다중벡터감출처리(내적처리)에서 합병지연시간의 제거를 위한 설계

조 영 일† · 권 혁 률††

요 약

다중 벡터감축처리는 벡터의 내적처리([C] = [A]⊙.□[B])에서 발생하며, 두 개의 입력포트를 갖는 파이프라인유니트에서 처리된다. 각각의 스칼라 결과값은 다중 벡터감축처리(⊙)에서 요소들의 합병지연시간을 가져야 생성된다. 본 연구에서는 다중감축처리에서 요소 합병지연시간이 제거되고, 감축처리(⊙)로부터 스칼라 결과값들이 파이프라인(□) 입력시간과 거의 같게 생성될 수 있는 즉, 내적처리만을 위한 전용 체인 파이프라인 유니트 설계기법을 제안한다.

On Design for Elimination of the Merging Delay Time in the Multiple Vector Reduction (Inner Product)

Young-Il Cho† · Hyeok-Ryool Kweon††

ABSTRACT

A multiple vector reductive processing occurs during the vector inner product operation ([C] = [A] ⊙.□ [B]) and proceeds at the hardware dyadic pipeline unit. Every scalar result has to be generated with the component merging delay time in the multiple vector reduction(⊙). In this paper we propose a new design method by which the component merging time could be eliminated from the multiple reduction and the scalar results from the reduction(⊙) could be generated nearly in the almost same condensed time as the input components are feeded in the dyadic pipeline unit(□) or the output components are drained out of the dyadic pipeline unit(□), so called a dedicated chained pipeline unit for only a inner product operation.

1. Introduction

The concept of pipelining is to divide a task T into subtasks ($t_0, t_1, \cdots, t_i, \cdots, t_{n-1}$) and to assign them to a chain of processing stations and to let them execute the sub-tasks parallel and simultaneous. This principle could be commonly applied to the instruction-level,the data-level and the memory access-level [1, 2]. In the data-level pipeline the dedicated functional unit for a specified operation is divided into the sub-units and these units must be ordered according to the specified operation for the dedicated algorithm. In the past these subunits were connected in a reconfigurable way due to the necessity of each operation, but nowadays, they are as a unit dedicated ***a priori*** for the specified operations [3, 4].

The functional unit in which process the data could be divided into the segments and they have to be ordered as cascade of the order in which the algorithm of the operation processes. Assuming that the order of

※ This work was supported by the Grant for Academic Research of Hallym Uni. 1998.
† 정 회 원 : 한림대학교 교수
†† 준 회 원 : 한림대학교 대학원 컴퓨터공학과
논문접수 : 1999년 11월 12일, 심사완료 : 2000년 12월 4일

segments is the hardware of the cascade-type, each segment has a common propagation delay time and does not need to be only some functional unit, but may be a latch for a delay time unit when needed and the segments could be treated not only as a physical hardware but as a timing unit for the propagation of a subunit.

What is characteristic here is that the same type data have to be processed in the same operations in the data-level pipeline. It means that the object of the operation is a data stream and that the processing must be executed without the blank during the whole segmented operation. A dedicated pipeline is composed of the functional hardware units from the beginning to the completion of the single operation and the functional hardware units are connected ***a priori*** in a fixed order. The propagation delay time of the segments has to be identically designed and the shorter the propagation delay time, the higher the performance of the pipeline [5-8]. The pipelined functional units composed of the stages could be sorted as the followings in terms of the operating methods of the vector data.

- Monadic pipeline : to operate on one vector unit as input and to result in one vector unit.
- Dyadic pipeline : to oprerate on two vector units as input and to result in one vector unit.
- Reductive pipeline : to operate on one vector unit as input and to result in a scalar.
- Chained pipeline : to operate on multiple vector unit as input and to result in multiple scalars (vector in inner product by a dyadic and a reductive pipeline).

In the monadic pipeline the number of the output components is the same as the number of the inputs and in the dyadic pipeline the number of the output is a half of the inputs, and the output components from the reductive pipeline is not vector but a scalar. In the Cray, the reductive operation is performed in the dyadic operation pipeline through the vector register external to the pipelined functional unit [8-12]. In the monadic and the dyadic pipeline of the dedicated functional units the draining time can be the same as the feed-up time, while the draining time in the reductive and the chained pipeline should be longer than the feed-up time.

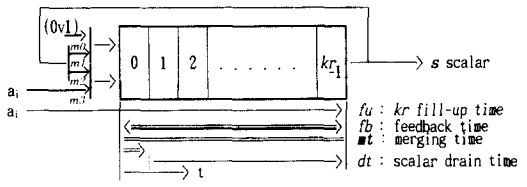
Therefore, we propose here a design principle that can be applied only for the chained pipeline unit to perform a dedicated inner product operation, in which the results can be drained as near as possible to the time of the feed-up operation, because the output draining time must be the same as the input time in a ideal data-level pipeline [13, 14].

2. Reductive pipeline operation

It is characteristic that the reductive pipeline unit has the two input ports and one of them is for external input and the other for the feedback of interim result and that the operation generates a scalar result finally. In other words, it means that the operation executes dyadic on the stream input of one vector unit data and results in a scalar output. This operation could be represented as $s \leftarrow f/V$; here f means a dyadic function as addition, subtraction, multiplication, division that could operate on two input operands and V is a vector that is composed with more than one components. Besides those operations there can be some operations like ***Search Maxima*** and ***Search Minima*** and so on ··· [8]. To generate a scalar result from the vector input, it is inevitable to use vector merging operations. In the past such a vector merging operation was achieved through the high utilization of the PE's [11]. Here we have examined the possibility that such an operation could be performed in a dedicated pipeline specified only for the reductive operation. The reductive operation can be represented as $s = a_0 \odot a_1 \odot a_2 \odot \cdots \odot a_i \odot \cdots \odot a_{n-1}$. The s is the scalar result of the operation($\odot$) and the a_i the components of the vector. The structure of the processing cycle time of the reductive operation($\odot$) can be analyzed as (Figure 1) and the followings.

1) Fill-up time(fu) : $fu = kr$, $n > kr$.

The length of time until all of the segment(kr) will be filled up from the beginning. The pipeline has to



(Figure 1) Timing structure of the reductive processing

operate dyadically on the components of input vector unit and the neutral operand (For example, 0 in a = a + 0, 1 in a = a·1).

2) Feed-back time(*fb*) :

① *fb1* = $n - 2 \cdot kr$, $n > 2 \cdot kr$ or fb1 = $n - 1 \cdot kr$, $n > 1 \cdot kr$.

The length of time until the last component(n-th) enters into the first segment(0) of the pipeline after the time of the fill-up operation.

② *fb2* = $n - kr$.

The length of time until the a_i data of ($n - kr$) processes with the result data b_j from the kr segments and at last each segment (kr) is filled with operands (kr) in the reductive pipeline operation.

3) Merging time (***mt***) :

The length of time until the operand that stays in each segment merges to a pair in the first segment and the last merging is completed in the first segment or the scalar result. It could be obtained as a constant determined by the number(kr-1) of the segments.

4) Draining time(*dt*) :

The length of time until the last pair of the operands for the last operation processes from the first segment (0) to the last kr-th segment and results in a scalar.

Besides the above described timing structure transpiring within the pipeline operation, we need the following cycle time units.

- B_{ct} : Bus cycle time at which one data occupies on the parallel bus.
- P_{ct} : Average propagation delay time at which the data are processed in every segments.
- Tp_ : Pipeline processing time from the first input till the first output, Set-up time.

Although the cycle time Tp_ is the length at which the n components are processed in the monadic - or the dyadic pipeline by the case $n >> kr$, it has to mean the time that the last two operands should merge in the first segment in the reductive pipeline. It is characteristic that a reductive pipeline operates ***dyadically*** on one input vector unit and results in a scalar value. Therefore, one input of the two has to accept the external input and the other from the feedback of the interim output results. It is called as internal input operands. The reductive operation can be represented as $s = a_0 \odot a_1 \odot a_2 \odot \cdots \odot a_i \odot \cdots \odot a_{n-1}$.

If n > kr, the total processing cycle time (Tpr) should be composed of fu + fb2 + ***mt*** + kr ; if n ≤ kr, fu + ***mt*** + kr without the feedback (fb2). After the feedback operation (fb2) the operands that stay in each segments have to become a pair of operands in the first segment. The time, in which the last pair is set up in the first segment, is called the merging time (***mt***). It is a kind of ***a priori*** delay time in the reductive pipeline operation and is constant numbers caused by the number (kr) of the segments as the followings.

```
kr : 2 3 4 5 6 7 8 9 10 . . . . 16 17 18 19 . . . 32 33 34 35
     ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ . . . . ↓ ↓ ↓ ↓ ↓ . . . ↓ ↓ ↓ ↓
mt : 2 5 8 12 16 20 24 29 . . . . 59 64 70 76 82 . . . 160 167 174 181
     \/ \/ \/ \/ \/ \/ \/ \/ . . . . \/ \/ \/ \/ \ . . \/ \/ \/ \/
dd : 3 3 4 4 4 4 5 5 . . . . 5 6 6 6 . . . 6 7 7 7 . .
     \2¹/ \  2² /  \  2³   /  \   2⁴  /  \ 2⁵
```

The merging time (***mt***) can be derived as followings, related with the number of the segment (kr). The value of the dd is the difference between the merging time (***mt***) of each reductive pipeline (kr). The number (kr) of pipeline segments can be represented as $2^x + i, 0 \le i < 2^x$. If i = 0, the merging time (***mt'*** : underlined ***mt***) could be calculated as the following (1). If i > 0, the distance of the merging time($dd = (x + 2) \cdot i$) has to be added to (1). Therefore, in case of $\boldsymbol{n} \ge kr$, the merging delay time is as the following (2).

$$\boldsymbol{mt'} = x \cdot 2^x \quad (1)$$

$$\boldsymbol{mt} = x \cdot (2^x + i) + 2 \cdot i = x \cdot kr + 2 \cdot i \quad (2)$$

Here the difference between the conditions ($n > kr$) and ($n \le kr$) that results from the dyadic operations is whether the feedback time (*fb2*) is in the reductive operation as the followings (3), (4) or not.

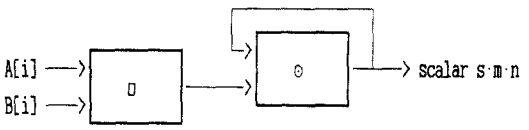
$$n > kr,\quad \mathrm{Tpr} = fu + fb2 + \boldsymbol{mt} = (kr + n - kr) + x \cdot kr + 2 \cdot i = n + \boldsymbol{mt} \qquad (3)$$

$$n \le kr,\quad \mathrm{Tpr} = fu + \boldsymbol{mt} = kr + \boldsymbol{mt} \qquad (4)$$

We consider here the condition of ($n > kr$) rather than ($n \le kr$). The reductive pipeline operates on two operands; In other words, dyadic operation. As described above in (Figure 1), it needs the external operands of the number of *kr* and the neutral operand (0 or 1) by the control signal *m0* in the fill-up time (*fu*) for the operands (*kr*). It could be called a neutral operation performed during the fill-up time (*fu*). The rest operands (n-*kr*) in the vector components have to be processed dyadically with the feedback of the interim result operands from the neutral operations until the last operand (n-th) is feeded in the first segment (0) of the pipeline. Here it needs the control signal *m1* and *m3* in the feedback time (*fb2*) for the processing($n - k \cdot kr$, $k = \{i \mid i = \text{integer}\}$). It could be called the feedback operation. Next, it needs the time to merge the operands stay- ing in each segment (*kr*). It could be operated on the operands that are distributed for the dyadic operation by the control signal *m1*,*m2*. This time is called the merging time (***mt***) and defined as a constant determined ***a priori*** from the number of pipeline segments (*kr*). The operation repeats until the last merging proceeds in the first segment. Therefore, it needs the processing cycle time ($n > kr$, Tpr) to operate on the components (n) of a vector unit reductive.

3. Chained operation for vector processing

The chained operation for vector processing is composed of both the dyadic vector operation and the above described reductive operation as (Figure 2). It is considered that there are 4 kinds of the dedicated functional unit for the operation of the inner product [8, 9].



(Figure 2) Chained operation structure

It needs at first two vector units(respectively 2-dimensions) as source operands for the inner product ($C[m, n] = A[m, \boldsymbol{\iota}] \odot. \square B[\boldsymbol{\iota}, n]$) and the length $\boldsymbol{\iota}$ of the row components of vector [A] and that of the column components of vector [B] are processed in the dyadic pipeline(□). Here the $\boldsymbol{\iota}$ is the vector length equivalent to n in the above (3),(4).

The result components of the dyadic operation will have the length $\boldsymbol{\iota}$ and conquently are resulted in a scalar from the reductive pipeline(⊙). In the inner product the results ($s_0, s_1, \cdots, s_i, \cdots s_{m \cdot n-1}, s_i = c_0 \odot c_1 \odot, \cdots, \odot c_i \odot, \cdots, \odot c_{l-1}$) of the dyadic operation (□) have to be grouped with each separate other $\boldsymbol{\iota}$ components and feeded in the reductive pipeline (⊙) and resulted in the number of m·n scalars. For this reason the result of an inner product is C[m, n].

The dyadic operation(□) for the ($c_{ik} = (a_i \square b_k)_l$, $i = 0, 1, 2, \cdots, m-1$. $j = 0, 1, 2, \cdots, \boldsymbol{\iota}-1$. $k = 0, 1, 2, \cdots, n-1$) have to perform the operation(□) $\boldsymbol{\iota} \cdot m \cdot n$ times, and the reductive operation(⊙) for the ($s_i = (c_{ik})_0 \odot (c_{ik})_1 \odot (c_{ik})_2 \odot \cdots \odot (c_{ik})_l \odot \cdots \odot (c_{ik})_{l-1}$, $i = 0, 1, 2, \cdots, i, \cdots, \boldsymbol{\iota}-1$. $\boldsymbol{\iota} > k$) of the s_i has to be executed with the input($\boldsymbol{\iota} \cdot m \cdot n$) at m·m times and results in the scalar of the m·n. The processing cycle time from the vector ($(c_{ik})_0 \odot (c_{ik})_1 \odot \cdots \odot (c_{ik})_l \odot \cdots \odot (c_{ik})_{l-1}$) to a scalar component($s_i$) of the result matrice C[m, n] is the $\boldsymbol{\iota} \cdot m \cdot n$. In case of $\boldsymbol{\iota} > kr$, the reductive pipeline cycle time (Tpr) is $fu + fb2 + \boldsymbol{mt}$. Therefore, the total processing cycle time for the matrice C[m, n] will be $(\mathrm{Tpr} \cdot m \cdot n + kr) \cdot P_{ct}$. The average propagation delay time P_{ct} of the segments should be as double as the bus cycle time B_{ct}, because the operands have to be distributed through the demultiplexing of the dyadic pipeline unit ($P_{ct} = 2 \cdot B_{ct}$).

3.1 Chained operation with sequential reductive processing

To chain the two operations, the dyadic pipeline (□) has to be connected with the reductive pipeline (⊙) and the results of the dyadic operation (□) have to become the input for the reductive operation (⊙). Until now such reductive operation that results in a scalar value from the vector has to be performed in the normal dyadic pipeline unit [11].

As mentioned above,the total processing cycle time of the chained operation could be Tpc =Tpd+Tpr. But it is only the case when the results of the dyadic pipeline can feed directly the reductive pipeline as input. Because of the merging delay in the reductive pipeline the result vector unit (i + 1) could be mixed with the components of the vector unit (i) in the reductive operations. Such a phase is called an ***operand conflict***. To avoid such an operand conflict a dummy segment buffer in which the components of the vector unit (i + 1) can stay for waiting during the time (*kr*) until the vector unit (i) merges completely, is designed between the dyadic (□) and the reductive pipeline (⊙) as the following (Figure 3) [8, 9] or during the reductive operation (⊙) of the vector unit (i) the intermediate results of the next vector unit (i+1) must be stored in the vector registers or in the memory (be a local memory). Such a method could bring about a ***bottleneck***.

The merging time of the reductive pipeline (⊙) is a constant value derived from the number (*kr*) of the reductive pipeline segments. In the consequence of that, each scalar result components of the C [m, n] have to be generated with the time distance equivalent to the merging delay time (***mt***). Therefore, the reductive operation (Tpr) for the inner product (C[m, n] = A[m, ***l***] ⊙□ B[***l***, n]) should proceed m·n times Tpr for the scalar component of the vector C[m, n] as in the following (5) totally.

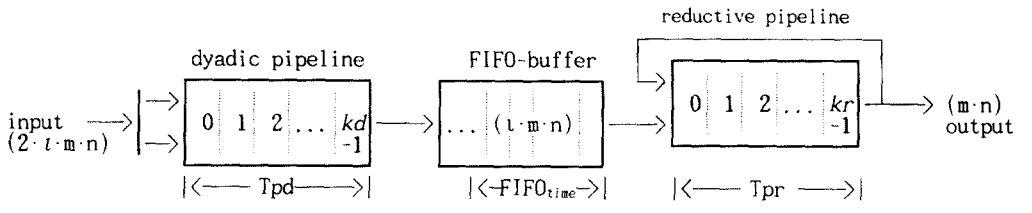
$$St[m,n] = Tpd + FIFO_{time} + Tpr = kd + FIFO_{time} + (l + mt)\cdot m\cdot n + kr \quad (5)$$

In the reductive processing time(Tpr = (***l*** + ***mt***)·m·n + *kr*) (5) the term (***l*** + ***mt***)·m·n could be analyzed as the term that is the results (***l***·m·n) from the dyadic pipeline (□) and also, the input operands for the reductive pipeline (⊙), and as the term that is the total merging delay time (***mt***·m·n) for the final results (m·n) which elapses m·n times sequentially, as if done in the non-pipelined functional units.

3.2 Chained operation with parallel reductive processing

As described above, to escape the operand conflict or the bottleneck, the scalar results could be generated with the distance of the merging time (***mt***) in the sequential processing pipeline. Such an useless merging time (***mt***) is proportional to the number (m·n) of the component of the vector C[m, n]. The reductive operation could eliminate the useless merging time (***mt***) by the parallel design of the reductive pipeline units (⊙) in an interleaved fashion without the buffer memory or the waiting state in memory between the dyadic pipeline and the reductive pipeline as in the following (Figure 4.)

We propose here a design method that can eliminate the merging time (***mt***). It is the method that the reductive pipeline units (⊙) have to be parallel designed for the elimination of the merging time (***mt***). Here the number (rp · n) of the reductive pipeline units (⊙) should be ***a priori*** the same number as the merging time (rp·n



(Figure 3) Sequential chained reductive pipeline with memory operation

= ***mt***), because the length (ι) m·n times should be distributed to the reductive pipelines during the merging delay time (***mt***) by the modulation.

The term ***mt***·m·n in (5) that means the sequential processing cycle time has to be expressed as the parallel processing of the reductive operation as in the (Figure 4). In other words, the term ***mt*** is the same as the pipeline segments and the m·n as the input vector, as the (k + ***n***) clock cycle time should be needed to complete n tasks using a ***k***- segments pipeline [14]. Therefore, the parallel reductive processing cycle time (Pt[m, n]) for the m·n scalar components of the vector C[m,n] is the sum of ***mt*** and m·n and the final draining time (dt = kr) as the following (6).

$$\begin{aligned} Pt[m, n] &= Tpd + Ppr \\ &= kd + \iota \cdot m \cdot n + (mt + m \cdot n) + kr \end{aligned} \qquad (6)$$

In the parallel chained pipeline the number of the reductive pipeline units is equal to the merging delay time of one reductive pipeline and thus, the length (ι) of one vector unit (i) could be distributed to each parallel reductive pipeline units. It is not necessary to wait for the merging delay time only in one reductive pipeline. In the (Figure 4) the module distributer should control the pipeline selecting demultiplexer that takes the result vector (ι·m·n) of the dyadic pipeline (□) as inputs to distribute the length unit (ι) in turn to each reductive pipeline unit.

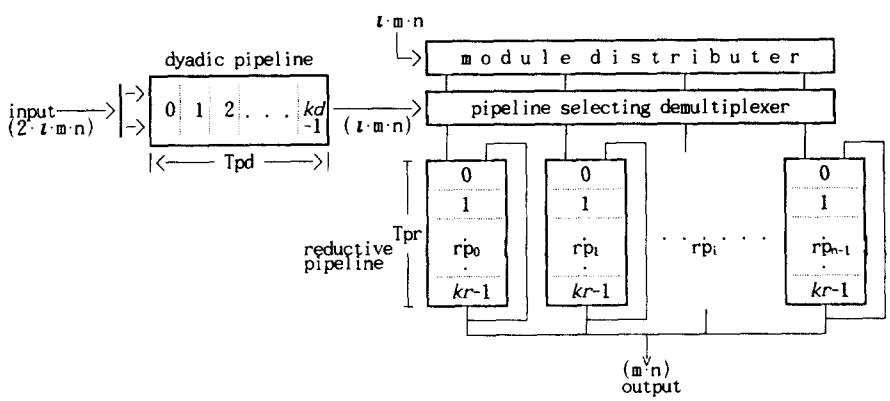
Control mechanic for distribution

The pipeline selecting demultiplexer that supplies the parallel reductive pipeline units with the vector of the ι units has to be controlled by the module distributer as in the (Figure 4). The module distributer as a control unit for the pipeline selecting demultiplexer is composed of a decoder and a decoding address counter (**D/A**-ctr) and a modul counter with parallel input ι. The **D/A**-ctr takes the pulse 1 for the up-counting modulo ($\lceil log_2 rp \cdot n \rceil$) whenever the modulo counter becomes (ctr| ι) ≡ ι and it should count up m·n times ι. Then the vector components (ι·m·n) from the dyadic operation could be supplied to each reductive pipeline in turn with length of ι at m·n times.

4. Performance evaluation

In the chained operation for an inner product the number of the input components for the dyadic operation (□) is 2·ι·m·n with the processing cycle time 2·ι·m·n·B_{ct} to- tally and the number of components ι·m·n results from that operation (□) with the speed P_{ct}, and then the components m·n as the output of the reductive processing (○) with the speed $\iota \cdot P_{ct}$.

As mentioned in the [9], the hardware for the inner product is implemented with the pipelined network. But here the implementing method is that the dyadic unit (□) and the reductive unit (○) are composed of the pipeline segments. If the reductive pipeline operation



(Figure 4) Inner product pipeline with parallel chained reductive operation

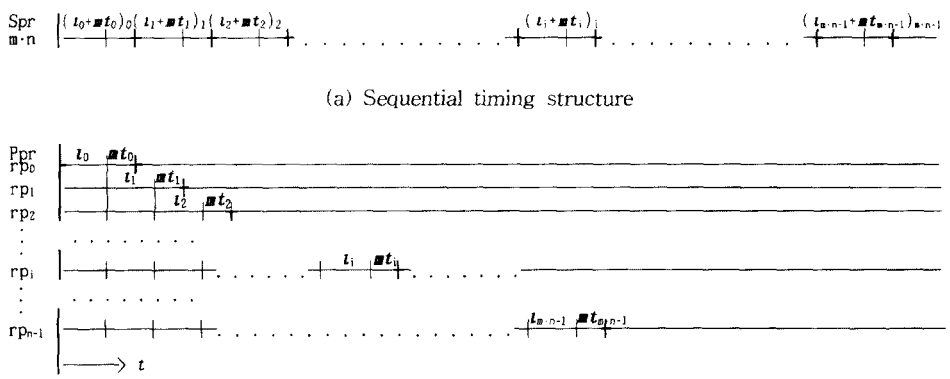
follows the dyadic operation directly, it is problematic when not one vector unit but the multiple vector units are processed with ι-length unit in the reductive unit (⊙). In other words, an ***operand conflicts*** could occur between the reductive operations of the multiple vector units. Therefore, it is necessary for the memory operation (or $FIFO_{time}$) to be inserted between the dyadic - and reductive operations. The sequential inner product operation has to be executed by dyadic functional unit (□), memory operation and reductive functional unit(⊙) in turn [8].

In the sequential reductive pipeline for the inner product by [8] the output components ($\iota \cdot m \cdot n$) of the dyadic pipeline (□) have to be feeded into the reductive pipeline (⊙) through the queue buffer operation by every reductive processing cycle time ($\iota + mt$) in the (5). As a consequence, it means that the every output of m·n from the reductive operation (⊙) has to be generated with the time distance ($\iota + mt$).

In the parallel reductive operation (6) for the multiple vector the components of $\iota \cdot m \cdot n$ from the dyadic pipeline (□) can be directly feeded into each different reductive pipeline units(⊙) during merging of the former vector without any delay time (mt) and without any queue buffer operation time except an initial merging time (mt), because the input components $\iota \cdot m \cdot n$ should be feeded with careful control into the different reductive pipeline units(⊙) with the number of every ι components by the control of the module distributer as in the (Figure 4).

Therefore, the scalar results generated from the each reductive pipelines could become a stream with the components m·n from the parallel reductive(⊙) and then the time distance between the components is $\iota \cdot P_{ct}$ in (6). If we compare the result output time from the sequential with that from parallel reductive operations, we could take the time distance between each output components as the following (Figure 5). As we can see in the (Figure 5-a), the processing of the vector length ι has to be executed m·n times by one reductive pipeline unit sequentially. But in the (Figure 5-b). because of the parallel reductive pipelines a merging delay operation (i) could be executed during the next feeding time (i + 1). Therefore, only the last merging time ($mt_{m \cdot n\ 1}$) has to be added physically to the total feeding time ($\iota_0 + \iota_1 + \iota_2 + \cdots + \iota_i + \cdots + \iota_{m \cdot n\ 1}$).

As a result, the ι of the $\iota \cdot m \cdot n$ in (5),(6) merely means the length unit that should be feeded into the other reductive pipelines, and the ($\iota + mt$) in (5) means the processing cycle time at which it could take the length (ι) of vector to be executed. As mentioned above, the comparison of $mt \cdot m \cdot n$ in (5) with $mt + m \cdot n$ in (6) is like that of the non-pipelined processor to the pipelined processor [12]. Therefore, we can see that the total time of the (5) is much longer than that of the (6) as in the following dif- ference(7).



(a) Sequential timing structure

(b) Parallel timing structure

(Figure 5) output time of the sequential and the parallel reduction

$$Diff.(St[m, n] - Pt[m, n]) = FIFO_{time} + mt \cdot (m \cdot n - 1) - m \cdot n \quad (7)$$

Hence, the $FIFO_{time}$ is simultaneous time in the total processing, so we can derive a pipeline unit time per output($m \cdot n$) from the sequential reductive and parallel operation time except the hardware unit times kd, $FIFO_{time}$, kr because of the intrinsic constant from the (5), as the followings (8), (9).

$$T_{sp} = (\iota \cdot m \cdot n + mt \cdot m \cdot n) / m \cdot n = \iota + mt \quad (8)$$

$$T_{pp} = (\iota \cdot m \cdot n + mt + m \cdot n) / m \cdot n = \iota + (mt / m \cdot n) + 1 \quad (9)$$

We can see from the two equations (8), (9) that each of output components (m, n) have the time ($\iota + mt$) and $\iota + (mt / m \cdot n)$, and from the equation (9) that the parallel processing time approaches to ι, if $m >> 0$, $n >> 0$. Therefore, we can define the ratio T_{sp} / T_{pp}, as a speedup (Sp) factor of a parallel reductive processing using $rp \cdot n$ processing hardware [13]. This is the ratio of the sequential processing time to the parallel processing time per vector unit ι.

$$Sp = \frac{T_{sp}}{T_{pp}} = \frac{\iota + mt}{\iota + (mt / m \cdot n) + 1} = \frac{\iota + mt}{\iota} \quad (10)$$

From the equation (10), we consider the case of $\iota > mt$, $\iota \leq mt$. If $\iota > mt$, $1 \leq Sp < 2$, and If $\iota \leq mt$, in other words, if $\iota << mt$, $2 \leq Sp < mt$. Therefore, we can see that the longer the length (ι) relative to the merging time (mt), the less the speedup (Sp), and the shorter, the better.

Moreover, we must consider that the memory operation time should be added to the (8). As regards the total components, the sequential processing has the time mt ($m \cdot n - 1$) longer than the parallel processing as we can see at the Figure 5. In the above (10), if $m >> 0$, $n >> 0$, the term $mt / m \cdot n \cong 0$. Therefore, we could discard the term ($mt / m \cdot n + 1$). The efficiency (Ep) of the parallel reductive operation could be defined as the ratio ($Ep = Sp / rp \cdot n \leq 1$) between the speedup factor (Sp) and the number of units ($mt = rp \cdot n$) as the following (11) and is a measure of the cost effectiveness of computations.

$$Ep = \frac{\iota + mt}{\iota \cdot mt} \quad (11)$$

As we can see in the (Figure 5), it needs not to be considered as the operation time, because the merging delay times ($mt_{m \cdot n - 1}$) of each reductive pipeline can be brought about simultaneously during the reductive operations except for the last ($\iota_{m \cdot n - 1}$).

We can derive a ratio from the number of the results ($m \cdot n$) of the parallel reductive pipeline to their processing cycle time ($\iota \cdot m \cdot n + mt$) and how many times the results (Rpp) can be processed in the parallel reductive pipeline during the time at which the results ($m \cdot n$) can be processed in the time ($\iota \cdot m \cdot n + mt \cdot m \cdot n$) in the sequential reductive pipeline.

$$Rpp = \frac{m \cdot n}{\iota \cdot m \cdot n + mt} \cdot (\iota \cdot m \cdot n + mt \cdot m \cdot n) = \frac{m^2 \cdot n^2 \cdot (\iota + mt)}{\iota \cdot m \cdot n + mt} \quad (12)$$

As in the (12), the parallel re··ductive pipeline could do the results. Here we can define as output the density of the parallel reductive pipeline operation,the ratio of the parallel reductive operation to the sequential reductive as the ($P_{dens.} = Rpp / m \cdot n$) in the same time as the sequential reductive operation as the (13).

$$P_{dens.} = \frac{m \cdot n \cdot (\iota + mt)}{\iota \cdot m \cdot n + mt} \quad (13)$$

5. Conclusion

It is characteristic that the speed of output draining after a certain propagation time should be the same as that of input feeding in a data-level pipeline for a vector processing. But in the chained operation, for example, inner product, it could not be so in the sequential reductive pipeline as mentioned above. Therefore, we propose here the parallel reductive pipeline should be designed.

In the consequence of the (13), if the results ($m \cdot n$) would be used as the elements of an image processing

we could expect that the parallel reductive pipeline could generate the better density or resolution than the sequential reductive pipeline in the same time.

References

[1] D. C. McCrackin, "Eliminating Interlocks in Deeply Pipelined Processors by Delay Enforced Multistreaming," IEEE Trans.on Comput., Vol.40. No,10. Oct. 1991.

[2] H. S. Stone, *High-Performance Computer Architecture*, 2nd ed., Addison-Wesley Publishing Company, 1990, pp.122-137.

[3] R. Gupta, A. Zorat and I. V. Ramakrishnan, "Reconfigurable Multipipelines for vector Super computers," IEEE Trans.on Comput., Vol.38, No.9. Sept. 1989.

[4] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill. 1984. pp. 151-154.

[5] J. R. Jump and S. R. Ahuja, "Effective Pipelining of Digital Systems," IEEE Trans. on Comput., Vol. c-27, No.9, Sept. 1978.

[6] S. R. Kunkel and J. Smith, "Optimal Pipelining in Supercomputers," The 13th Annual Int.Symp. on Comp. Arch. Conf. Proc. ACM, 1986.

[7] P. M. Kogge, *The architecture of pipeline computer*, McGraw-Hill, 1981, pp.134-173.

[8] L. M. Ni and K. Hwang, "Vector-Reduction Techniques for Arithmetic-pipelines, "IEEE Trans. on Comput., Vol.c-34, No.5, May, 1985.

[9] D. J. Kuck, The Structure of Computers and Computations, Vol.1, New York : Wiley, 1978. pp.257-258.

[10] E. E. Swartzlander, B. K. Gilbert, I. S. Reed, "Inner Product Computers," IEEE Trans. on Comput., Vol. c-27, No.1, Jan. 1978.

[11] J. R. Vanaken and G. I. Zick, "The Expression Processor : A Pipelined, Multiple-Processor Architecture," IEEE Trans. on Comput., Vol.c-30, No.8, Aug. 1981.

[12] R. M. Russel, "The Cray-I Computer System," Cray Research, Inc., Tutorial Advanced Computer Architecture, IEEE Computer Society, Order No.667, 1986, pp.15-24.

[13] C. V. Ramamoorthy & H. F. Li, "Pipeline Architecture, "Tutorial Computer Architecture, Comp. Soc. Or. Nr. 704, THE COMPUTER SOCIETY OF THE IEEE, 1987, pp.38-79.

[14] D. I. Moldovan, *Modern Parallel Processing*, University of Southern California, 1986.

[15] M. M. Mano, *Computer System Architecture*, 3rd. Prentice International Ed. 1993, pp.305.

조 영 일

e-mail : yicho@sun.hallym.ac.kr

1971년 고려대학교 졸업(학사)

1980년 (서독) Berlin 공대 컴퓨터 공학부 졸업(석사)

1982년 (주)금성사 중앙연구소

1985년~현재 한림대학교 교수

1996년~건국대학교 대학원(박사)

관심분야 : 컴퓨터 구조설계, 병렬처리 설계

권 혁 률

e-mail : kweonhr@korea.com

1998년 강원대학교 졸업(학사)

1999년~현재 한림대학교 컴퓨터 공학과 대학원

관심분야 : 컴퓨터 구조설계, 병렬처리 설계