SISD 머신에 부착 가능한 SIMD 벡터 머신의 개념적 설계

조 영 일[†] · 고 영 웅[††]

요 약

데이터 주소의 계수를 위한 하드웨어 설계가 없는 본 노이만(von Neumann) 개념(SISD)의 컴퓨터에서 데이터의 주소지정은 소프트웨어적으로 수행된다. 그러므로 벡터 데이터 요소들의 주소지정은 인덱싱 기법에 의해 그 요소 수만큼 해당 변수들을 만들어서 사용해야 한다. 이것은 데이터 계수기 없이 명령어 계수기, 즉 PC(program counter)만 하드웨어로 설계되기 때문이다. 본 연구에서는 중앙처리장치 외부에 외형적 구조와 크기를 갖는 단위 벡터의 요소를 액세스하는 하드웨어 유닛의 설계를 제안한다. 벡터 처리는 고속처리가 전제되기 때문에 파이프라인 처리기법(SIMD)으로 설계되어야 한다. 제안한 방법은 시뮬레이션을 통하여 성능 검증을 하였으며, 실험 결과 동일한 프로세싱 유닛을 가지는 벡터 머신 아키텍쳐보다 12 - 30 % 정도 우수한 성능을 내는 것을 확인하였다.

키워드 : SISD, SIMD, 벡터 컴퓨터, 스칼라 컴퓨터, 2차원 계수기

On the Conceptual Design of the SIMD Vector Machine Attachable to SISD Machine

Young-Il Cho[†] · Young-Woong Ko[††]

ABSTRACT

The addressing mode for data is performed by the software in von Neumann- concept(SISD) computer *a priori* without hardware design of an address counter for operands. Therefore, in the addressing mode for the vector the corresponding variables as much as the number of the elements should be specified and used also in the software method. This is because not for operand but only for an instructions, *quasi* PC(program counter) is designed in hardware physically. A vector has a characteristic of a structural dimension. In this paper we propose to design a hardware unit physically external to the CPU for addressing only the elements of a vector unit with the structure and dimension. Because of the high speed performance for a vector processing it should be designed in the SIMD pipeline mechanics. The proposed mechanics is evaluated through a simulation. Our result shows 12% to 30% performance enhancement over CRAY architecture under the same hardware consideration(processing unit).

Key Words : SISD, SIMD, Vector Computer, Scalar Computer, 2-dimensional Counter

1. 서 론

오늘날 사용되고 있는 컴퓨터는 범용 컴퓨터와 특수 분야에서 사용되고 있는 초고속 벡터처리 슈퍼컴퓨터로 대별할 수 있다. 범용 컴퓨터는 반도체 공정과 설계의 경이적인 발전으로 인하여 수 기가 헤르츠(GHz) 이상의 속도로 발전하였으며, 벡터처리 컴퓨터는 CDC STAR 100A, TI-ASC 등의 1세대를 지나 오늘날의 초고속 벡터처리 슈퍼컴퓨터에 이르렀다[1]. 그 두 방향의 컴퓨터를 처리원리에 따라서 분류해 보면 범용 컴퓨터는 SISD(Single Instruction Single Data) 처리 원리 설계이고, 슈퍼컴퓨터는 SIMD(Single Instruction Multiple Data) 처리 원리라고 할 수 있다. 다시 말해, SISD 처리 원리는 스칼라 처리지향(스칼라 컴퓨터)설계에, SIMD 처리원리는 벡터 처리지향(벡터 컴퓨터)설계에 적합한 것이다[2].

스칼라 컴퓨터에서는 하나의 명령어로 하나의 소스 오퍼랜드를 처리(SISD)하여 단 하나의 결과 값, 즉 하나의 스칼라 결과를 내는 특징을 갖는다. 그러나 벡터 컴퓨터는 하나의 명령어로 여러 개의 단위벡터를 처리(SIMD)해야 한다. 여기서 벡터는 n(n>2)개의 요소를 가지며, 하나의 구조 내지는 크기를 갖는 구조성 단위이다. 벡터가 처리되기 위해서 그 요소들은 메모리로부터 스트림(stream)으로 이동처리되어야 하며, 벡터 요소 수만큼의 변수가 필요하고, 그 숫자만큼의 ALU 기능에 의한 주소계산과 반복적 명령어 수행을 위한 시간을 필요로 하게 된다. 스칼라 컴퓨터에서 벡터를 처리하려면 필수 불가결로 발생하는 요소 수만큼의 변수,

† 정 회 원 : 한림대학교 정보통신공학부 교수
†† 정 회 원 : 한림대학교 정보통신공학부 조교수
논문접수 : 2004년 12월 16일, 심사완료 : 2005년 4월 8일

주소 계산과 명령어의 반복적 수행은 스칼라 처리에서 장시간을 요구하게 되며, 이런 점이 벡터처리를 위해 구조적 기능 변화를 필요로 하는 이유이다.

벡터처리 전용 컴퓨터인 CRAY에서 이런 문제점을 해결하기 위해 메모리와 데이터 처리부 사이에 벡터 레지스터(8x64 words)를 설계하고 있다[1]. CYBER 205, ETA 10을 제외한 모든 CRAY, Fujitsu VP, Hitachi, NEC SX 계열 등이 모두 레지스터간(register to register) 처리 기법 설계이다[3]. 하지만, 이 방법은 벡터를 레지스터로 옮기는 주소계산시간과 벡터의 처리시간이 직렬로 소요된다는 것에 문제가 있다. 본 논문에서는 보다 더 빠른 벡터처리를 위해 메모리간(memory to memory) 처리 기법 설계를 제안하고 있다.

본 논문의 순서는 다음과 같다. 2장에서는 스칼라 방식에 벡터 처리 방식을 부착하여 사용하는 연구에 대해서 기술하며, 3장에서는 메모리간 처리 기법을 사용하여 VPU를 설계하는 방법에 대해서 구체적으로 기술한다. 4장에서는 시뮬레이션을 통한 성능 분석을 수행하며, 5장에서 결론을 맺는다.

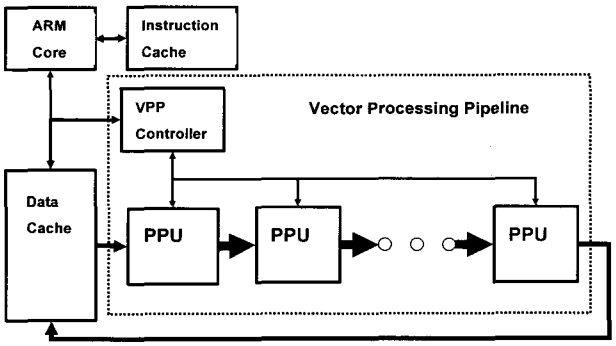
2. 관련 연구

본 논문에서 제안하는 본 노이만[4] 구조의 스칼라 프로세서에 벡터 프로세서를 부착하는 방법과 유사한 연구로는 VPP(Vector Processing Pipeline) 프로그래밍 모델[5]과 벡터 방식의 지능형 램(VIRAM: Vector Intelligent RAM)[6, 7, 8, 9]이 대표적이다. VPP는 계산 집약적인 무선 프로토콜을 처리하기 위한 유연한 구조로 제안되었으며, 프로그래밍 가능한 특성을 가지는 벡터 코프로세서(vector coprocessor)이다. 설계의 핵심은 데이터 레벨의 고수준의 병렬성을 효율적으로 이용하는 메모리 스트리밍 방식의 매크로 파이프라인(macro-pipelined) 벡터 구조를 이용하고 있는 것이다. VPP가 처리하는 무선 프로토콜의 경우 생산자 소비자 모델에 기반하고 있으며, 데이터가 병렬적으로 처리될 수 있는 구조를 취하고 있다. 즉 데이터가 스트림으로 입력되면 다양한 시그널 프로세싱 유닛에 의해서 처리된 후에 스트림으로 출력된다. VPP에서는 이러한 특성을 매크로 파이프라인 기법을 통해서 잘 활용하고 있으며, 이러한 구조를 지원하기 위해서 두 가지 인스터럭션 집합이 제공되고 있다. 각각은 매크로 ISA(macro Instruction Set Architecture), 마이크로 ISA(micro Instruction Set Architecture)이다. 이와 같은 방식은 Imagine Steam Architecture, VIRAM(Vector Intelligent RAM) 벡터 구조 등에서 사용하고 있다. 매크로 ISA는 호스트 프로세서(VPP에서는 arm core)에서 제공하고 있으며, 단순한 스칼라 인스트럭션이고, 마이크로 ISA는 벡터 SIMD 스타일의 인스트럭션으로 구현되어 있다.

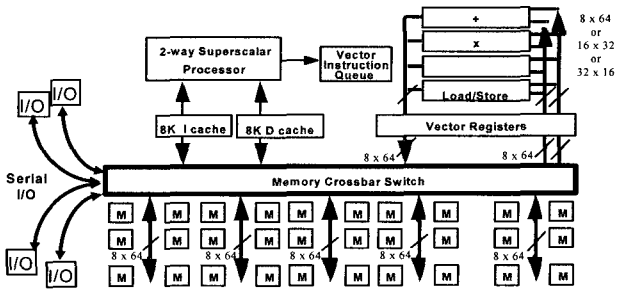
VPP 구조는 그림과 같이 크게 두개의 모듈로 나누어진다. 첫 번째는 VPP 제어기(controller)이며 나머지는 파이프라인 처리 유닛(PPU: Pipeline Processing Units)의 집합이다. 전체적인 구조는 스트림 아키텍처로 구성이 되어 있으며, 데이터는 데이터 캐쉬에서 읽혀진 후에 매크로 파이프라인으로 입력된다. VPP 제어기는 호스트 프로세서 코어와 상호 작용을 하며, PPU로 입력되는 데이터의 흐름을 제어해 준다. VPP 제어기는 내부 상태를 유지하기 위해서 레지스터 집합을 별도로 가지고 있다. PPU는 매크로 파이프라인의 벡터 프로세싱을 처리하며, 내부에 캐쉬 구조가 없다. 그 이유는 응용 프로그램이 스트리밍 특성을 가진다고 가정하기 때문이다. 대신에 각 PPU는 인스트럭션 버퍼(instruction buffer), 레지스터 파일(register file), 그리고 입출력 큐(input/ouput queue)를 가지고 있다. PPU들은 버스로 서로 연결이 되어 있으며, 각 PPU는 이전 PPU가 처리한 데이터를 받아서 처리 할 수 있는 구조를 취하고 있고, 응용 프로그램의 특성에 따라서 필요한 성능만큼 PPU의 개수를 조절하는 것이 가능하다. VPP 연구는 특정한 용도(무선 프로토콜과 같은 분야)에 적합하게 사용될 수 있으나, 데이터가 생산자 소비자 모델을 따르지 않는 응용에 대해서는 효과를 낼 수 없다는 단점이 있다.

VIRAM(Vector Intelligent RAM)은 멀티미디어를 처리하는 다양한 임베디드 장치를 지원해주기 위한 연구로 개발되었다. VIRAM의 주요 컴포넌트는 스칼라 코어(scalar core), 벡터 제어 유닛(vector control unit), 벡터 레인(vector lanes) 그리고 내장형 DRAM(embedded DRAM)으로 구성되어 있으며 (그림 2)는 VIRAM의 주요 컴포넌트 구성도를 보이고 있다.

스칼라 코어는 MIPS M5Kc 코어를 사용하고 있으며 내부에 8KB의 인스트럭션 및 데이터 캐쉬가 있으며, 코프로세



(그림 1) Vector Processing Pipeline 구조도



(그림 2) VIRAM 구조도

서 인터페이스를 통해서 스칼라 부동 소수 연산 유닛(scalar FPU)과 벡터 제어 유닛(vector control unit)에 연결되어 있다. 하나의 벡터 레인은 8KB의 벡터 레지스터 파일을 가지고 있으며 32개의 레지스터에 32 64-bit, 64 32-bit 또는 128 16-bit의 엘리먼트를 가질 수 있다. 각 벡터 레인은 미디어 응용 프로그램을 지원해주기 위해 두 개의 산술 유닛과 로드(load)/스토어(store) 유닛 그리고 플래그 처리 유닛을 제공한다. 또한 VIRAM에서는 고정 DSP 유형 오퍼레이션(fixed-point DSP-type operations)과 RGB 픽셀 처리를 위한 메모리 오퍼레이션(load sequential, every other, every third memory element), 그리고 이미지 및 비디오 처리에서 종종 사용하는 FFT(fast Fourier transforms)의 성능을 향상 시킬 수 있는 감축(reduction) 오퍼레이션을 지원해준다. 하지만 VIRAM의 경우도 앞에서 살펴본 VPP 기법과 마찬가지로 특수 목적의 응용에 적합한 형태로 설계되었으며 범용적인 벡터 처리를 지원하기에는 부족하다. VPP 프로그래밍 기법과 VIRAM 방식 이외에도 벡터 처리 기법을 적용하여 다양한 분야에서 성능을 향상하려는 노력이 최근에 다양한 분야에서 연구되고 있다[10, 11, 12, 13, 14, 15, 16].

3. 범용 벡터연산을 위한 Vector Processing Unit 설계

컴퓨터 처리 기법에서 하나의 연산자(○)로 여러 개의 데이터 요소를 처리하는 것($C[i]=○A[i], C[i]=A[i]○B[i], i=0,1,2,3,\cdots,n-1$)을 벡터처리라 한다. 여기서 벡터 요소들은 하나의 외형적 구조를 가지며, i는 각각의 요소들이 벡터구조 내에서 갖는 고유의 순서 내지는 인덱스라 한다. 초기 본 노이만(von Neumann) 컴퓨터에서 드럼 메모리의 기계식 회전속성이 명령어 액세스를 위한 명령어 계수기(PC)로 전환 설계되어 중앙처리장치(CPU)안으로 들어왔으나, 데이터는 외형적 구조 크기가 없는 단일 요소의 스칼라, 즉 SISD 개념의 스칼라 처리를 위한 설계이므로 수행 중에 오퍼랜드의 순서를 계수하는 데이터 계수의 필요성을 느끼지 못했었다. 이러한 개념(SISD)의 컴퓨터에서 n(n>2)개의 벡터 요소들이 처리되기 위해서는 n번의 SISD 수행이 필요한 것이다. 데이터 요소가 n개라는 이유로 같은 명령어가 n번 반복되어야 하고, 여기에 벡터 요소들의 주소 계산도 데이터 처리부(ALU)에서 주소 처리(addition, increment)가 필연적으로 n번 수행되어야 한다.

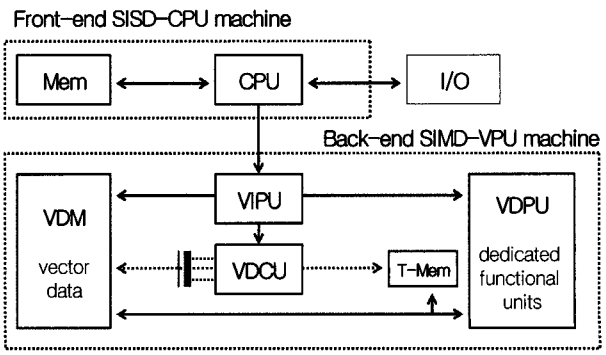
이런 처리기법은 데이터 처리부의 원래 목적인 데이터의 수·논리 처리 외에 (n-1)번의 메모리 액세스와 데이터 주소계산의 부담까지 추가되는 것이다. 다시 말해, 본 노이만 개념(SISD)은 하드웨어적 오퍼랜드 계수 개념 설계가 없기 때문이다. 이렇게 주소 계산 처리와 데이터 처리를 하는 데이터 처리부의 수행시간(clock period) 부담은 SIMD 개념에 의한 데이터 처리부 외부에 데이터 계수기의 설계로 완화될 수 있는 것이다. 그러므로 메모리에 집단적 연속성의 벡터 요소를 계수하여 데이터 처리부에 공급할 수 있는 데이터 계수장치(data counter)를 중앙처리장치(CPU) 또는 데이터 처리부 외부에 하드웨어로 설계해야 된다.

벡터처리에서는 고속처리를 위해 벡터 요소들이 스트림으로 공급되어야 하기 때문에 그 기능이 없는 스칼라 컴퓨터에서는 벡터처리의 효율이 거의 없다. 즉 데이터의 스트림 공급과 처리를 위한 전용기능의 데이터 처리부는 단항벡터처리, 2-항 벡터처리, 연계 벡터처리로 분류되며 파이프라인으로 설계되어야 한다. 벡터를 고속 처리하는 슈퍼컴퓨터 CRAY는 메모리와 데이터처리부 사이에 레지스터(8x64 words)를 이용하는 레지스터간 처리순서를 갖는다. 그러나 벡터 단위로 고속처리하기 위해서는 메모리간 처리순서를 가져야 하며, 고속 스트림 처리를 위해 메모리는 병렬 모듈로 설계되어야 한다[17][18]. 이러한 문제점들을 해결하기 위하여 본 연구에서는 기존의 스칼라 컴퓨터(SISD)에 부착사용을 전제로 벡터를 고속처리 하는 벡터 처리전용 프로세서(VPU : vector processing unit)를 제안한다.

1. 벡터처리장치(VPU)는 벡터 명령처리부(VIPU: vector instruction processing unit), 벡터 계수장치(VDCU: vector data counting unit), 벡터 데이터 처리부(VDPU: vector data processing unit), 벡터데이터 메모리(VDM: vector data memory)로 구성된다.
2. 벡터는 하나의 벡터단위(vector unit)를 위해 문자변수를 사용하고, 그 벡터변수는 벡터의 구조 크기와 메모리주소를 갖는 변수 기술어(Vd:variable descriptor)를 지시한다.
3. 벡터 요소들의 주소계산을 위해 벡터 처리장치 외부에 벡터 데이터 계수장치(VDCU)를 설계하여 벡터 요소들만 계수하도록 한다.
4. 벡터 처리장치는 벡터 오퍼레이션에 따라서 전용기능을 가지는 벡터 데이터 파이프라인 유닛(VDPU)으로 설계 한다.
5. 벡터를 제외한 모든 처리는 기존의 SISD 중앙처리장치(CPU)로 하고, 벡터 처리는 SIMD 원리에 의한 벡터 처리장치(VPU)로 한다.

(그림 3) 범용 벡터 연산 처리를 위한 벡터 처리 전용 프로세서

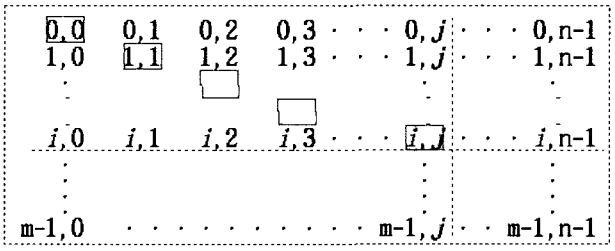
제안하는 기법에서 변수기술어(Vd)는 벡터 단위와 벡터처리에 사용되는 스칼라를 위해 사용되고, SISD 개념의 중앙처리장치(CPU)와 SIMD개념의 벡터처리장치(VPU)는 Front-Back 관계를 가지며, VPU는 CPU의 지시를 받아 벡터만 처리하는 구조를 취한다[19][20]. 전반적인 개념도는 (그림 4)와 같다.



(그림 4) SISD-SIMD의 Front-Back 개념도

3.1 벡터 주소 지정과 처리 원리

스칼라 컴퓨터에서 중앙처리장치(CPU)는 제어부(Control Unit)와 데이터 처리부(ALU)로 구성되고, 제어부는 메모리에 선형으로 저장된 명령어들을 위해 하드웨어로 설계된 계수기(PC)가 순차계수 제어(program flow control)하는 특성을 갖고 있다. 그러나 스칼라는 메모리 내에 순차적 저장이 아니기 때문에 명령어에 수반되는 주소에 의해서만 소재가 탐색된다. 벡터처리를 위해서는 벡터의 특수한 외형적 구조성과 그로 인한 높은 처리속도가 고려되어야 한다. 그림 5의 외형적 행(row)과 열(column)의 좌표와 구조의 크기[m;n]을 갖는 벡터는 벡터메모리(VDM) 내에서 m·n개의 벡터 요소들이 행계수 모드에 의해 집단적 연속형으로 저장된다[21].



(그림 5) 크기 m·n 의 벡터 [m;n] 구조

메모리 내에서 행계수 모드로 집단적이며 연속적 선형을 이루는 벡터요소(0, 1, 2,⋯,(i·n+j), ⋯, m·n-1)들은 외형적 좌표값 인덱스[i, j]와 크기[m;n]에 의해 상대주소(relative address)가 결정된다. 다음의 식(1)은 상대주소를 표현하고 있다.

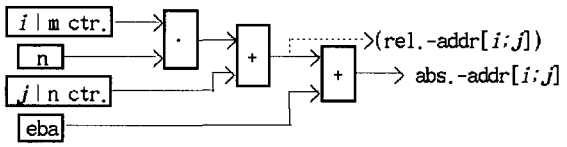
$$\text{rel.-addr}[i;j] = I \cdot n + J \quad (1)$$
$$I = 0,1,2,3,\cdots,i,\cdots,m-1,$$
$$J = 0,1,2,3,\cdots,j,\cdots,n-1$$

벡터[m;n]에서 유도된 식(1)은 I와 J의 행계수(row major)모드에서 $(i,0 \rightarrow i,1 \rightarrow i,2 \rightarrow i,3 \rightarrow \cdots [i,j] \rightarrow \cdots i,n-1)$순서로 j가 증가하여 (n-1)에 도달 할 때마다 i는 +1씩 증가하여 (m-1)까지 계수하여 인덱스[(m-1),(n-1)]에 도달한다. 열계수(column major)모드에서는 $(0,j \rightarrow 1,j \rightarrow 2,j \rightarrow 3,j \rightarrow \cdots [i,j] \rightarrow \cdots m-1, j)$순서로 i가 증가하여 (m-1)에 도달 할 때마다 j가 +1씩 증가하여 (n-1)에 도달하므로 연속 선형에서는 n 만큼의 건너뛰기(stride)로 액세스 하여 인덱스[(m-1),(n-1)]에 도달한다. 대각선계수(diagonal major)모드에서 [i,j]가 동시 증가(0,0→1,1→2,2→ ⋯[i,j]→⋯(n-1),(n-1))하여 인덱스[(n-1),(n-1)]에서 완료된다. 다시 말해, N-모듈로 계수(modulo- N count)에 의한 2-차원계수기법(2-dimensional count mechanic)의 원리이다.

인덱스 i와 j는 벡터구조[m,n]내에 행-, 열-, 대각선 순서로 상대주소(relative address)를 생성하고, 실기본주소(eba: effective base- address)에 가산되어 메모리 내에 선형을 이루고 있는 벡터 요소들의 절대주소(absolute address)로 전환되어 다음의 식(2)으로 표현된다.

$$\text{abs.-addr}[i;j] = \text{eba} + (i \cdot n + j) \quad (2)$$

식(2)는 i와 j가 N-모듈로 계수하여 각각의 실기본주소에 가산되는 2개의 소스데이터 카운터와 1개의 결과 벡터 또는 입력 벡터의 저장을 위해 실기본주소에 가산되는 결과 벡터 카운터로 절대주소를 스트림 생성하는 주소생성장치(address generator)가 파이프라인으로 설계되어야 한다.



(그림 6) 파이프라인 모드의 주소생성장치

3.2 벡터 명령어

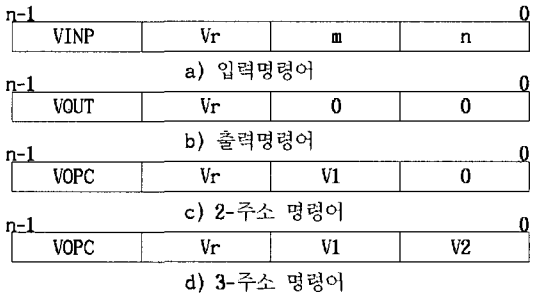
벡터 명령어는 벡터 데이터의 입·출력과 벡터처리 명령어로 분류되며, 스칼라 컴퓨터는 그들 명령어 알고리즘의 매크로루틴을, 벡터 오퍼레이션코드(VOPC)에 의한 처리를 벡터 프로세서(VPU)가 수행하도록 설계된다. 입력 명령어(VINP)는 벡터($a_0\ a_1\ a_2 \cdots a_i \cdots a_{k-1}$)가 뒤따르는 immediate mode로서 벡터가 스칼라 컴퓨터 입·출력 장치를 통해서 외부로부터 입력될 때 벡터의 외형적 크기[m;n]와 첫 번째 요소가 갖는 실기본주소로 (그림 7) 형식의 변수기술어를 작성하여 참조용(lookup)으로 변수기술어 테이블(variable descriptor table)에 기록하고, 벡터는 벡터 메모리에 선형으로 저장한다. 출력명령어(VOUT)는 벡터 메모리에 선형으로 저장되어 있는 벡터 요소들을 변수기술어에 있는 벡터의 구조크기[m;n]로 출력하도록 한다.

(그림 7) 변수기술어 형식

벡터 처리명령어의 매크로루틴 수행은 하나의 벡터처리에서 소스벡터 변수로 변수기술어를 참조하여 결과 벡터의 변수기술어를 미리 작성하고, 벡터 오퍼레이션코드(VOPC)와 변수기술어들의 내용을 벡터프로세서로 보내준다. 명령어 형식은 (그림 8)과 같으며 그 오퍼레이션들은 다음의 계수 모드와 명령어를 갖는다.

a) 입력 명령어(AGr 행계수 모드) : VINP m,n a_0 a_1 $a_2 \cdots a_i \cdots a_{k-1}$ (immediate mode)
b) 출력 명령어(AGr 행계수 모드): VOUT Vr
c) 2-주소 명령어 : 감축처리(AG1 행계수 모드, AGr 스칼라 모드) : VRDT
   전치 행렬처리(AG1 행계수 모드, AGr 열계수 모드): VTRP
d) 3-주소 명령어 : 동일 좌표요소 처리(AG1,AG2,AGr 행계수 모드) : VADD,VMUL

행과 열의 곱셈 처리(AG1,AGr 행계수 모드, AG2 열계수 모드) : VRCM
벡터 스칼라 처리(AG1,AGr 행계수 모드, AG2 스칼라 모드) : VSML,VSDV
내적처리(AG1,AGr 행계수 모드, AG2 열계수 모드): INPR



(그림 8) 벡터처리 명령어 형식

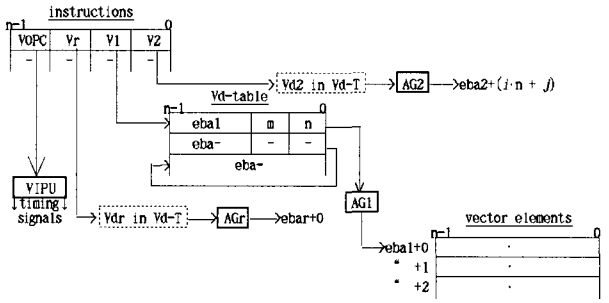
(그림 8)의 벡터 명령어는 스칼라 컴퓨터에 사용자 프로그램으로 저장되며, 명령어에 수반되는 소스벡터의 변수(V1, V2)들은 데이터 입력 시에, 결과 벡터 변수(Vr)는 스칼라 중앙처리장치(CPU)의 수행으로 단위벡터의 첫 요소가 저장된 메모리 실기본주소(ebar)와 그 벡터가 갖는 크기 [m;n]을 포함하는 변수기술어(Vdr)형식으로 처리되어 스칼라 메모리에 저장되고 벡터메모리(VDM)에는 벡터 데이터만 저장된다.

스칼라 컴퓨터로부터 벡터 오퍼레이션코드(VOPC)와 변수(V1,V2,Vr)들의 해당 변수기술어(Vd1, Vd2,Vdr)를 넘겨받은 벡터 명령처리부(VIPU)는 벡터 오퍼레이션코드(VOPC)와 벡터구조의 해독으로 변수기술어를 벡터데이터 카운터(VDCU)로 넘겨주고, 오퍼레이션코드(VOPC)와 벡터크기 [m;n] 에 따라서 연동되는 각 유니트들의 수행시차에 맞추어 필요한 처리제어 시그널을 보낸다. 본 설계 개념에서 벡터처리 오퍼레이션들은 메모리간 수행으로 고속 파이프라인 처리를 한다.

3.3 벡터의 주소지정

본 개념의 주소지정 기법은 벡터단위의 데이터를 벡터 데이터 메모리(VDM)에 입·출력시키기 위해 메모리내의 장소(주소)를 지정하는 방법이다. 그러므로 외부로부터 또는 오퍼레이션 수행의 결과저장에 의한 입력과 시스템 외부로 또는 오퍼레이션 수행을 위해 메모리로부터 출력하는 것이다. 이러한 주소지정은 식(2)에 의한 설계구조를 갖는 3개의 주소생성장치(AGr, AG1, AG2)들로 구성된 벡터 데이터 카운터(VDCU)에 의해 수행된다. 벡터변수(V1, V2)들은 벡터의 입력 저장시에 스칼라 컴퓨터에 의해 (그림 7)의 형식으로 변수기술어 테이블(Vd-T)에 정리된 변수기술어(Vd1, Vd2)를 참조하게 된다. 또 벡터 오퍼레이션 명령을 스칼라 컴퓨터가 매크로루틴 수행에 의해 벡터 오퍼레이션 결과로 생성될 결과 벡터(Vr)의 해당 변수기술어(Vdr) 내용을 생성하고 테이블에 정리한다. 오퍼레이션 처리를 위한 벡터메모리 출력은 벡터변수(V1, V2)에 의해 변수기술어 테이블에서 해당 변수기술어 내용을 참조하여 벡터데이터 카운터(VDCU)의 주소생성장치(AG1, AG2)로 보낸다. 결과 벡터의 메모리 저장은 결과변수(Vr)에 의해 변수기술어 테이블(Vd-T)에서 결과 변수기술어(Vdr)를 참조하여 결과 벡터 주소생성장치(AGr)가 결과 벡터의 메모리 저장을 위해 절대주소를 행계수 모드로 생성하게 한다.

특히 2개의 소스벡터(V1, V2)를 액세스 할 경우(AG1∧AG2), 데이터 처리부(VDPU)에 오퍼랜드를 쌍으로 공급하기 위해 V1과 V2의 요소별 주소들이 행계수 모드(내적처리에서 AG1은 행계수, AG2는 열계수)로 교차 생성된다. 이 기법은 변수기술어(Vd)를 참조하는 변수기술어 주소지정 기법(variable descriptor addressing mode)으로, 변수기술어 간접기법(variable descriptor indirect mode)을 포함하여 유일하게 본 설계개념(VPU)에만 있는 고유의 벡터 주소지정 기법이다. 그러므로 식(2)에 의한 주소생성장치(AGr, AG1, AG2)들은 오퍼레이션에 따라 다르게 서로 유기적 관계로 수행해야 한다.



(그림 9) 벡터 액세스 순서도

외부로부터 벡터를 입력하는 명령어는 (그림 8) (c)의 벡터 데이터가 뒤따르는 immediate mode 명령어 형식을 가지며, 그 벡터의 외형적 구조 [m;n]는 실기본주소(eba)와 함께 (그림 7)의 변수기술어(Vd-) m, n 란에 기록된다. 결과 변수(Vr)의 결과 벡터는 주소생성장치(AGr)의 행계수 모드에 의해 벡터메모리에 저장된다. 벡터 스트림의 메모리 입·출에서 충돌을 피하기 위해서는 오퍼레이션 수행 동안에 결과 벡터(Vr)의 주소생성(AGr)은 벡터처리 오퍼레이션과 동시에 중간 값들을 임시메모리(T-Mem:temporary Memory)에 임시저장 하였다가 오퍼레이션이 끝난 다음에 벡터메모리(VDM)로 옮기기 위해 다시 한번 생성되어야 한다. (그림 9)와 같이 벡터 명령어로부터 벡터메모리 액세스까지 수행되기 때문에, 프로그램상에 데이터 액세스 이동을 위한 별도의 명령어는 설계되지 않는다. 본 설계의 벡터처리기법은 CRAY의 벡터 레지스터에 의해 64개 요소단위로 처리하는 "short vector machine"기법이 아니라, 벡터 처리에서 벡터 크기에 제한이 없는 "long vector machine" 처리 기법이다.

3.4 VPU 기법의 데이터 처리 분석

데이터 처리 파이프라인 설계기법에는 기능별 실시간 조

립형(reconfigurable)과 전용기능 고정형(dedicated functional units)이 있다. 전자는 스칼라 처리지향이고, 후자는 벡터 처리지향이라 할 수 있다. 본 연구에서 제안하는 기법은 벡터 메모리로부터 액세스되는 벡터 스트림의 고속처리를 위해 데이터 처리부는 전용기능 고정형으로 설계되어야 한다.

벡터 데이터 처리부(VDPU)는 단항처리와 2-항 처리를 위한 2종류의 오퍼레이션 전용 파이프라인으로 분류 설계되어야 한다. 즉 단항처리에는 요소별 처리(Km)와 감축처리(Kr), 2-항처리 등의 전용기능 파이프라인(Kd)이 있다. 다시 그들 파이프라인은 fixed-point와 floating-point로 분리 설계되고, 필요에 따라서는 오퍼레이션 전용 유닛이 더 증설될 수도 있다. 앞에서 열거된 벡터처리 명령어 외에 전치행렬 명령어(VTRP)는 벡터의 외형적 구조를 바꾸기 위한 오퍼레이션이므로 데이터 처리부(VDPU)의 파이프라인 처리수행은 거치지 않고, 임시 메모리(T-Mem)만 거친다. 그 외의 소스벡터 항수에 따라서 벡터처리를 위한 명령어들은 벡터처리 전용기능 고정형 파이프라인과 유닛별로 그 처리시간을 다음과 같이 분석 할 수 있다.

3.4.1 단항벡터 처리

a) 요소별 처리 파이프라인(Km)

하나의 단위벡터를 소스로 입력하여 각 요소별로 결과 값이 산출되도록 처리한다. 결과 벡터의 크기(m·n)는 소스벡터의 크기와 같다. 입력구조에서 오퍼랜드의 방향 분배가 필요 없으며, 따라서 수행상에 입력분배로 인한 시간지연도 발생하지 않는다. 파이프라인 스테이지의 수행시간은 프로세서의 클럭 속도(CP:clock period)이므로 파이프라인 절대처리 시간(Km=스테이지 수)과 m·n개의 요소 처리시간(Tpm)은 다음의 식(3)으로 정리될 수 있다. 해당 오퍼레이션의 명령어는 VSQR(vector square root)가 있다.

$$Tpm = Km + m \cdot n \quad (3)$$

b) 감축처리 파이프라인(Kr)

하나의 소스벡터단위를 입력으로 하나의 스칼라 결과 값을 내기 위한 파이프라인이다. 즉 $s = a_0 \circ a_1 \circ a_2 \circ a_3 \circ, \cdots, \circ a_i, \cdots, \circ a_{m \cdot n-1}$과 같은 오퍼레이션이다. 여기서 감축처리는 최종 스칼라 값이 나오기까지 임시 중간 값의 되돌림(feedback) 현상이 있으며, m·n개의 요소들이 모두 입력된 후 Kr개의 되돌림 합병 수행과정에서 발생하는 Kr개의 오퍼랜드 합병 지연시간(merging delay time)이 오퍼레이션 지연시간(mt:$mt=X\cdot(2X+i)+2\cdot i=X\cdot Kr+2\cdot i$,$(X = \lfloor \log 2Kr \rfloor, Kr=2X+i, 0 \le i<2X)$)으로 연장된다. 벡터 요소 m·n개의 입력으로 스칼라(s)가 출력되는 전체 감축처리시간(Tpr)은 다음 식(4)으로 정리되며, 해당 오퍼레이션의 명령어는 VRDT(vector reduction)가 있다.

$$Tpr = mt + m \cdot n \quad (4)$$

3.4.2 2-항벡터 처리

2개의 단위 벡터에서 같은 좌표의 요소들이 쌍으로 입력되어 하나의 단위 벡터를 결과로 처리해 내는 오퍼레이션에서는 벡터 A(=V1)와 벡터 B(=V2)의 각각 n개의 요소들이 $a_0b_0, a_1b_1, a_2b_2, \cdots, a_ib_i, \cdots, a_{n-1}b_{n-1}$ 순서로 a_i와 b_i가 각각 행계수 모드로 교차 입력되어 행요소들로 처리된다.

내적처리의 전반부 오퍼레이션에서 벡터 A(=V1)와 벡터 B(=V2)에서 각각 n개의 요소들이 $a_0b_0, a_1b_1, a_2b_2, \cdots, a_ib_i, \cdots, a_{k-1}b_{k-1}$ 순서로 a_i는 행계수, b_j는 열계수로 교차 입력되어 파이프라인(Kd)처리 되어야 한다. 전체적으로는 2·n개의 요소가 입력되며, 파이프라인 입력에서 방향 분배가 있어야 하고 파이프라인 스테이지 수행시간은 클럭 속도(CP)의 2배로 설계해야 한다. 파이프라인 처리시간(Tpd)은 다음 식(5)으로 정리된다.

$$Tpd = Kd + 2 \cdot n \quad (5)$$

벡터스칼라 나눗셈(VSDV:vector/scalar)경우 매크로루틴 알고리즘으로 젯수(scalar divisor)의 역수 값(1/n)을 스칼라 컴퓨터(SISD)에서 처리하여 VPU의 벡터스칼라 곱셈(VSML)으로 처리한다. 해당 오퍼레이션의 명령어는 VADD, VMUL, VRCM, VSML, VSDV 등이 있다.

3.4.3 벡터연계 처리

연계처리(Kc)는 3.4.2의 2-항 처리 파이프라인(Kd:VSML)에 해당하는 전반처리와 3.4.1.b)의 파이프라인(Kr:VRDT)에 해당하는 감축처리가 후반처리로 연계되어 하나의 유니트(Kc)로 오퍼레이션을 수행하는, 즉 내적처리 오퍼레이션 전용 파이프라인이다. 전반 2-항처리 오퍼레이션에서 소스벡터 A[m;k]는 행계수 모드, B[k;n]는 열계수 모드로 2·m·k·n개 요소들이 스트림 액세스 되고, 방향분배 입력으로 2-항처리되어 k개의 요소($c_0, c_1, c_2, \cdots, c_i, \cdots, c_{k-1}$)를 갖는 m·n개의 다중 벡터(k·m·n)인 전반처리 결과를 임시결과로 낸다.

임시 결과(k·m·n)들은 3.4.1.b)의 감축처리 파이프라인(Kr) 오퍼레이션으로 들어가게 된다. 여기서 k개 요소단위로 처리되며, b)의 마지막 Kr개의 합병처리에서 지연시간(mt)이 발생하게 된다. 이것은 다중벡터 감축처리(multiple vector reduction)로 벡터 스트림($s_{i+1}=c_0+c_1+c_2\cdots+c_i \cdots+c_{k-1}$)의 입력은 벡터 스트림($s_i$)의 입력보다 mt+a 로 지연되어야 한다. 이것은 감축처리 결과 값(s_i)이 나오는 시간은 b)의 감축처리 수행에서 발생하는 되돌림(feedback) 현상의 시간지연(mt)에 의해 k개의 요소가 입력되는 시간보다 mt만큼 더 길게 처리되기 때문이다.

여기서 m·n개의 벡터 요소(s_i)를 버퍼(FIFO-queue)사용에 의한 연속처리(Spc) 하드웨어 설계로서 처리해 내기까지는 식(6)의 시간이 지연된다. 두 번째 설계로는 b)의 감축처리 유니트(Kr)를 다중 병렬처리(Ppc)로 설계하면 식(7)의 수행시간으로 계산된다.

$$Spc = (k + mt)\cdot m\cdot n + Kr \quad (6)$$

$$Ppc = k\cdot m\cdot n + mt + Kr \quad (7)$$

식(6)에 사용되는 버퍼(FIFO) 용량과 식(7)에 사용되는 감축처리단위 유니트(Kr) 수는 스테이지 수(Kr)에서 기인되는 합병지연시간(mt)에 의해 상수로 설계된다. 연속처리설계(Spc)가 병렬처리설계(Ppc)보다 m·n·mt 만큼 처리시간이 더 길다는 것을 알 수 있다.

세 번째 벡터연계 처리방법으로 감축처리 유니트(Kr) 자체를 설계하지 않고, 벡터 2·m·k·n을 기존 3.4.2)의 2-항처리 유니트(Kd)에서 VRCM 명령(행과 열의 2항곱셈)처리하고, 중간결과 값 m·k·n개를 임시저장 메모리(T-Mem)에 저장 후 다시 감축처리 유니트(Kr) 대신에 2-항처리 유니트(Kd)로 처리하여 m·n개의 스칼라를 얻도록, 즉 2·m·k·n → m·k·n → m·n 로 오퍼레이션 단위를 소프트웨어적으로 연계처리 시키는 방법이다. 해당 오퍼레이션의 명령어는 INPR (inner product) 이다.

위에 설명된 3가지 설계기법들은 각각 나름대로 장단점이 있으나, 연계처리 절대시간으로 본다면 식(7)에 의한 병렬처리설계가 가장 빠른 것이라 할 수 있다. 회로 설계와 제어의 복잡성은 있겠으나, m·n 요소들 간에 시차가 없는 출력으로 인해 영상처리(image processing)에서 처리시차가 시각적으로 발생하지 않기 때문에 특수 영상처리(MRI,CT 등)에 적합한 설계라 할 수 있다.

3.4 벡터 저장장치(VDM)

벡터메모리(VDM)은 벡터프로세서(VPU)에서 처리될(된) 벡터데이터만을 저장하며, 높은 대역폭을 위해 주 메모리(VDM)와 임시 메모리(T-Mem) 2개로 병렬 모듈로 설계된다. 주 메모리는 2개의 소스 오퍼랜드(V1,V2)가 동시 교차 액세스가 가능하며, 벡터 오퍼레이션 수행동안 벡터의 메모리 입·출력에서 액세스 스트림의 충돌을 피하기 위해 데이터 처리부(VDPU)의 오퍼레이션의 결과 값들이 임시 저장되도록 임시 메모리(T-Mem)를 설계한다. 벡터데이터 카운터(VDCU)의 주소생성장치(AG1,AG2,AGr)들로부터 발생되는 전체 n 비트의 절대주소(eba + (i·n+j)) 비트에서 단순 분리된 하위 m 비트는 모듈주소(Au : $0 \le Au \le 2^m-1$)로, 상위 (n-m) 비트는 모듈내의 벡터주소(Av : $0 \le Av \le 2^{n-m}-1$)로 설계되어야 한다. 따라서 메모리 모듈 수는 2^m개로 설계되고, 모듈주소 m 비트를 제외한 나머지 상위 (n-m) 비트는 하위 m 비트가 지정한 모듈 내에서 액세스 될 데이터 요소의 주소를 의미한다. 모듈주소(Au)와 그 모듈 내에 벡터주소(Av)는 식(8), (9)로 표현된다.

$$Au = (eba + (i\cdot n + j)) \ \% \ 2^m \quad (8)$$

$$Av = (eba + (i\cdot n + j)) \ / \ 2^m \quad (9)$$

임시 메모리는 주 메모리보다 비교적 적은 용량의 메모리로 저장시에는 AGr의 행계수 모드로 실기본주소(eba)없는 상대주소(rel.-addr=i·n+j)만으로 액세스하며, 연계처리(inner product) 오퍼레이션에서 중간 값들이 저장되기도 한다. 데이터 처리부(VDPU)의 벡터처리 오퍼레이션이 완전히 끝난 뒤에 실기본주소(eba)의 가산과 더불어 절대주소(abs.-addr = eba + (i·n+j))를 AGr의 행계수 모드에 의해 주 메모리(VDM)에 옮겨지게 된다.

3.5 VPU 기법의 장단점 분석

본 연구는 기존의 SISD 원리와 SIMD 원리의 병행적 개념(Front-Back) 설계라 할 수 있다. 다시 말해, 기존 SISD 개념의 스칼라 컴퓨터에 SIMD 개념의 벡터프로세서를 부착하여 사용할 수 있도록 설계한 것이다. 그러나 벡터프로세서에는 SISD 개념의 명령어계수기(PC)는 설계되지 않기 때문에 기존 스칼라 컴퓨터가 소유하는 기능은 모두 사용되며, 스칼라 컴퓨터(Front-end)는 벡터프로세서(Back-end)의 벡터처리를 위해 다음과 같은 역할을 한다.

- 외부와 연결통로
- 벡터를 제외한 모든 정보 저장
- 컴파일링 및 주변기기 제어
- 응용 프로그램 및 벡터처리 매크로루틴 수행
- 벡터 변수단위의 실기본주소 정리 및 기록
- 벡터 입력 및 처리에 의한 변수기술어 생성처리
- 벡터프로세서 처리지시

스칼라 메모리에 있는 응용 프로그램에서 벡터 명령어들은 해당 알고리즘의 매크로루틴을 스칼라 컴퓨터가 수행하게 된다. 즉 내적처리 오퍼레이션 경우 명령어(VOPC Vr, V1, V2)에서 수반되는 벡터변수(V1, V2)는 벡터 입력시 스칼라컴퓨터(CPU)가 수행처리하여 저장해둔 변수기술어 테이블(Vd-T)을 참조한다. 아직 테이블에 기록되지 않은 결과변수 Vr의 변수기술어(Vdr)는 소스벡터 V1와 V2의 변수기술어(Vd1, Vd2)에 있는 크기[m, k]와 [k, n]에 의해서 Vr의 크기[m;n]와 실기본주소(ebar)로 변수기술어(Vdr)를 생성 처리하여 변수기술어 테이블에 기록하고, 벡터프로세서에서 벡터처리를 위해 오퍼레이션코드(VOPC)와 벡터변수 Vr, V1, V2의 해당 변수기술어(Vdr, Vd1, Vd2)들의 내용을 벡터프로세서(VPU)의 명령처리부(VIPU)로 보낸다.

벡터 명령처리부(VIPU)는 오퍼레이션코드(VOPC)의 해독과 더불어 변수 Vr, V1, V2들의 해당 변수기술어(Vdr, Vd1, Vd2) 내용들을 다시 벡터 데이터카운터(VDCU)의 AGr, AG1, AG2로 보내고, 주소생성장치(AG-)의 계수모드, 벡터메모리(VDM)의 액세스 모드(R/W), 오퍼레이션에 따르는 전용 파이프라인 선택과 유닛에 따르는 파이프라인 제어처리 및 벡터 오퍼레이션 전체를 제어한다. 벡터프로세서(VPU)에서 벡터데이터 카운터(VDCU)의 주소 생성장치, 병렬모듈메모리(VDM)의 액세스, 기능전용 데이터 처리부(VDPU)는 모두 파이프라인으로 설계되기 때문에 벡터명령처리부(VIPU)는 유닛별 수행시차(vector start-up times)에 맞추어 제어 수행한다. 스칼라 컴퓨터로부터 연속된 벡터처리 명령어

(i+1) 전송은 벡터프로세서(Back-end)에서 현재 수행처리중인 명령어(i)의 벡터 크기에 따라 충분한 시차를 두고 이루어져야 한다.

4. 시뮬레이션 및 성능 평가

본 연구에서는 같은 크기의 벡터를 스칼라 컴퓨터 처리기법과 CRAY 처리기법, 본 연구에서 제안하는 VPU 처리기법을 구조성 데이터의 2-차원적 액세스 처리기법에 대해 각각 시뮬레이션하고 수행 처리시간을 측정하여 성능을 비교 검토하여 평가 하였다. 시뮬레이션은 벡터 동일좌표 처리와 벡터 내적 처리에 대해서 수행 시간을 측정 하였으며 벡터 요소 수(64의 배수)로 하여 벡터의 동일좌표 처리 및 내적 연산에 대해서 성능을 측정하였다. 실험에 사용된 컴퓨터 사양은 Intel Pentium 4 2.8 GHz, 512 MByte RAM, IDE 방식의 80GByte HDD이며, 운영체제는 Linux kernel 2.6 상에서 수행되었다. 실험은 다음과 같이 4가지 방식에 대해서 결과를 구하였다.

a) REAL : PC 상에서 랜덤(random)하게 생성한 벡터 데이터에 대해서 벡터 동일 좌표 연산 및 벡터 내적에 대한 처리 속도를 구하였다.
b) SCALAR : 스칼라 방식으로 동작되는 프로세서 아키텍쳐에 대한 모델링을 하였으며, 메모리 접근 시간, 프로세서의 데이터 처리 시간, 캐쉬 히트율(cache hit ratio) 등을 고려해서 수행 시간을 측정하였다.
c) CRAY : 64개 단위로 벡터 연산을 수행하는 CRAY 아키텍쳐를 모델링 하였으며, 벡터 프로세서 유닛이 5개 있다고 가정을 하였으며, 메모리 접근 시간은 스칼라 방식과 동일한 값을 사용하였다.
d) VPU : 본 논문에서 제안하는 방식으로 Front-end SISD-CPU 모듈에서 VDM으로 전송된 벡터 데이터를 CRAY와 동일하게 5개의 벡터 프로세싱 유닛이 있다고 가정을 하여 수행 결과를 구하였다. 이때 T-Mem에 접근하는 속도는 일반 메모리에 접근하는 속도와 동일한 값을 사용하였다.

(그림 10) 시뮬레이션 방식

시뮬레이션을 위해서 각 아키텍쳐에 대해서 수행 시간을 결정하는 요소들을 추출하였다. 스칼라 방식으로 수행되는 경우에는 다음과 같은 요소를 사용하였다.

- SC_ADDR_GEN_TIME : 주소 생성 시간
- SC_MEM_ACC_TIME : 메모리 접근 시간
- SC_DATA_PROCESS_TIME : 데이터 처리 시간
- SC_CACHE_ACC_TIME : 캐쉬 접근 시간
- SC_CACHE_HIT_RATIO : 캐쉬 히트율

스칼라 방식은 동일 좌표 연산 및 내적 연산 등에 있어서 주소 생성 시간 및 메모리 접근 시간, 그리고 데이터 처리 시간 등이 각 요소 수만큼 발생하게 된다. 다만, 메모리에서 데이터를 가져오는데 있어서 블록 단위로 캐쉬에 이동이 되기 때문에 캐쉬 효과를 고려해야 정확한 시뮬레이션이 될 수 있다. 이를 위해서 본 실험에 있어서는 캐쉬 히트율을 90%로 설정하였다. 시뮬레이션 계산은 벡터 데이터의 요소 수에 대해서 주소 생성 시간, 메모리 접근 시간, 데이터 처리 시간을 모두 합산하였으며, 캐쉬 히트율을 고려한 메모리 접근 시간을 사용하여 시간을 측정하였다.

CRAY 방식으로 동작되는 경우에 대해서는 다음과 같은 요소를 사용하였다.

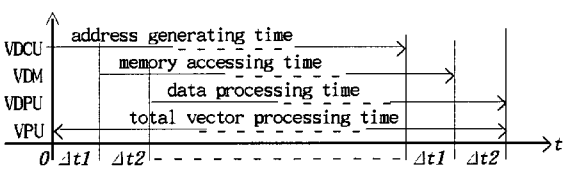
- CR_ADDR_GEN_TIME : 주소 생성 시간
- CR_MEM_ACC_TIME : 메모리 접근 시간
- CR_DATA_PROCESS_TIME : 데이터 프로세싱 시간

CRAY 처리기법은 데이터(주소) 처리부가 64개 요소 단위로 벡터 주소를 계산하고 그 주소들에 의해 64개의 요소 단위로 벡터 레지스터를 거쳐 처리한다. 다시 말해, 64개 단위의 주소처리와 벡터 요소처리를 레지스터간 중복 처리하는 기법이다. CRAY 방식을 VPU와 동일한 조건에서 비교해야 하므로 CRAY와 VPU 구조에서 차이가 나는 부분인 벡터 요소를 메모리에서 가져와서 처리하는 과정에서 발생하는 메모리 접근 횟수가 얼마나 줄었는지가 중요한 요소가 된다. 따라서 시뮬레이션은 64개 요소 단위로 벡터 주소를 계산하는데 있어서 주소 생성 시간, 메모리 접근 시간 그리고 데이터프로세싱 시간이 어느 정도 소요되는지 측정하였다. 벡터 방식은 데이터를 캐쉬하지 않으므로 캐쉬 부분에 대한 고려가 없다.

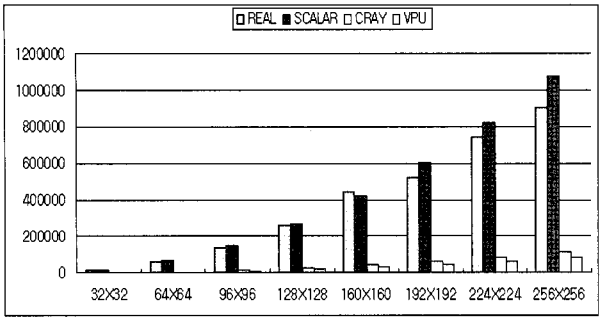
VPU 방식에서는 다음과 같은 요소를 사용하였다.

- VPU_HOST_ADDR_GEN_TIME : 호스트의 주소 생성 시간
- VPU_MEM_TO_VDM : 메모리에서 VDM으로 데이터 복사 시간
- VPU_VDM_TO_TMEM : VDM에서 임시 메모리인 TMEM으로 복사 시간
- VPU_HOST_PROCESS_TIME : 호스트의 데이터 프로세싱 시간
- VPU_VDPU_PROCESS_TIME : VDPU의 데이터 프로세싱 시간

VPU의 메모리간 처리기법은 변수 V1, V2가 변수기술어(Vd1, Vd2)를 참조하여 벡터데이터 카운터(VDCU)에 의해 주소(a_i, b_i)를 한 단위의 벡터에서 벡터 길이(CRAY 경우 64개)에 제한 없이 m·n만큼의 스트림을 생성하므로 영상처리 등에 유리한 설계이다. 다시 말해, 벡터데이터 카운터(VDCU)의 주소생성, 메모리(VDM) 액세스, 데이터 처리부(VDPU)의 처리, 즉 3가지 처리가 마치 하나의 파이프라인



(그림 11) VPU의 각 유니트에 의한 시간중복처리도

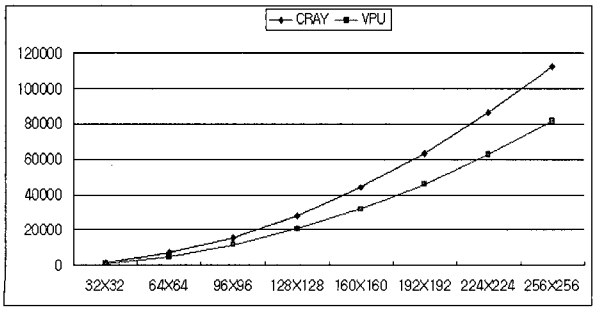


(그림 12) REAL, SCALAR, CRAY, VPU의 동일 좌표 처리 결과

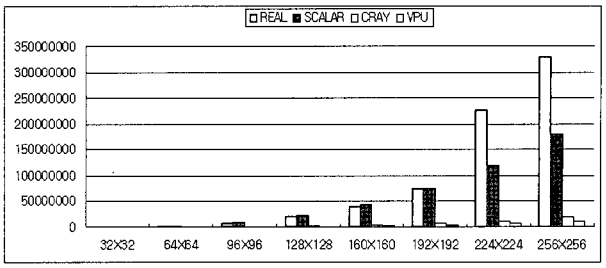
수행으로 (그림 11)과 같이 중복처리로 수행되므로 요소 수에 따르는 오퍼레이션 처리시간은 주소 생성시간(VDCU)에 벡터 액세스와 데이터 처리 수행시차(vector startup times: $\Delta t1$, $\Delta t2$)를 연장한 것과 같다고 할 수 있다. 그러나 여기서 n>>0 이면, ($\Delta t1$, $\Delta t2$)≅0 로 간주 된다.

(그림 12)에서는 REAL, SCALAR, CRAY, VPU 방식으로 동일 좌표 처리 시간을 측정하였으며, X좌표는 벡터 연산에 사용된 배열의 크기를 의미한다. 즉 32X32는 행과 열이 각각 32개의 정수로 이루어진 배열이다. Y좌표는 수행시간이며 단위는 나노세컨드(nano second)이다. 실험 결과에서 보면 실제 배열 데이터를 가지고 PC 상에서 수행된 REAL과 PC 구조를 모델링한 SCALAR가 거의 유사한 값을 보이고 있음을 알 수 있다. CRAY 방식과 VPU 방식의 처리 결과는 VPU 방식이 처리 속도를 줄이고 있음을 보여주며, 벡터의 행과 열의 값이 커질수록 성능 차이가 크게 나타나고 있음을 알 수 있다. CRAY와 VPU의 수행 시간에 대한 부분만을 비교하면 (그림 13)과 같다. 본 논문에서 제안하는 VPU의 동일 좌표 연산에 대한 성능 향상 비율은 CRAY와 비교하여 27-30% 향상되었다.

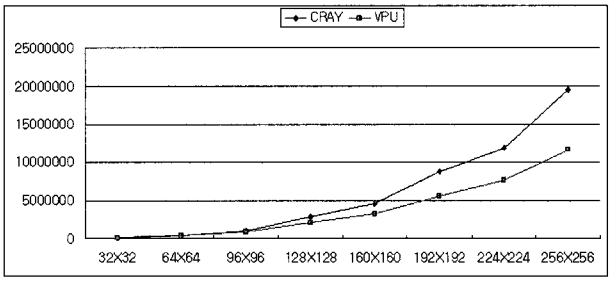
(그림 14)는 REAL, SCALAR, CRAY, VPU 방식으로 내적 처리 시간을 측정하였으며, X좌표와 Y좌표는 동일 좌표 처리 결과와 동일하다. 동일 좌표 처리 결과에 비해서 REAL과 SCALAR의 수행 시간이 배열의 크기가 커질수록 차이가 벌어지고 있으며, 이것은 프로세서 내부의 캐쉬 히트율이 떨어지기 때문에 발생하는 것으로 추론된다. 왜냐하면 작은 크기의 벡터 연산에 대해서는 캐쉬에 가져온 데이터를 이용해서 메모리에 접근하는 시간을 줄일 수 있지만,



(그림 13) CRAY, VPU의 동일 좌표 처리 결과 비교



(그림 14) REAL, SCALAR, CRAY, VPU의 내적 처리 결과 비교



(그림 15) CRAY, VPU의 내적 처리 결과 비교

벡터 연산에 사용되는 배열의 크기가 커질수록 캐쉬를 통한 이득이 점차 떨어지기 때문이다. (그림 15)는 CRAY와 VPU의 수행 시간을 자세하게 보여주고 있다. 동일 좌표 연산과 마찬가지로 내적 연산의 경우도 큰 크기의 벡터 연산에 대해서 VPU가 높은 성능을 보여주고 있음을 알 수 있다. 본 논문에서 제안하는 VPU의 내적 연산에 대한 성능 향상 비율은 CRAY와 비교하여 12-40% 향상되었다.

5. 결 론

본 논문에서는 기존 벡터 처리 방식의 문제점을 CRAY 아키텍쳐를 중심으로 찾아보았으며 이를 해결하는 방안을 제시하고 있다. 벡터를 고속 처리하는 슈퍼컴퓨터 CRAY는 메모리와 데이터 처리부 사이에 레지스터를 이용하는 레지스터간 처리순서를 갖고 있으며, 이러한 방식은 벡터 단위로 고속처리하기 위해서 성능상의 병목지점이 될 수 있기 때문에 메모리간 처리순서를 제공해 주어야 한다. 본 연구에서는 기존의 스칼라 컴퓨터(SISD)에 부착사용이 가능한 벡터 처리전용 프로세서(VPU)를 제안하고 있다. 제안하는 기법에서 변수기술어는 벡터 단위와 벡터처리에 사용되는 스칼라를 위해 사용되고, SISD 개념의 중앙처리장치(CPU)와 SIMD개념의 벡터처리장치(VPU)는 Front-Back 관계를 가지며, VPU는 CPU의 지시를 받아 벡터만 처리하는 구조를 취한다. VPU 아키텍쳐의 유용성을 증명하기 위해서 SIMD 스칼라 아키텍쳐, CRAY 아키텍쳐 그리고 VPU 아키텍쳐를 모델링하고 시뮬레이션을 수행시켰으며, 수행 결과 동일 좌표 벡터 연산 및 벡터 내적과 같은 고속의 벡터 처리 문제에 대해서 CRAY 아키텍쳐보다 우수한 처리 속도를 보여주었다.

근래의 컴퓨터 발전 동향을 보면 범용적인 시스템 환경에서 다양한 멀티미디어 응용 프로그램들이 수행되고 있으며 주어진 작업을 처리하기 위해서 대용량의 프로세싱 파워를 요구 하고 있다. 이러한 주변 환경을 고려할 때, 본 논문에서 제안하는 방법은 기존 프로세서에 VPU를 추가하는 방법을 통해서 높은 프로세싱 파워를 제공해줄 수 있으므로 고가의 장비를 대체할 수 있는 방법으로 사용될 수 있다. 향후 멀티프로세서 환경에서 부착 사용할 수 있는 벡터 처리 전용 프로세서에 대한 연구를 수행할 예정이며 이를 통해서 병렬 처리 시스템이나 실시간 시스템 등에서 활용할 수 있는 모델을 제시할 계획이다.

참 고 문 헌

[1] R.M. Russell, "The CRAY-I Computer System," Cray Research Inc. 1986.

[2] F. Boeri and M. Auguin, "OPSILA : A Vector and Parallel Processor," IEEE Trans. on Comput., Vol. 42, No.1, Jan., 1993. pp.76-82.

[3] K.L.Chung, W.M.Yan, "Fast Vectorization for Calculating a Moving Sum," IEEE Trans. on Comput., Vol.44, Nov., 1995.

[4] J.Backus, "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs," CVCM 21, 8, 1978. pp.613-641.

[5] Yuan Lin, Nadav Baron, Hyunseok Lee, Scott Mahlke, and Trevor Mudge,"A Programmable Vector Coprocessor Architecture for Wireless Applications", Proc. 3rd Workshop on Application Specific Processors, Sep., 2004.

[6] C. Kozyrakis, J. Gebis, D Martin, et al., "VIRAM: A Media-oriented Vector Processor with Embedded DRAM," In the Conference Record of the Hot Chips XII Symposium, Palo Alto, CA, August, 2000.

[7] C. Kozyrakis. "A Media-enhanced Vector Architecture for Embedded Memory Systems," Technical Report CSD-99-1059, Computer Science Division, University of California at Berkeley, 1999.

[8] C. Kozyrakis. Scalable Vector Media-processors for Embedded Systems. PhD thesis, University of California at Berkeley, 2002.

[9] C. Kozyrakis and D. Patterson. "Vector vs. superscalar and vliw architectures for embedded multimedia benchmarks," In MICRO, Nov., 2002.

[10] G. Sohi. "High-bandwidth Interleaved Memories for Vector Processors - A Simulation Study," IEEE Transactions on Computers, 42(1):34{45, January, 1993.

[11] J.E. Smith and W.R. Taylor. "Characterizing Memory Performance in Vector Multi-processors," In the Proceedings of the Intl. Conference on Supercomputing, pages 35-44, Minneapolis, MN, July, 1992.

[12] J. H. Ahn et al. "Evaluating the imagine stream architecture," In ISCA, Jun. 2004. Publishing, 1997.

[13] M. Bedford et al. "Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ilp and streams," In ISCA, Jun., 2004.

[14] K. Asanovic. Vector Microprocessors. PhD thesis, Computer Science Division, University of California at Berkeley, 1998.

[15] R. Espasa and M. Valero. "Decoupled Vector Architecture," In the Proceedings of the 2nd Intl. Symposium on High-Performance Computer Architecture, San Jose, CA, February, 1996.

[16] J.D. Gee and A.J. Smith. "The Performance Impact of Vector Processor Caches," In the Proceedings of the 25th Hawaii Intl. Conference on System Sciences, pp.437-449, January, 1992.

[17] D.H.Bailey, "Vector Computer Memory Bank Contention," IEEE Trans. on Comput., vol. c-36, No.3, Mar., 1987. pp.293-297.

[18] A.L.Decegama, The Technology of Parallel Processing, Parallel Processing Architectures and VLSI Hardware vol. 1, Prentice-Hall Int.Editions 1989. pp.71-77, 166-177.

[19] Y.i.CHO, "A Design of the Dedicated Functional Unit for Reductive Operation in Pipeline Processing," Proceedings of PARALLEL PROCESSING SYSTEM, Vol. 9, No.3, September, 1998.

[20] Y.i.CHO, H.R.Kweon, "On Design for Elimination of the Merging Delay Time in the Multiple Vector Reduction(Inner Product)," THE TRANS. OF THE KOREA INFORMATION PROCESSING SOCIETY, Vol. 7, No. 12, Dec., 2000.

[21] F. Ayres, Jr, "Theory and Problems of Matrices," Si(metric) edition, SCHAUM'S OUTLINE SERIES, 1974. McGraw-Hill.

조 영 일

e-mail : yicho@hallym.ac.kr

1971년 고려대학교 독어독문학과(학사)

1980년 Techniche Univ. Berlin(석사)

1996년 건국대학교(박사)

1982년~1985 금성사 중앙연구소 선임연구원

1986년~현재 한림대학교 정보통신공학부 교수

관심분야 : 병렬 처리, 컴퓨터 구조, 병렬 알고리즘

고 영 웅

e-mail : xyuko@hallym.ac.kr

1997년 고려대학교 컴퓨터학과(학사)

1999년 고려대학교 컴퓨터학과(석사)

2003년 고려대학교 컴퓨터학과(박사)

2003년~현재 한림대학교 정보통신공학부 조교수

관심분야 : 운영체제, 멀티미디어 시스템, 임베디드 시스템, 실시간 시스템