X_{N\times N\times N} = [X(i,j,k)]$, $0 \le i,j,k \le N\text{-}1$, be an input data cube. The 3D type-II discrete cosine transform of $X(i,j,k)$, and a data cube $Y_{N\times N\times N} = [Y(s,r,p)]$, $0 \le s,r,p \le N\text{-}1$, can be defined as:

$$Y(s,r,p) = \Phi \cdot \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} X(i,j,k)\times C(k,p)\times C(j,r)\times C(i,s),$$

where

$$\Phi = \sqrt{\frac{8}{N^3}}\cdot\varepsilon(s)\cdot\varepsilon(r)\cdot\varepsilon(p),$$

$$\varepsilon(m) = \begin{cases} \dfrac{1}{\sqrt{2}}, & for \quad m = 0; \\ 1, & for \quad m > 0; \end{cases} \qquad m \in \{s,r,p\},$$

and elements of the coefficient matrix $C_{N\times N} = [C(u,v)]$ are pre-computed as

$$C(u,0) = \frac{1}{\sqrt{2}}, \quad C(u,v) = \cos\left(\frac{\pi(2u+1)v}{2N}\right),$$

$$u \in \{i,j,k\}, (u,v) \in \{(i,s),(j,r),(k,p)\}.$$

The 3D-DCT is a linear transformation that converts an original 3D coordinate system (i,j,k) into a new 3D system, (s,r,p). Coefficient matrix $C(k,p) = C(j,r) = C(i,s) = C(u,v)$ is used for converting k-, j-, and i-axis into p-, r-, and s-axis.

3. Array Processor Architecture for 3D-DCT

3.1 Overview

The structure of PE used in our array processor is shown in Figure 3. For $N\times N\times N$ 3D-DCT, each PE is mainly composed of three I/O selectors, a MAC unit, and $3N$+4 registers. Each selector is controlled by a *state* signal and synchronized with clock signal. $3N$ registers are for coefficients, 4 registers are for input data and calculation results.

Figure 4 overviews our array processor architecture. Each PE is basically connected with adjacent PEs; the end PEs have torus connections. The data transfer direction is shown in Figure 5. All PE has homogeneous structure and performs systolically. The sequence to acquire the result of $N\times N\times N$ 3D-DCT is as follows: data input, step1, 2, and 3 for 3D-DCT calculation, and data output. On the traditional array processors, the data swapping or replacement during the 3D-DCT calculation was performance bottleneck, because of its high-speed operations and low-speed memory data access. This proposed array processor keeps the input data integrity and locality, and there is no data re-arrangement during its 3D-DCT calculation. This is a significant feature to improve the performance.

3.2 Data Input/Output

This array processor inputs $N\times N$ data simultaneously. The input ports are arranged on $k = N\text{-}1$ plane; each PE transfers

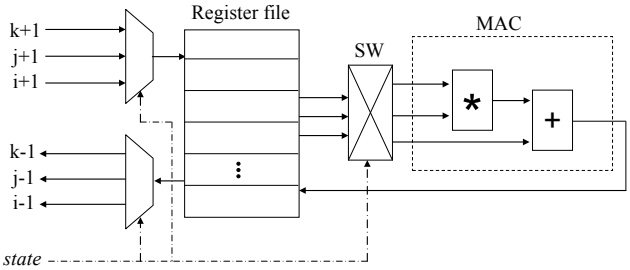

Figure 3: PE architecture for 3D-DCT array processor. Each selector is controlled by *state* signal and synchronized with clock signal, 1) Receive data from adjacent PE and send it to destination register, 2) Perform multiply-add operation, 3) Send data stored in register to adjacent PE

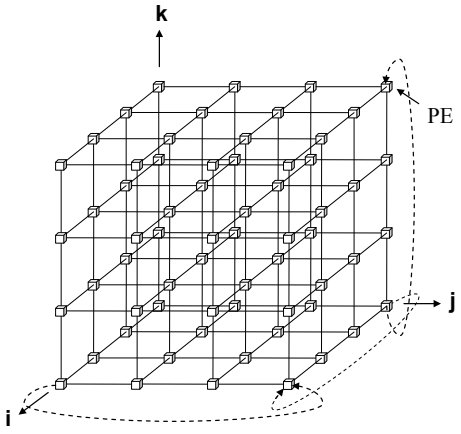

Figure 4: Overview of the array processor architecture for 4×4×4 3D-DCT (Control unit is not shown)

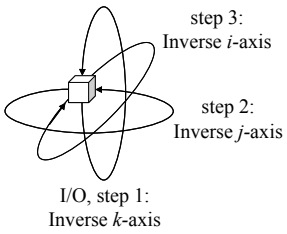

Figure 5: Direction of the data transfer

input data to $k\text{-}1$ adjacent PE, along k-axis. N-time process needs to input all of the $N\times N\times N$ data. The output sequence is almost the same as the input. The output data of PEs at $k = 0$ plane are fed to the corresponding PEs at $k = N\text{-}1$ plane using the torus connections. The PEs at $k = 0$ plane also has output ports, and final results are obtained using the output ports. The partial I/O connections of this array processor are shown in Figure 6.

3.3 3D-DCT Calculation

As shown in Section 2, 3D-DCT is a heavy task that requests many MAC operations, and it is difficult to realize a hardware circuit to perform the 3D-DCT directly. However, since 3D-DCT is a separable transform, we can implement it as three 1D transforms and reduce the hardware cost [9]. In our array processor, there are three steps to obtain the 3D-DCT result. To perform 3D-DCT with ordinary single or array processors,

- the coefficients and input data stored in memory have to be accessed many times, and/or
- some rearrangements of the partial results are required whenever each 1D transform is finished.

These processes are significant performance bottlenecks for the fast 3D-DCT calculation. By pre-storing the 3D-DCT coefficients in each PE, and moving the partial calculation results appropriately using the torus connections between PEs, our array processor can completely eliminate the above mentioned data rearrangement to obtain the 3D-DCT results. This is a key issue why our array processor can perform 3D-DCT effectively.

In the first step, multiplication of $X(i,j,k)$ and $C(k,p)$ is performed. Then, $X(i,j,k)$ is multiplied with $C(j,r)$ at the second step. Finally, multiplication of $X(i,j,k)$ and $C(i,s)$ is done at the third step. Each step has N micro-steps. The processing sequence of PE in every micro-step is as follows:

- First, receive data from adjacent PE and store it to a register.
- Second, multiply received data with a coefficient which is pre-stored in registers on each PE. Then, the calculation result is accumulated into a register.
- Finally, transfer received data to adjacent PE.

Transferred data in each step is:

- input data, for the 1st step, or
- the calculation result of 1st step, for the 2nd step, or
- the calculation result of 2nd step, for the 3rd step.

For each micro-step, the coefficient data used in PE(i,j,k) is

- $C((k+microstep) \bmod N, k)$ for the 1st step, or
- $C((j+microstep) \bmod N, j)$ for the 2nd step, or
- $C((i+microstep) \bmod N, i)$ for the 3rd step.

The coefficients are preset in each PE before processing. For 3D-IDCT, data of transposed coefficient matrix ${}^tC$ is used and the matrix element is preset on each PE.

The total amount of calculation time-steps of the proposed array processor is $3N$ for an $N\times N\times N$ size input data cube. Thus, its computational complexity becomes of $O(N)$ while that of the 3D-DCT direct calculation is $O(N^4)$.

4. I/O Interfaces

The I/O interface can adapt any $N\times N\times N$ array processor tuned to 3D-DCT. The I/O parts are connected to a surface of k-axis of PE array. I/O interfaces are provided to implement our array processor on 2D devices. Figure 7 overviews this I/O architecture. They consist of six parts: an input memory, input address generator, input buffer, output buffer, output address generator, and output memory. As shown in Figure 7, the input architecture performs following operations in each clock cycle:

- Derive input memory address by using input address generator.
- Get one pixel data from input memory and put it into proper FIFO of the input buffer.
- Update some control signals for the input buffer, address generator, and array processor.

In addition, the input buffer sends sub-frame data structured by $N\times N$ pixels to the array processor simultaneously when the array processor requests data for the next calculation, and the input

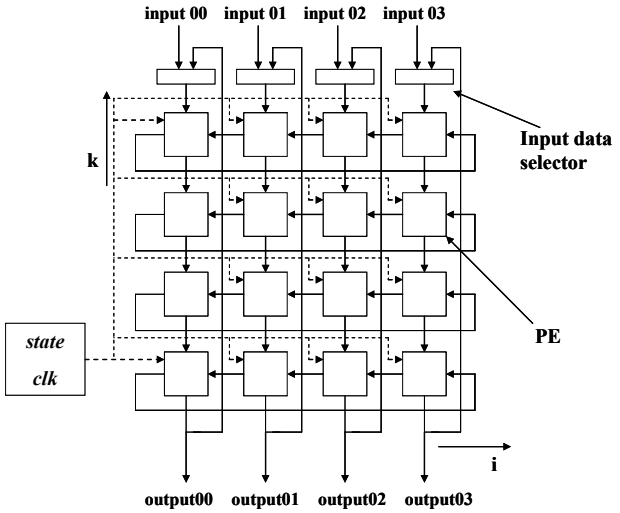

Figure 6: One of the *i-k* planes' sliced view

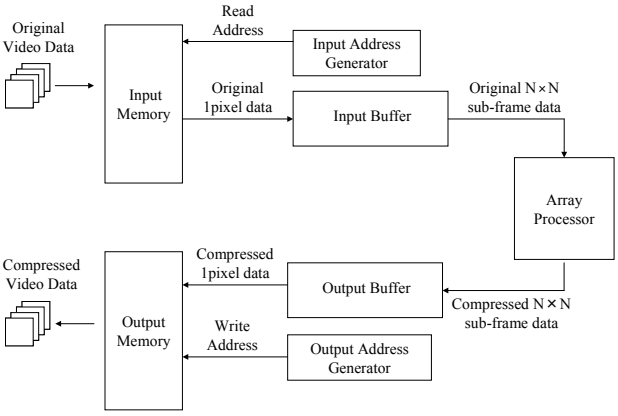

Figure 7: Block diagram of I/O interface architecture

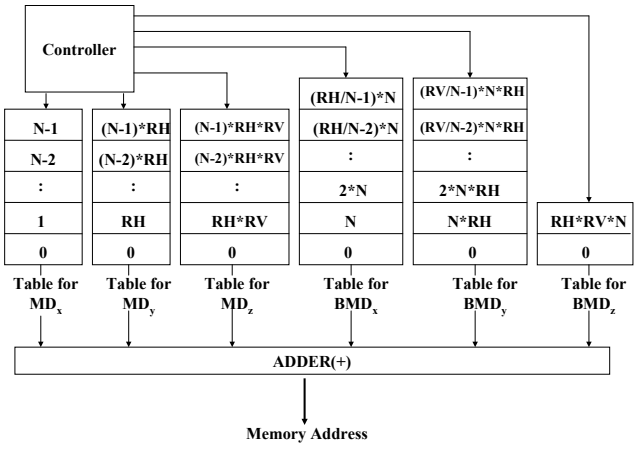

Figure 8: Overview of address generator

memory gets N frames data from the host computer at appropriate timing.

The input memory is used as a frame buffer for the input data. This memory is connected to a host computer and the input buffer. The video data from the host computer are stored in this memory sequentially. On the other hand, output data of the input memory is managed by input address generator to generate $N\times N$

pixels sub-frame data. Input memory has 8 bits data word and enough capacity to contain $N\times2$ frames video data. The data sent to array processor is overwritten by new input data. The array processor is able to begin its operations after the first N-frame video data arrived to input memory.

The input address generator produces memory address for the input memory access in each clock cycle. It requests the following four parameters to work: base address (BA), vertical resolution of video data (RV), horizontal resolution of video data (RH), and block size of sub-frame image data (N). As shown in Figure 8, the address generator has six address tables for address calculation and an adder. Contents of each address tables are calculated by following expressions:

$MD_x[i] = i$, for $0 \le i < N$

$MD_y[i] = RH\times i$, for $0 \le i < N$

$MD_z[i] = RH\times RV\times i$, for $0 \le i < N$

$BMD_x[i] = N\times i$, for $0 \le i < RH/N$

$BMD_y[i] = RH\times N\times i$, for $0 \le i < RV/N$

$BMD_z[i] = RH\times RV\times N\times i$, for $0 \le i < 2$

Here, MD_x, MD_y, and MD_z are distances on each axis and used to generate $N\times N\times N$ data block for the array processor. BMD_x, BMD_y and BMD_z are distances of data block and used to control data block flow.

The input address generator selects a proper value from these tables to generate a memory address. Here, the memory address is easily obtained by the following expression:

$$ADDR = BA + MD_x + MD_y + MD_z + BMD_x + BMD_y + BMD_z$$

Here, *ADDR* is the target memory address for the input memory access.

The input buffer is used for timing gap absorption between the array processor and the input memory. Figure 9 shows the input buffer in detail. It has $N\times N$ asynchronous FIFOs and a buffer controller. The buffer controller has three control signals: *write-enable* for each FIFO, *full* for the input address generator, and *read-request-enable* for the array processor. The *write-enable* signal controls writing in FIFOs to store $N\times N$ pixels sub-frame data correctly. The *full* signal is asserted when all FIFOs become full. The input address generator doesn't update the input memory address when *full* is asserted. The *read-request-enable* signal is asserted when FIFOs have enough data for the next 3D-DCT calculation. The array processor stops input operations until *read-request-enable* is asserted. The input buffer receives one pixel data from the input memory and put it into proper FIFO to generate a set of the pixel data for one video sub-frame containing $N\times N$ pixels. On the other hand, the input buffer sends the sub-frame data for the next 3D-DCT calculation. The *read-request* from the array processor is directly connected to each FIFO to request the next data to the FIFO.

The data flow scheme of the output architecture is opposite to that of the input architecture. The output buffer is used for timing gap absorption between the output memory and the array processor output. Figure 10 overviews the output buffer architecture. The output buffer receives the data for one video sub-frame from the output ports of the array processor simultaneously and sends them to proper locations of the output memory using a selector. As shown in Figure 10, the buffer controller generates four control signals for the output buffer.

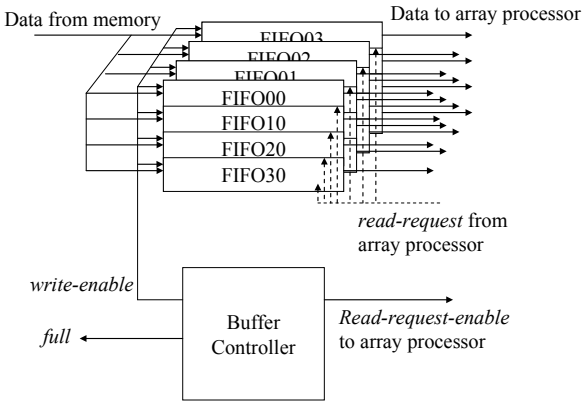

Figure 9: Overview of input buffer

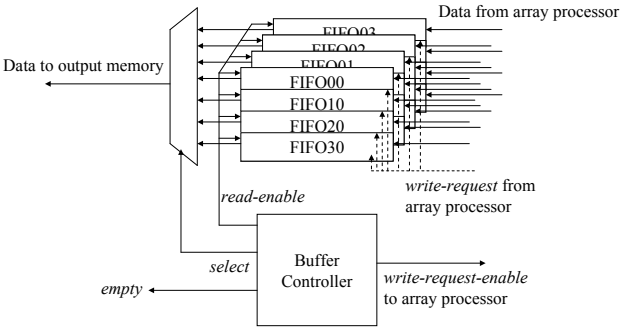

Figure 10: Overview of output buffer

The *select* and *read-enable* signals are used to control output data of FIFO. The *write-request-enable* signal is asserted when each FIFO have enough capacity to receive N frames data from the array processor. The array processor is able to send data only when this signal is asserted. The *empty* signal is asserted when all FIFOs become empty, and connected to *write-enable* port of the output memory and the output address generator. The data in the output memory and the output memory address are updated only when *empty* is negated.

The output address generator generates the memory address for the output memory access. The processing sequence of the output address generator is the same as input address generator.

The output memory stores the output data from the array processor. The data allocations in the output memory are the same as those of the input memory. The data from the output buffer is stored to the proper locations of the output memory generated by the output address generator. Moreover, the output memory sends N-frame data to the host computer simultaneously at a proper timing.

By using this proposed I/O interfaces, an effective I/O data management is realized without any data stream modifications in the host computer.

5. Throughput Improvement

To improve the throughput of the array processor shown in the previous section, two pipelined architectures featuring data pipelining mechanisms are newly proposed. Although the improved architectures request more hardware resources compared to the original one, their performance improvement is

significantly expected.

The first version of the pipelined architecture requires the following hardware resources:

- ×2 MACs
- Input ports are arranged on $i = N-1$ plane; each PE transfers input data to $i-1$ adjacent PE, along i-axis

The pipeline scheme is shown in Figure 11. Second version of pipelined architecture requires the below items:

- I/O specified wires among PEs
- I/O operations are separated from calculation
- ×3 MACs
- 3N+4·3 registers

The pipeline scheme is shown in Figure 12. Pipelined ver.1 architecture performs 3D-DCT calculation ×2.5 faster than sequential architecture, theoretically. Pipelined ver.2 architecture performs ×5.0 faster than sequential architecture, however, it costs much larger than ver.1. In addition, its structure becomes more complex, especially for wiring.

Stage A Direction	Input i	1st step k	2nd step j	3rd step i	Output k		Input i	1st step k	2nd step j
Stage B Direction			Input i	1st step k	2nd step j	3rd step i	Output k		Input i
Stage C Direction					Input i	1st step k	2nd step j	3rd step i	Output k

Figure 11: Pipeline scheme for pipelined ver.1
top: process in each stage / bottom: data transfer direction

Stage A Direction		1st step k	2nd step j	3rd step i	1st step k	2nd step j	3rd step i	1st step k	2nd step j
Stage B Direction			1st step k	2nd step j	3rd step i	1st step k	2nd step j	3rd step i	1st step k
Stage C Direction				1st step k	2nd step j	3rd step i	1st step k	2nd step j	3rd step i
Input	Stage A	Stage B	Stage C	Stage A	Stage B	Stage C	Stage A	Stage B	Stage C
Output	-	-	-	-	Stage A	Stage B	Stage C	Stage A	Stage B

Figure 12: Pipeline scheme for pipelined ver.2
top: process in each stage / bottom: data transfer direction; input: input data for Stage X, Output: output data of Stage X

6. Area Improvement

Since our array processor needs $O(N^3)$ hardware resources, decreasing them, especially multiplier, is significant issue for implementation. Figure 13 shows the number of register and operator in our sequential/pipelined architecture in the case of N = 2, 4, and 8. If we could perform N^3 size 3D-DCT by $(N/2)^3$ size architecture, the number of operator decreases exponentially.

Assume N=8, 4×4×4 array processor architecture performing 8×8×8 3D-DCT is shown in Figure 14. The features of this architecture are as follows:

- 8×8×8 3D-DCT calculation is divided into 8 4×4×4-sized 3D-DCT calculation.
- 4×4 FIFO are inserted into each toroidal-connection of i-, j-, and k-axis; each FIFO stores 4 data.
- 3×8 time-steps is needed for 4×4×4-sized 3D-DCT calculation.
- Number of operator is 4×4×4 (1/8).

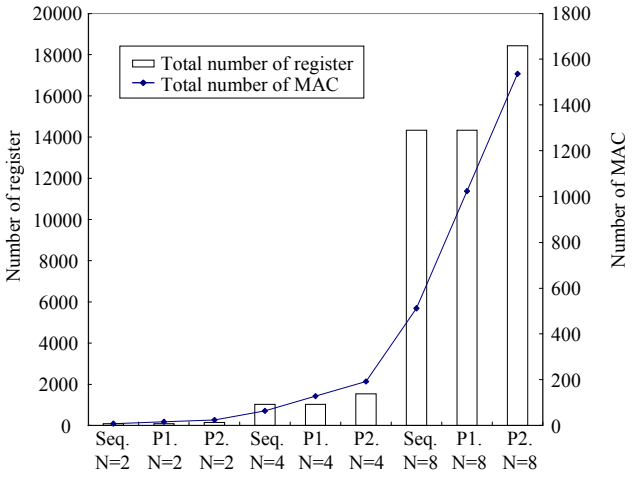

Figure 13: Number of registers and MAC
Seq: Sequential, P1: Pipelined ver.1, P2: Pipelined ver.2

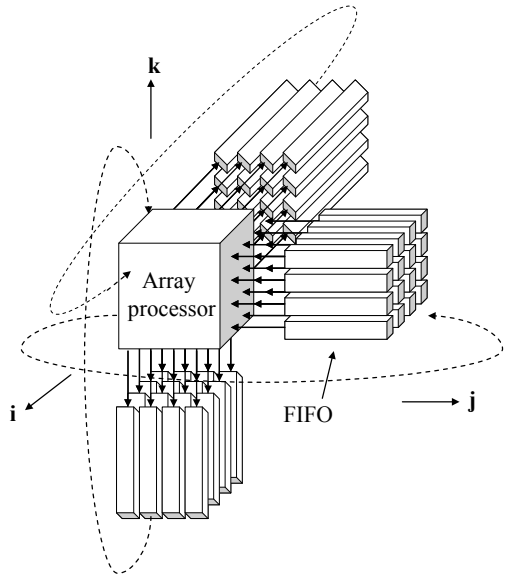

Figure 14: 8×8×8 3D-DCT calculation by using 4×4×4 array processor

- Number of coefficient registers for each PE is same as 8×8×8 array processor architecture.
- Number of I/O data is $4^3 \times 4$ to derive 4×4×4-sized 3D-DCT calculation result.

Compared with the architectures proposed in Section 3, the number of multiplier/adder is exponentially reduced. Thus, it makes easy to implement into area-limited hardware such as FPGAs (Field Programmable Gate Arrays).

7. Experimental Results

7.1 Array Processor

First, we evaluate the theoretical minimum operating frequencies to perform real-time 3D-DCT/IDCT analytically. We assumed N = 8, one micro-step operates in 2 clocks, and input video data is single color. For fps (frame per second) = 32 and 60, the minimum operating frequencies of proposed architectures are shown in Table 1.

For the VGA-size input video data, the minimum operating frequency of:

- sequential architecture is less than 200/360 KHz,
- pipelined version 1 architecture is less than 80/150 KHz,
- pipelined version 2 is less than 40/75 KHz, when fps = 32/60; this is considered one of the frequently-used video data formats. The shown frequency values are very low, thus, it affords low-power consumption.

For the UXGA-size input video data, the minimum operating frequency of:

- sequential architecture is less than 2.3 MHz,
- pipelined version 1 architecture is less than 1.0 MHz,
- pipelined version 2 is less than 0.5 MHz, when fps = 60; this is considered one of the high-quality video formats. The frequency values above are only a few MHz. Playing high-quality video on the portable devices is one of the upcoming demands, so this architecture will be useful in such applications.

To perform 3D-DCT/IDCT for RGB-color video, three times of shown operating frequency is required. 4×4×4 array processor architecture performing 8×8×8 3D-DCT proposed in Section 5 needs 8 times the frequency.

To estimate and evaluate the actual hardware cost of our architecture, it is implemented to a Xilinx FPGA; we used "Xilinx ISE 9.2i" as a logic synthesis tool and “Virtex-5 LX330” FPGA as an implementation target for this purpose. The device utilizations are shown in Table 2. Here, we used the embedded multipliers on the Virtex-5 FPGA to realize the multiplications for 3D-DCT. Because the number of embedded multipliers is limited on the FPGA, the sequential architecture for $N = 4$ is only examined. As shown in Table 2, our architecture does not require large hardware resources. We can easily estimate the logic utilization for larger N, because it will increase almost sequentially: for 2 times of N, logic increases 8 times.

7.2 I/O Interfaces

In this section, we evaluate performance of the I/O architecture. We selected 4×4×4 sequential version array processor and 60 fps video data as a target, because this combination demands one of the highest operating frequencies to I/O architecture. The required operating frequencies and simulated operating frequencies are shown in Table 3: first row shows size of video data, and first column shows description of each row. The implementation target is also “Virtex-5 FPGA”, and logic synthesis tool is “Xilinx ISE 9.2i”. As shown in Table 3, operating frequencies are much higher than required operating frequencies, so it has enough performance for real-time 3D-DCT transform; although it requires higher operating frequency than array processor’s operating frequencies, because of the Von-Neumann bottleneck. In other words, if a 3D-LSI device is our implementation target, its performance becomes dramatically increase.

Next, we evaluate logic utilization of I/O architecture. Figure 15 shows logic utilizations and required memory capacitances for each resolution size. As shown in this figure, only 1.2 % registers and 2.5 % LUTs are required at a maximum. Thus, this I/O architecture is cost-effective for registers and LUTs. However, required memory capacity severely increases when resolution size becomes large. Thus, some external memory modules will be needed if the implementation target does not have enough memories.

8. Conclusions

This paper proposed a three dimensional array processor

Table 1 – Minimum operating frequency for real-time en/decoding; N=8, Seq: Sequential, P1: Pipelined ver.1, P2: Pipelined ver.2

fps = 32	Size	Min. freq. for real-time en/decoding (Seq)	Min. freq. for real-time en/decoding (P1)	Min. freq. for real-time en/decoding (P2)
QVGA	320x240	48.0 [KHz]	19.3 [KHz]	9.7 [KHz]
VGA	**640x480**	**192.0 [KHz]**	**76.9 [KHz]**	**38.5 [KHz]**
SVGA	800x600	300.0 [KHz]	120.1 [KHz]	60.1 [KHz]
XGA	1024x768	491.6 [KHz]	196.7 [KHz]	98.4 [KHz]
SXGA	1280x1024	819.2 [KHz]	327.7 [KHz]	163.9 [KHz]
UXGA	1600x1200	1,200.0 [MHz]	480.1 [KHz]	240.1 [KHz]
fps = 60	**Size**	**Min. freq. for real-time en/decoding (Seq)**	**Min. freq. for real-time en/decoding (P1)**	**Min. freq. for real-time en/decoding (P2)**
QVGA	320x240	90.0 [KHz]	36.1 [KHz]	18.1 [KHz]
VGA	**640x480**	**360.0 [KHz]**	**144.1 [KHz]**	**72.1 [KHz]**
SVGA	800x600	562.5 [KHz]	225.1 [KHz]	112.6 [KHz]
XGA	1024x768	921.6 [KHz]	368.7 [KHz]	184.4 [KHz]
SXGA	1280x1024	1,536.0 [KHz]	614.5 [KHz]	307.3 [KHz]
UXGA	**1600x1200**	**2,250.0 [KHz]**	**900.1 [KHz]**	**450.1 [KHz]**

Table 2 – Device utilization summary (estimated values) of sequential architecture, for $N = 4$; Target FPGA: Virtex-5 LX330, Logic synthesis tool: Xilinx ISE 9.2i

Logic utilization	Used	Available	Utilization
Number of slice registers	5,618	207,360	2%
Number of slice LUTs	10,132	207,360	4%
Number of fully used bit slices	5,027	10,723	46%
Number of bonded IOBs	293	1,200	24%
Number of DSP48Es	64	192	33%

Table 3 – Required operating frequencies for 8×8×8-size sequential array processor and operating frequencies

	Size	**Req. freq.**	Op. freq. (in)	Op. freq. (out)
QVGA	320 × 240	**5.78 [MHz]**	284.20 [MHz]	284.20 [MHz]
VGA	640 × 480	**23.01 [MHz]**	279.67 [MHz]	279.67 [MHz]
SVGA	800 × 600	**57.25 [MHz]**	280.15 [MHz]	280.15 [MHz]
XGA	1024 × 768	**93.48 [MHz]**	287.38 [MHz]	287.38 [MHz]
SXGA	1280 × 1024	**157.13 [MHz]**	288.75 [MHz]	279.52 [MHz]
UXGA	1600 × 1200	**223.78 [MHz]**	280.15 [MHz]	280.15 [MHz]

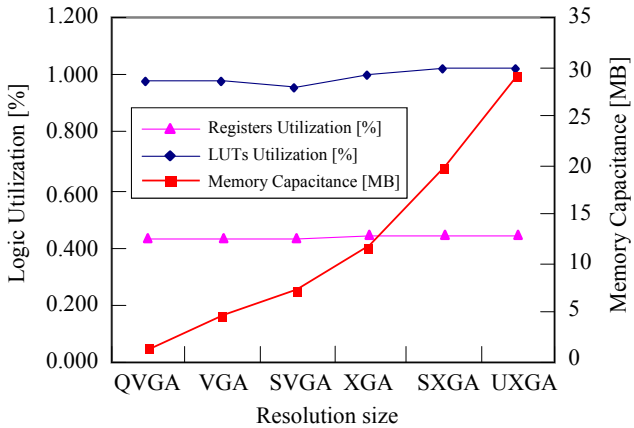

Figure 15: Logic utilizations and required memory capacitance

dedicated to 3D-DCT. The array processor tremendously reduces the data swapping or replacement during the 3D-DCT calculation. Thus, it must contribute to the performance improvement. The computational complexity of the proposed array processor is O(N) for an $N \times N \times N$ input data cube while that of the 3D-DCT direct calculation is O(N^4). In addition, more advanced architecture featuring data pipelining technique was proposed to improve throughput or hardware implementation costs. The pipelined architecture performs 3D-DCT calculation $\times 2.5$ or $\times 5.0$ faster than the basic architecture based on a sequential data handling. The $4 \times 4 \times 4$ array processor architecture performing $8 \times 8 \times 8$ 3D-DCT reduces its operators to 1/8 of original $8 \times 8 \times 8$ architecture. Furthermore, the proposed architecture was implemented with its specified I/O architecture that could adapt this three dimensional array processor to two dimensional devices and realized on an FPGA. The evaluation result shows that our architecture has enough performance for real-time applications.

9. References

[1] X. Li and B. Furht, “An approach to Image Compression Using Three-Dimensional DCT,” *Proc of the Visual 2003 Conference, Miami, Florida*, 2003

[2] S. Saponara, L. Fanucci, and P. Terreni, “LOW-POWER VLSI ARCHITECTURES FOR 3D DISCRETE COSINE TRANSFORM (DCT),” *Proc of the 46th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'03)*, Cairo, Egypt, vol. 3, pp. 1567-1570, 2003.

[3] T. Tsuchiya, K. Kirisawa, and H. Wakui, “A study of color motion picture coding using 3D-DCT,” *Proc of the IEICE General Conference 1996*, pp. 35, 1996.

[4] H. Morishita and Y. Ohno, “Volume Data Compression Using Three-Dimensional Discrete Cosine Transform,” *IPSJ SIG Notes 95 (63),* pp. 9-16, 1995

[5] A. Burg, R. Keller, J. Wassner, N. Felber, W.Fichtner “A 3D-DCT Real-Time Video Compression System for Low Complexity Single-Chip VLSI Implementation”, *Proc of the MoMuC2000*, pp. 1B-5-1, 2000.

[6] O. Alshibami, and S. Boussakta, “Fast Algorithm for the 3-D DCT,” *Proc of the IEEE International Conference on Acoustics, Speech, and Signal Processing 2001*, pp. 1945-1948, 2001.

[7] M. Modarressi, and H. Sarbazi-Azad, “Parallel 3-Dimensional DCT Computation on *k*-Ary *n*-Cubes,” *Proc of the 8th International Conference on High Performance Computing in Asia Pacific Region (HPC Asia 2005)*, pp. 91-97, 2005.

[8] M. Motoyoshi and M. Koyanagi, “3D-LSI technology for image sensor,” *The Pixel 2008 Workshop*, JINST 4 P03009, 2009.

[9] M. Servais and G. D. Jager, “Video Compression Using the Three Dimensional Discrete Cosine Transform (3D-DCT),” *Proc of COMSIG ’97, South African*, pp. 27-32, 1996.