IMPLEMENTATION AND EVALUATION OF CONDITIONAL STREAM IN STREAM PROCESSOR

SUI Bingcai, XING Zuocheng, MA Anguo, HUANG Ping, and ZHANG Minxuan
School of Computer Science
National University of Defense Technology, ChangSha, HuNan, China (410073)
Email: lymmeng@yahoo.com.cn

KEYWORDS

Conditional stream, stream processor, stream architecture, data-dependent control, data parallelism, data-routing

ABSTRACT

Stream processor is a new architecture designed to deal with the applications which contain abundant data-parallelisms, and it can obtain high performance for regular data-parallel applications. But if there are a few data-dependent controls in the application, its performance will be reduced very much. Conditional stream can convert data-dependent control into data-routing which can be executed in stream processors. The experimental result has shown that conditional stream can improve the performance by 2X on average at very little cost.

INTRODUCTION

Stream architecture is a new SIMD data-parallel architecture which specializes in media processing. The stream programming model partitions the application into a series of kernels, computation-intensive functions that operate on streams, and a stream program that defines the high-level control-flow and data-flow between kernels(Peter Mattson , 2002),as illustrated in Figure 1.

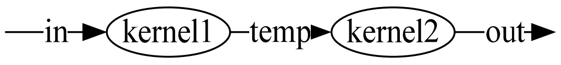


Figure 1: Stream Programming Model

The Imagine Stream Processor researched by Stanford University contains 8 clusters, which includes many ALUs and local register files (LRF). Similar to Imagine processor, Figure 2 shows the architecture of a SIMD stream processor with 4 clusters, which receive the same instructions from microcontroller and access their own LRFs with the same address. Existing study has shown that stream processor can provide several orders of magnitude higher performance efficiency than conventional programmable processors (Brucek Khailany, 2003).

Although the data-parallel architectures are excellent in the applications with regular data-parallelism, a simple data-dependent control can sharply reduce their performance. Because stream processors combine ideas from other architectures, the problem is potentially even more serious for stream processors (Ujval J. Kapasi, 2004).

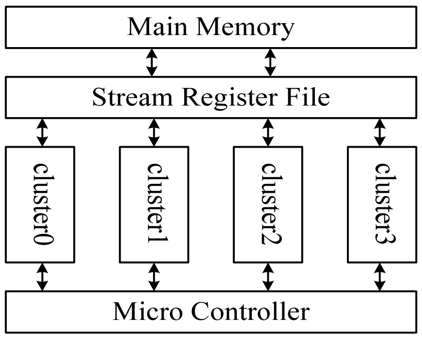


Figure 2: Stream Processor with 4 Clusters

In order to solve the problem, Ujval J. Kapasi put forward conditional stream which can convert data-dependent control into data routing, extending the application range of data-parallel architectures (Ujval J. Kapasi et al, 2000). Stream processors with conditional stream can execute the applications with data-dependent control more efficiently.

Conditional stream is a data stream that is accessed conditionally based on the conditional codes (CC) local to the cluster (Ujval J. Kapasi et al, 2000). According to the direction of stream accessed, conditional stream can be classified into two modes:

◆ Conditional output stream

In this mode, data from one stream can be dispatched into different streams, each of which will contain homogeneous data.

During conditional output stream accessing, according to its CC, each cluster decides whether to output its result into stream buffer or not, through the communicating unit and inter-switch. If half of the double buffer is full, the data will be outputted into stream register file.

◆ Conditional input stream

In this mode, data from two or more streams which contain inhomogeneous data can be combined into one stream.

Contrary to conditional output stream, each cluster decides whether to input one data or not, through the communicating unit and inter-switch, basing on the CCS. If half of the double buffer is empty, the data will be inputted into the double buffer from stream register file.

Each processing element of conventional SIMD processors has to execute the same program code. The data-dependent conditionals are commonly implemented by mask streams. For each input element, all possible outputs are calculated and associated mask streams are generated to indicate which elements of each output stream are valid. This approach leads to some deficiencies (Ujval J. Kapasi, et al, 2000).

So in order to avoid invalid results, the processor with conditional streams routes the inhomogeneous data from input stream into two or more streams, then the stream with homogeneous data can be handled apart according to corresponding conditional clause, which can avoid invalid computing and associating mask streams since there is no invalid data existing in the stream already.

PROCESS OF CONDITIONAL STREAM

Figure 3 shows the process of an output stream in a stream processor with 8 clusters. Each cluster produces the CC needed in conditional access, according to the input data or its own computing result. In the first loop, the CCs of cluster0-cluster7 are 01110110, the bit value of which determines whether this conditional transmission is valid or not. For example, the CC of cluster0 is 0, so the result which cluster0 generates is set to be invalid, and there is no valid conditional output ongoing or valid result which is accepted by communicating unit. Contrarily, the CC of cluster1 is 1, so its result is valid, value-H, and the valid transmission is ongoing.

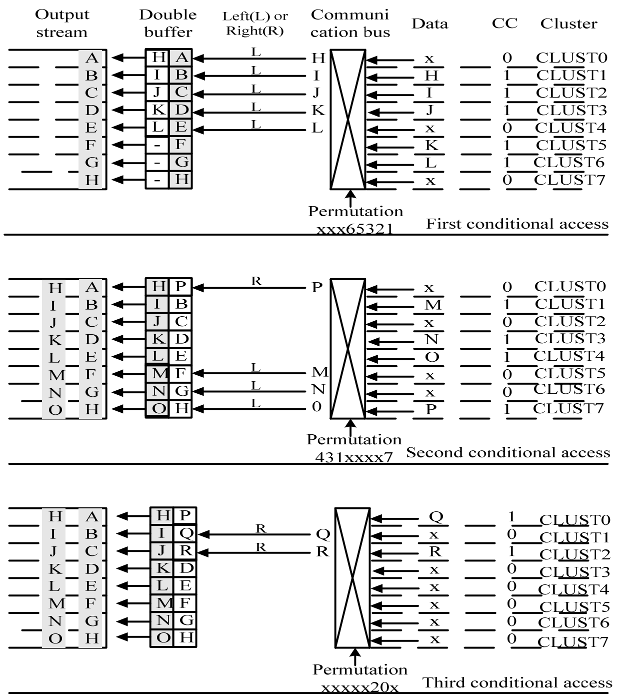


Figure 3: The Process of Conditional Stream

The number of the data outputted into stream buffer (the number of CCs) is variable. During one conditional transmission the data may not fill stream buffer to proceed stream output operation once, but the data generated during two transmissions may overflow the buffer, so a double buffer is set between the stream register file and clusters, half of which can hold all the data in a conditional transmission needed in most case, as illustrated in Figure 3.

Normal stream access is unconditional and simultaneous for all clusters, and each cluster can only access the data in the associated SB bank, for example, cluster0 can only access the first, ninth, seventeenth,…,data of stream, cluster1 can only access the second, tenth, eighteenth,…,data. But in conditional transmission, clusters must access the stream based on the CCs to expand or compress the stream, and the entry of double buffer must be accessed orderly, therefore, the valid data must be transmitted to or from right entry through a communicating unit with full crossbar. In Figure 3, communicating unit transmits the valid data to the corresponding entry in the control of permutation, which is computed by each cluster according to the ID of the start buffer entry and the CCs of all the clusters. For example, in the first loop of Figure 3, the CCs of cluster0-cluster7 are 01110110, and the ID of start buffer entry is 0, so the permutation is xxx65321, which stands for where the data written into the entry comes from. Otherwise, it has to be pointed out into which half of the double buffer the data is written. In Figure 3, all the data of first loop are written into the left half, but only part of data in second loop are written into the left half, the remnant are written to the right. When half of buffer is full, output operation proceeds to move data to stream register file. The process of conditional input stream is contrary to conditional output stream.

IMPLEMENTATION OF CONDITIONAL STREAM

From the analysis above, it is necessary that we need a double buffer between clusters and stream register file to store data, a communicating unit to exchange data between clusters and buffer. Besides those parts, a controller is also needed to generate permutation, store the state of conditional stream and so on.

The controller can be implemented outside clusters as a whole part or inside each cluster as separate parts. In the second case, intra-switch must be implemented inside each cluster, and each controller stores their own conditional state. In other words, buffers of all the clusters have to be combined to realize the function of global buffer, each of which is equal to one entry of global buffer.

Figure 4 shows the architecture of the clusters. Each cluster can be divided into two parts: ALU and the units of conditional stream, which include COM, SP, JB and VAL. COM exchanges the data between clusters so that each cluster can receive the right data from the double buffer entry. As the double buffer, SP is used to buffer data between clusters and stream register file. JB and VAL deal with the control signals sent to COM, SP and stream register file. There are two buses outside the cluster: one for the CCs and the other for the data from clusters. In order to complete the communicating operation, each cluster must send the data others need and receive the data from another cluster.

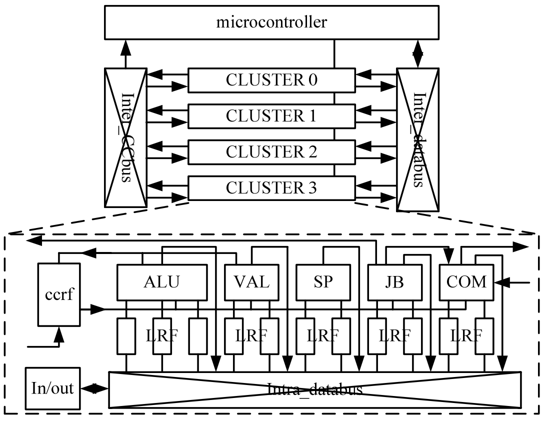


Figure 4: Architectuer of Cluster

IMPLEMENTATION OF IF-ELSE STATEMENT

There are two typical if-else statements:

- ◆ The if-else statement with two outputs

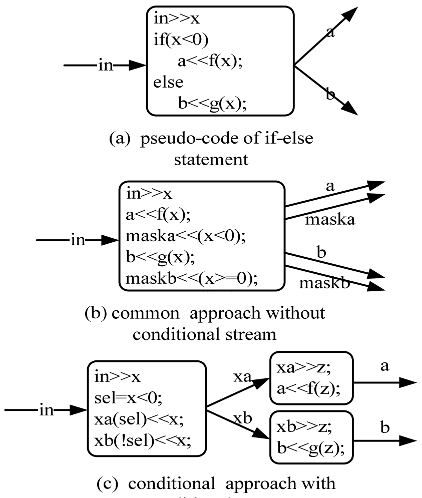
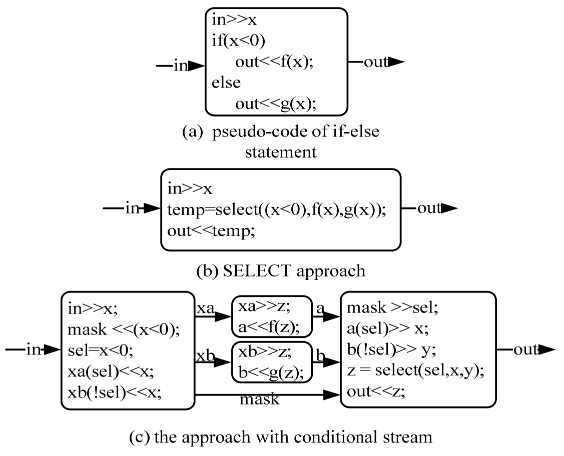


Figure 5: If-else Statement With Two Outputs

The pseudo-code of if-else statement is shown in Figure 5(a). If the data x read from input stream is minus, f(x) is computed and outputted into stream a. Otherwise, g(x) is computed and outputted into stream b.

The common approach in SIMD processor without conditional stream is shown in Figure 5(b). It partitions the if-else statement into two kernels: kernel f and kernel g. In kernel f, all the clusters compute f(x) for all x from input stream, and store the results in the stream a. Meanwhile a mask stream (maska) is generated according to the sign of the data in order to indicate which data is valid in stream a. In kernel g, all the clusters compute g(x) for all x and output the results into stream b, another mask stream (maskb) is also generated.

The approach in SIMD stream processor with conditional stream is illustrated in Figure 5(c). The whole if-else statement is partitioned into three kernels: kernel expansion, kernel f and kernel g. In kernel expansion, all the clusters read data from input stream in and store it into different streams (xa and xb) according to the sign of the data. Then in kernel f, all the clusters computes f(x) for all minus data in stream xa and the results are outputted into the stream a. Similar to kernel f, all the non-negative data are dealt in kernel g, and the results are stored in stream b. The conditional output operation in Figure 5(c) can convert the data-dependent control into data routing, so redundant computes and invalid results are avoided to reduce the extra memory and communicating operations to handle result streams.



```
in>>x
if(x<0)
    out<<f(x);
else
    out<<g(x);
```

(a) pseudo-code of if-else statement

```
in>>x
temp=select((x<0),f(x),g(x));
out<<temp;
```

(b) SELECT approach

```
in>>x;
mask <<(x<0);
sel=x<0;
xa(sel)<<x;
xb(!sel)<<x;
```

```
xa>>z;
a<<f(z);
```

```
xb>>z;
b<<g(z);
```

```
mask >>sel;
a(sel)>> x;
b(!sel)>> y;
z = select(sel,x,y);
out<<z;
```

(c) the approach with conditional stream

Figure 6: If-else Statement With Only One Outputs

◆ The if-else statement with one output

As shown in Figure 6(a), both f(x) for the minus data and g(x) for the others are computed and stored in stream out. But different with last sort, there is only one output stream.

The approach in SIMD processor without conditional stream is illustrated in Figure 6(b). The whole if-else statement is completed by one kernel. Each cluster computes f(x) and g(x) for each data from input stream in, then choose the right result according to the sign of the data to output stream out.

Figure 6(c) shows the approach in SIMD stream processor with conditional stream. There are four kernels in all: kernel expansion, kernel f, kernel g and kernel combination. The first three kernels are similar to the kernels in Figure 5(c). But the difference is that there is one more output stream (mask) in kernel expand, which is used in kernel combination. The conditional input operation is used in kernel combine. According to the value from the stream (mask), each cluster reads data from the stream xa or xb and outputs it into the stream out. It can be seen that the SELECT operation is also used here, but there are some nuances.

PERFORMANCE ANALYSIS OF CONDITIONAL STREAM

Now we take the if-else statement in Figure 5 as an example to analyze the performance of conditional stream. Suppose that the number of the minus is x, and the number of the non-negative data is y. Moreover we assume that m cycles are needed for computing f(x) once and n cycles for computing g(x) once.

So in Figure 5(b) it needs (x+y)*m cycles to finish computing f(x) for all the data in kernel f and (x+y)*n cycles to finish computing g(x) for all the data in kernel g. Then the total computing time of two kernels is $time_{no\_cond}=(x+y)*(m+n)$.

In the kernel f of Figure 5(c), computing f(x) for the minus needs x*m cycles and computing g(x) for the others needs y*n cycles. So the total computing time is $time_{cond}=x*m+y*n$.

Therefore for the if-else statement in Figure 5(a), compared with the approach without conditional stream, the speed-up of conditional stream is:

$$
\begin{aligned}
speed\_up &= time_{no\_cond}/time_{cond} \\
&=(x+y)*(m+n)/(x*m+y*n) \\
&=1+(xn+ym)/(xm+yn) \\
&=(1+ym/xn)/(m/n+y/x) \qquad (1)
\end{aligned}
$$

Assuming that the value of y/x is a, and the value of m/n is b, so in Equation (1) the speedup is:

$$speed\_up= 1+(1+ab)/(a+b). \qquad (2)$$

In fact, the data are randomly distributed, so the average value of y/x is 1, in other words, the value of a is 1. Therefore, speed_up=1+(1+ab)/(a+b) =2,we can conclude:

Conclusion 1: Compared with the approach without conditional stream, the average computing speedup of infinite distribution obtained by conditional stream is 2.

Conclusion 2: when a or b approaches 0, and b=1/a, we can get that a*b=1, a+b→∞. So (1+ab)/(a+b)→0, at this case, there is the poorest speed_up which approaches 1.

Conclusion 3: when a and b approach infinity, the speed_up =ab/(a+b) =a/2→∞, this case is the best result.

The execute time of each kernel is made up of by computing time and stall time (Sridhar Rajagopal, 2004). The first part is a large proportion. Besides, generally speaking, the computing amount in the kernel is greater, the computing proportion in the whole kernel time is larger. But above analysis only accounts for the speedup of the computing time, in fact, because there is some unavoidable overhead on memory access, the speedup of total kernel time attained by conditional stream may be less than 2.

EXPERIMENTAL RESULT

◆ Experiment 1

For the if-else statement in Figure 5, we make that $f(x)=\sin^2x+\cos^2x$, $g(x)=\sin^2x-\cos^2x$, the number of data is 1000 and the number of minus data is random. The conditional experimental result is illustrated in Figure 7(thousand cycles).

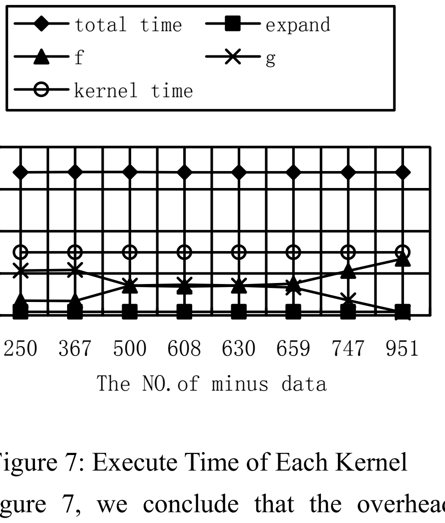


Figure 7: Execute Time of Each Kernel

From Figure 7, we conclude that the overhead of kernel expand is constant as long as the number of data is constant, and that the overhead of kernel f and kernel g change along with the number of minus data. But because the computing time of f(x) in kernel f is equal to g(x) in kernel g, $time_{cond}=(x+y)*m=1000*m$, the whole program time does not change along with the number of minus data.

The approach in Figure 5(b) must compute f(x) for all the data and g(x) for all the data. So long as the total number of data is constant, f(x) and g(x) won't change, and the overhead of the approach without conditional stream is constant no matter how many the minus data there are. The experimental result without conditional stream is shown in Table 1(cycles).

Table 1: Spending Without Conditional Stream

Total time	256634
Kernel f	71031
Kernel g	71031
Total of kernels	142062

From the above experimental result, we can find that the speedup of kernels is 1.8X. Because m is equal to n, according to the formulate 1, the theoretical value of speedup is speed_up=1+(1+ab)/(a+b) = 2, which is independent of the number of valid data. It is because of the unavoidable memory access overhead that make the real kernel speedup a little less than 2. The total speedup is 1.51X. Because conditional stream partitions the if-else statement into several kernels, processor needs more time to load the extra program code to micro controller. Besides, kernel f and kernel g need more time to access the memory which kernel expansion writes, the total speedup is less than the speedup of kernels.

◆ Experiment 2

Table 2: Spending of Different Data Size

Data size	1000		100	
Cond or not	Y	N	Y	N
Total time	68612	146936	40966	47303
Kernel time	6057	73044	627	7344
Kernel speedup	12.1		11.7	
Total speed up	2.14		1.15	

For the if-else statement in Figure 5, make that $f(x)=\sin^2x+\cos^2x$, $g(x)=x$, change data size, and make the number of minus data be zero. The experimental result of is shown in Table 2(cycles).

Again for the if-else in Figure 5, make $f(x)=\sin^2x+\cos^2x+\sin^2x-\cos^2x$, $g(x)=x$, change the data size, and make the number of minus data be zero, another experimental result is shown in Table 3(cycles).

Table 3: Spending of Another f(x) and g(x)

Data size	1000		100	
Cond or not	Y	N	Y	N
Total time	93280	234038	73882	76959
Kernel time	6058	127043	658	12743
Kernel speedup	20.9		19.4	
Total speedup	2.51		1.04	

◆ **Experiment 3**

For the if-else statement in Figure 6, make $f(x)=\sin^2x+\cos^2x$, $g(x)=\sin^2x-\cos^2x$, and data size is 1000, the number of minus data is random.

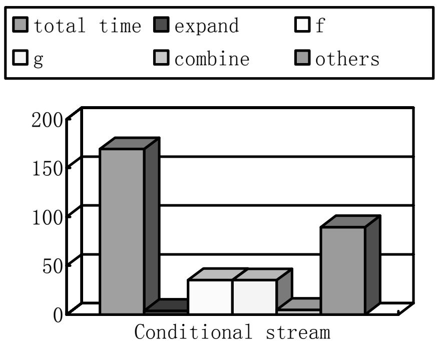


Figure 8: Experiment Result of Conditional Stream

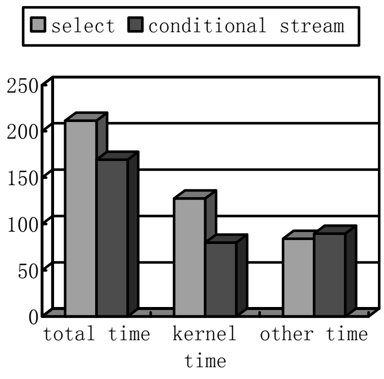


Figure 9: Comparisons of Two Approaches

Conditional stream partitions the statement into four kernels: kernel expansion, kernel f, kernel g and kernel combination. The experiment result is shown in Figure 8 and Figure 9(thousand cycles).

Of four kernels, only kernel f and kernel g execute the valid computation, and kernel expansion and kernel combination only resort the data to make computation simple and to generate right result. So the time only for computing is $timeB_{fB}+timeB_{gB}=70778$clycles, and with respect to the SELECT implement, the speedup of computing is 127531/70778=1.80; the speedup of kernel is 127531/79804=1.60; the speedup of total time is 211522/169265=1.25.

From Figure 9, we can get that the proportion of kernel time in the approach of conditional stream reduces appreciably; it is because that there are four kernels to execute and the spending on loading the program code can not be hidden.

SYNTHESIS RESULT

The X stream process with 4 clusters has been taped out at CHARTER in 130nm process. The floor plan of the processor is shown in Figure 10. The chip area is 12*12 mm^2, and it gets 34*34 mm^2 after package. The total power of the whole chip is just 8.6W.

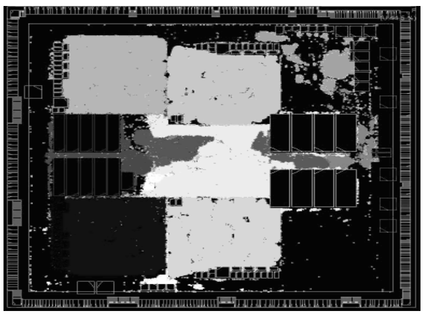


Figure 10: The Floor Plan of X Stream Processor

Table 4: Area of Conditonal Stream and Cluster

	Conditional stream	cluster
Ports	997	2128
Nets	1502	7281
Com	153292	4313977
NCom	333608	182507
Total	486895	6138413

Table 4 and Table 5 show the synthesis result of the whole cluster and conditional parts. From Table 4, we can conclude that the combination area of conditional stream units occupies 3.6% of total combination area of the whole cluster; the non-combination area occupies

18.3% of total non-combination area. The percentages of other area parameters are show in Table 4. From Table 5, we can find that the power of conditional stream only occupies a little percentage of the whole cluster, less of the whole chip.

Table 5: Power of Conditional Stream and Cluster

mw	Conditional stream	cluster
CIP	39.9687	440.5532
NSP	8.6013	19.2499
TDP	48.5700	459.8031
CLP	1.2624	20.5072

In short, conditional stream can obtain great speedup at much less cost of area, power and so on, and it is a efficient mechanism to deal with the data-dependent control.

CONCLUSIONS AND FURTHER RESEARCH

SIMD stream processor is suited to deal the applications with regular data-parallelisms, and existing studies show that it can provide more efficient performance than some conventional processor. Because stream processor combines the ideas of SIMD, VLIW, vector and so on, although stream processors can attain high performance for the data-parallel applications, the performance will slide down very much if there is only a few data-dependent controls in the applications. Conditional stream can convert data-dependent control into data routing.

Through the analysis, it shows that conditional stream can improve the performance by 2X on average in theory. Under the worst circumstances, conditional stream can still maintain the performance without any improvement, and at the best case, it can improve the performance enormously. But in fact because of memory access overhead, the average speedup may be lower than the value in theory, which can be validated through the experiments in this paper. The synthesis result shows that conditional stream only occupies a little part of the cost of the whole one cluster, and it is very efficient for stream processors.

Although conditional stream is a good mechanism, there are also some disadvantages. For example, the cluster which gets the invalid data also computes because of SIMD controlling mode. This is great waste of power, so we can stall the cluster which gets invalid data to save the power.

REFERENCES

Ujval J.Kapasi. March 2004."*Conditional Techniques for stream processing Kernels*", PhD thesis, Stanford University

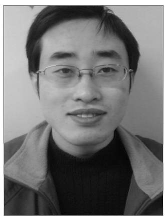
Ujval J.Kapasi, William J.Dally, Scott Rixner, Peter R. Mattson, John D. Owens, Brucek Khailany.2000. "Efficient Conditional Operations for Data-parallel Architectures", *In Proceedings of the 33rd Annual International Symposium on Micro architecture,* 159–170,Monterey,CA, ACM Press

Peter Mattson.2001."*A Programming system for the imagine media processor*", PhD thesis, Stanford University, Stanford, CA

Brucek Khailany. June 2003."*The VLSI Implementation and Evaluation of Area- and Energy-efficient Streaming Media Processors*", PhD thesis, Stanford University, Stanford, Palo Alto, CA

Sridhar Rajagopal. May, 2004."*Data-parallel Digital Signal Processors: Algorithm Mapping, Architecture Scaling and Workload Adaptation*", PhD thesis, Rice University ,Houston, TX

AUTHOR BIOGRAPHIES



SUI BING CAI was born in Shan Dong, China and went to the National University of Defense Technology, where he studied Electronic Science and Technology and obtained his bachelor degree in 2004 and master degree in 2006. Now he studies in this university for the doctor degree. His e-mail address is: `lymmeng@yahoo.com.cn`.

XING ZUO CHENG was born in An Hui, China. As a professor, he now works in the National University of Defense Technology and mainly researches the high performance and low-power microprocessors. He is now leading a large research group in the field of high performance computers. His e-mail address is: `zcxing@nudt.edu.cn`.