□ 기술해설 □

병렬처리시스템의 특징

병렬처리 컴퓨터의 특징과 처리기법

충북대학교 김석일*·이충한·지승현**
국방과학연구소 김대식***

목 차

1. 서 론

성능이 월등히 좋은 상위 기종을 총칭하는 슈퍼컴퓨터는 계산속도를 극대화하기 위한 첨단 기술의 집결체라고 할 수 있으며, 각 나라마다 보다 빠른 기종을 개발하기에 심혈을 기울이고 있다. 따라서 슈퍼컴퓨터의 구성방식은 시대에 따라 상이한 구조적 특징을 지닌다. 예를 들어, 80년대 초기에는 파이프라인방식의 벡터처리를 통하여 빠른 계산을 수행하는 방식이 주류를 이루었으나, 80년대 후반 이후 90년대에 들어와서는 단일 벡터프로세서로 추구할 수 있는 성능의 한계를 극복하기 위하여 다수의 프로세서를 연결하는 병렬처리 방식이 보편화 되었다.

이렇듯 병렬처리 기법이 보편화되면서 종래에는 상위기종에서만 발견되었던 여러가지 특징들이 하위기종에도 적용되기 시작하였고, 요즈음에는 저속의 프로세서를 대규모로 연결하여 슈퍼컴퓨터의 능력을 뛰어넘는 시스템도 등장하고 있으며, 심지어는 병렬처리가 지원되는 탁상용 PC도 개발되기에 이르렀다. 따라서 컴퓨터를 분류함에 있어서 상위기종과 하위기종을 구분하거나, 연산 능력에 따른 구분은 더이상 의미가 없어졌다. 병렬처리기법은 PC에서 슈퍼컴퓨터에 이르는 광범위한 시스템의 곳곳에 적용되고 있으며, 또한 이러한 특징이 보다 빠른 계산환경을 지원하는 토대가 되고 있다. 본 고에서는 이러한 경향에 따라서 여러가지 병렬처리 시스템의 특징을 살펴보고 병렬처리 메카니즘을 알아보고자 한다.

Flynn[1]은 컴퓨터의 특징을 크게 네가지로 구분한다. 그중에서 병렬처리컴퓨터에 해당되는 구조로 SIMD(Single Instruction-stream Multiple Data-stream)와 MIMD(Multiple Instruction-stream Multiple Data-stream)을 지목하였다. 그러나 Flynn의 구분은 요즈음의 병렬처리 컴퓨터가 여러가지 특징들을 혼합한 형태로 개발되거나 데이타 구동(data driven) 컴퓨터 등의 특징을 적시하지 못하고 있어서 여러가지 새로운

* 종신회원
** 준회원
*** 정회원

분류법이 제안되고 있다. 그러나 아직 널리 인정받는 분류법이 없는 관계로 본 고에서는 Flynn의 분류법을 근간으로 하여 벡터 연산기를 SIMD에 포함시키고, MIMD와 SIMD의 특징을 동시에 가지고 있는 MSIMD(Multiple SIMD)는 MIMD에 묶어서 다루고, 공유 메모리와 분산 메모리 구조의 특징을 동시에 가지고 있는 분산 공유메모리 컴퓨터(distributed shared memory computer)와 근래에 많은 연구가 진행중인 대규모 병렬처리 컴퓨터(MPP: Massively Parallel Processing computer)는 기존의 MIMD와 다른 여러가지 독특한 특징이 있으므로 확장성이 큰 컴퓨터(Scalable computer)로 분류한다. 또한, 함수형 계산모델을 대표하는 데이타 플로우 컴퓨터(dataflow computer)나 리덕션 컴퓨터(reduction computer) 등을 미소연산입자 컴퓨터(fine grain computer)로 구분하여 각각의 특징을 개괄하기로 한다.

2. 이상적인 병렬처리 시스템

Fortune과 Wyllie[9]가 모델화한 PRAM(Parallel Random Access Machine)은 프로세서 동기 및 메모리 참조비용이 무시될 수 있는 이상적인 병렬처리 컴퓨터이다. 그림 1과 같이 n개의 프로세서가 공유메모리와 연결되어, 모든 프로세서는 동시에 메모리를 읽거나 연산 및 메모리 쓰기를 수행할 수 있다.

PRAM은 공유 메모리를 다루는 방법에 따라 여러가지 모델로 구분된다. 즉, 한번에 하나의 프로세서만이 공유 메모리로부터 자료를 읽을 수 있도록 제한하는 ER(Exclusive Read), 메모리 쓰기가 순차적으로 수행되도록 제한하는 EW (Exclusive Write), 동시에 여러 프로세서가 공유메모리의 자료를 읽을 수 있도록 허용하는 모델인 CR(Concurrent Read), 그리고 동시에 여러 프로세서가 메모리 쓰기를 하도록 허용하는 CW (Concurrent Write) 모델이 그것이다. 이들 모델을 혼합한 PRAM 모델도 가능하다. 그 중에서 메모리 읽기와 쓰기를 순차적으로 수행되도록하는 EREW와 동시에 메모리 쓰기와 읽기가 허용

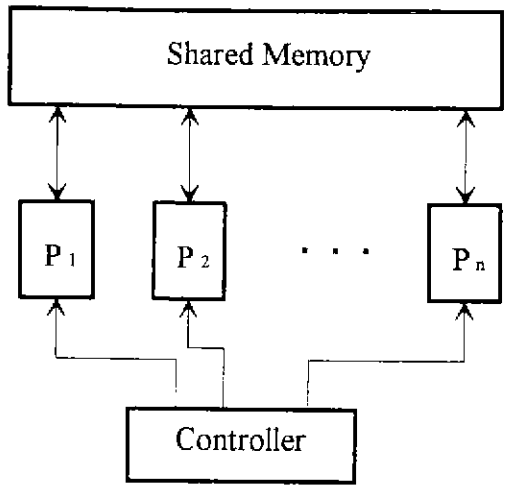


그림 1 PRAM의 구조

되는 CRCW 모델이 광범위하게 이용된다.

EREW모델은 순차처리방식을 의미하며 CRCW 모델은 병렬처리를 위해서는 가장 이상적이나, 동시에 메모리 쓰기를 허용함에 따른 충돌문제를 회피하는 방안이 마련되어야 한다. Fortune과 Wyllie[9]는 이를 해결하기 위해 핫스팟(hot spot)을 마련하거나, 쓰기를 하는 프로세서에 우선순위를 부여하는 방법 등을 제안하였다.

PRAM은 이론적인 병렬처리모델이므로 병렬알고리즘을 개발하고, 시스템의 확장성(scalability) 및 복잡도(complexity)를 분석하는 기준으로써 중요한 의미를 지니며, 앞으로의 병렬처리 컴퓨터가 발전해나가야 할 방향을 제시해 주는 면에서 가치가 있다.

3. 병렬처리 시스템의 구조와 특성

3.1 SIMD 시스템

SIMD 컴퓨터는 여러개의 연산소자(processing element)가 하나의 클럭에 동기되어 연산을 수행하는 배열(array) 구조의 프로세서로 PRAM에 가장 근사한 구조이다. 모든 연산소자들은 하나의 제어 장치(global control unit)가 제공하는 클럭에 동기되어 주어진 피연산자에 대해 동

일한 연산을 수행한다. 한편 파이프라인은 하나의 연산을 몇개의 단계로 세분화하여 모든 단계를 거쳐야 하나의 연산이 종료되는 연산 장치로, 매 클럭마다 하나의 출력을 생산한다. 따라서 파이프라인과 배열 프로세서는 클럭에 동기되는 연산이라는 점에서 유사하다.

3.1.1 배열 프로세서의 연산 모델

배열 프로세서는 여러개의 연산소자를 배열 구조로 구성하고, 연산 작업, 메모리 악세스 및 연산소자간의 자료 교환 작업등을 하나의 제어 장치의 클럭에 동기시켜 수행하는 장치이다. 따라서 배열 프로세서를 가장 효과적으로 사용하는 알고리즘은 모든 연산소자가 클럭마다 동일한 작업을 수행할 수 있는 계산 형태이어야 한다. 또한 배열 프로세서는 모든 연산소자가 동일한 경로(routing)로 이웃한 연산소자와 자료를 주고 받으므로 연산소자간의 연결구조가 기하학적으로 간단한 대칭 구조를 이룬다. 예를 들어 Illiac-IV[5]는 64개의 연산소자가 8×8 배열 구조로 되어 있으며, 각각의 연산소자가 동서남북 방향의 네 연산소자와 연결된다.

배열 프로세서는 분산메모리 모델과 공유메모리 모델로 구분된다. 분산메모리 모델은 분할된 연산 데이타가 연산소자의 제어를 받는 지역메모리(local memory)에 저장되어 빠르게 참조가 가능하도록 하며, 연산소자간의 정보교환은 연산소자간을 연결하는 네트워크를 통하여 이루어진다. 네트워크를 통한 경로배정도 중앙의 제어장치에 따라 제어된다.

이와는 달리 공유메모리 모델은 연산소자와 메모리를 분리하여 그 가운데 연산소자와 메모리를 접속하는 네트워크를 삽입한 구조이다. 이 모델에서도 네트워크의 경로배정방식은 중앙의 제어장치에 의하여 제어되나, 동일한 자료를 여러 연산소자가 참조하는 현상을 피할 수 없기 때문에 널리 사용되지 않는다. 즉, 대부분의 배열 프로세서가 분산메모리 모델의 형식을 취하고 있으며, 여러가지 다양한 네트워크를 이용하여 연산소자를 연결하고 있다. 공유메모리형 컴퓨터로는 Burroughs Scientific Processor(BSP)[12]가 있으며, 분산메모리 모델의 시스템은 이차원 배열인 Illiac-IV[5], AMT의 DAP-610 등과 하이퍼큐브 네트워크를 채용한 Thinking Machine사의 CM-2[15]및 X-Net을 채용한 MasPar사의 MP-1[16]등이 있다. 이들 네트워크는 MIMD의 네트워크와 기본개념이 같으므로 4절에서 다루기로 하자.

n개의 연산소자를 가진 배열 프로세서는 n개의 연산소자가 동시에 하나의 연산을 수행하므로 전체적으로는 하나의 클럭마다 n개의 연산을 수행할 수 있다. 즉, 공간적 병렬성(spatial parallelism)을 따르고 있으며, 연산의 수가 n의 배수인 것이 유리하다.

3.1.2 벡터 프로세서의 연산 모델

벡터 프로세서는 배열 프로세서와는 달리 연속된 입력 자료를 파이프라인에 공급하고, 파이프라인의 모든 단계를 거쳐야 완전한 출력이 얻어진다. 첫번째 출력은 파이프라인 단계가 모두 채워지는 안정상태에 도달하는 시간만큼 지연된다. 마찬가지로 입력자료의 공급이 종료되더라도 마지막 출력은 모든 파이프라인 단계가 작업을 완료하는 시간만큼 지연된다. 따라서 파이프라인의 효과는 파이프라인의 단계 및 파이프라인에 입력되는 피연산자의 길이에 의해서 결정된다. 이런 점에서 벡터 프로세서는 시간적 병렬성(temporal parallelism)을 추구한다.

벡터 프로세서에서 가장 중요한 관건은 파이프라인 방식으로 처리되는 입력 자료를 얼마나 빨리 공급하느냐에 달려있다. 따라서 대개의 경우 메모리와 파이프라인 사이에 벡터 레지스터(vector register)를 두어 일정한 양의 입력자료를 저장하고 이를 빠르게 파이프라인에 공급한다. 그러나 벡터 레지스터의 길이를 무한히 크게할 수 없으므로 전체적으로 메모리와 벡터 레지스터간의 데이타 전송속도가 벡터처리의 성능을 좌우한다. 이러한 문제점을 개선하기 위해서 빠른 메모리 참조가 가능하도록 동시에 여러 메모리 블럭으로부터 벡터값을 참조하여 이를 순차적으로 파이프라인에 공급하는 일종의 메모리 참조 파이프라인(memory access pipeline) 또는 인터리빙(interleaving)기법이 개발되었다. 그러나 이들 기법을 채용하고 있는 시스템에서도 이

러한 기능을 충분히 이용하기 위해서는 연속된 벡터를 여러 메모리 블럭에 나누어 저장하여 파이프라인에 의한 메모리 참조의 효과를 극대화해야 한다. 또한 한번에 참조하여 파이프라인에 제공할 수 있는 벡터의 길이를 크게할수록 유리하다.

최근의 벡터 프로세서들은 여러개의 파이프라인 장치를 마련하고 각 장치들이 순서대로 연결되어 연속된 벡터연산을 수행하는 슈퍼 벡터(supervector)기능이 있다[8]. 이 기능은 연속된 벡터처리시 두번째 파이프라인에 의한 파이프라인 지연시간을 줄여서 전체적으로 속도를 개선한다.

3.1.3 시스톨릭 배열과 VLIW

시스톨릭 배열(systolic array)[3]은 구조와 형태면에서 배열 프로세서와 유사하지만 연산소자간의 연결방법이 고정된 점이 다르다. 즉 각각의 연산소자에서 수행된 연산결과가 고정된 연결방법을 따라서 이웃의 연산소자로(연속적으로) 전달되어 최종적으로 원하는 결과를 출력한다. 그러므로 시스톨릭 배열은 각각의 연산소자가 완전한 연산의 일부분을 담당하는 다차원 파이프라인 연산장치 또는 벡터 프로세서에 가깝다.

VLIW[4]는 여러개의 연산장치를 마련하고 이들을 동시에 병행사용할 수 있도록 다수의 명령어를 병렬처리하는 컴퓨터구조로써 매 클럭마다 하나의 인스트럭션을 수행하므로 클럭에 동기되는 프로세서의 일종이다.

3.2 MIMD 시스템

MIMD는 여러개의 연산소자를 네트워크로 연결한 SIMD와 그 구조면에서는 유사하나 각 연산소자가 중앙의 클럭에 동기되어 연산을 수행하는 SIMD와는 달리 연산소자와 결합된 제어장치의 도움으로 다른 연산소자의 수행작업에 관계없이 자신의 속도에 따라 작업을 수행하는 점이 다르다. 이런 의미에서 MIMD에서는 제어장치를 포함하는 연산소자를 프로세서라는 용어로 표현하는 것이 옳다.

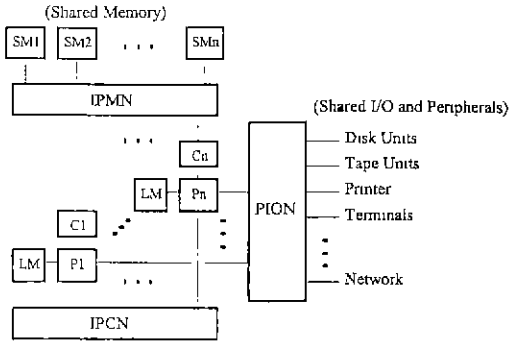
MIMD는 기본적으로 여러개의 프로세서를 네트워크로 연결한 다중 프로세서(multiprocessor)이다. 이러한 구조는 하나의 프로세서로 해결할 수 없는 연산 능력의 한계를 극복하거나 저고장율 컴퓨터 체계 (low fault-rate computer)를 실현하는 방법으로 연구되어 왔다.

MIMD에서의 프로세서간 자료종속관계는 프로세서간의 정보 교환으로 대체된다. 정보 교환은 공유 메모리를 통하는 방법과 직접 프로세서간의 통신선로를 통하여 이루어지는 메세지전송의 두가지 방법이 있다. 즉, MIMD의 특징은 어떤 방법으로 프로세서간의 통신을 처리하는지와 어떤 네트워크로 프로세서를 연결하는지에 달려있다.

3.2.1 프로세서 연결방식

MIMD의 구조는 그림 2와 같이 프로세서들과 여러개의 메모리 모듈을 네트워크로 연결한 구조이다. 즉, 프로세서들은 프로세서간 네트워크(IPCN: interprocessor communication network)을 통하여 프로세서간 통신이 가능하며, 프로세서-메모리 네트워크(IPMN: interprocessor-memory network)을 통하여 메모리 모듈과도 연결된다. 또한 여러가지 I/O장치는 마찬가지로 입출력 네트워크(PION: processor I/O network)을 통하여 여러 프로세서들이 공유할 수 있다.

그림 2에 보인 MIMD는 IPCN과 IPMN의 유무에 따라 여러가지 모델의 특징을 지닌다. 예를 들어, IPCN이 생략된 시스템은 공유메모리를



LM : Local memory
SM1-SMn : Shared memory
C1-Cn : Cache
P1-Pn : processor

그림 2 MIMD의 일반적인 구조

통해서만 프로세서간 통신이 이루어지는 공유 메모리 MIMD이다. 즉, 메모리 모듈과 연산소자 간에 네트워크를 설치하고 프로세서와 메모리간의 접속방법을 변경하면서 공유메모리에 정보를 기록하거나 참조하여 정보를 교환한다. 따라서 이 시스템은 어느 프로세서나 공유하는 메모리를 통하여 자료를 직접 참조할 수 있으므로 처리시간이 짧아 밀결합(tightly coupled) MIMD의 특징을 지닌다. 그러나 프로세서의 수가 증가하면 같은 메모리 모듈을 경쟁적으로 참조하여 성능이 저하되는 단점이 있다.

IPMN이 없는 모델은 네트워크를 통한 프로세서간의 통신으로 동기화가 이루어지는 메시지 전달(message passing) 시스템이 된다. 이 시스템에서는 프로세서간에 메시지를 주고 받음으로 프로세서간의 자료 교환이 일어난다. 따라서 통신을 수행하는데 필요한 시간이 길어 소결합(loosely coupled) MIMD의 특징을 보이나, 동시에 여러 쌍의 프로세서간에서 통신이 수행될 수 있으므로 많은 프로세서를 연결하여 MIMD를 구성할 수 있는 이점이 있다.

MIMD에 적용되는 연결방법은 그 형태에 따라 링 구조, 격자형, 크로스바 스위치(crossbar switch), 다단계 연결방법(multiple stage interconnection network) 및 하이퍼큐브 등이 있다[2]. 적용되는 네트워크의 모양은 이용하려는 연산모델과 직접적인 관계가 있다. 예를 들면, 링 구조는 좌우측의 연산소자와 밀접하게 정보를 교환하는 알고리즘에 적합하며, 격자형(mesh) 구조는 Illiac-IV와 마찬가지로 네방향의 프로세서와 정보를 교환하는 경우에, 크로스바 스위치는 임의의 프로세서간에 빠른 정보교환이 이루어져야 하는 경우에 적합하다. 다단계 네트워크는 크로스바 스위치와 같이 몇차례의 단계를 거쳐 임의의 프로세서와 통신할 수 있는 경우에 적은 비용으로 마련할 수 있는 네트워크이다. 하이퍼큐브는 자기 경로배정(self routing)이 가능하며, 링, 격자 등 여러가지 네트워크로 구조 변경이 가능하여 소결합 MIMD에 많이 이용된다. 밀결합 MIMD의 프로세서와 메모리간에는 크로스바 또는 다단계 네트워크가 많이 이용된다. 그림 3은 MIMD와 SIMD용으로 사용될 수 있는 여러가지 네트워크의 형태이다.

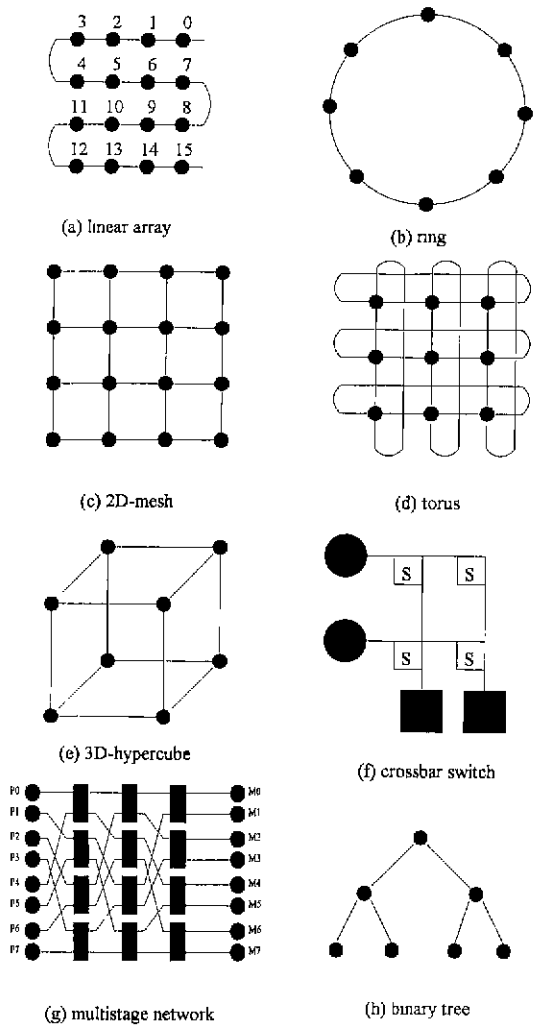


그림 3. 병렬처리시스템용 네트워크

밀결합 MIMD에서의 정보교환 속도는 소결합 MIMD에 비하여 매우 빠르며, 일반적으로 교환 비용은 계산 비용의 산정시 무시될 만큼 적다. 그러나 소결합 MIMD의 경우에는 그 비용이 무시할 수 없을 정도로 커서 어느 경우에는 통신 비용이 계산 시간을 능가할 경우도 발생한다. 따라서 소결합 MIMD에서의 효과적인 연산 형태는 프로세서간 정보 교환 횟수가 적으며 프로세서별로 병렬 처리할 계산량이 비교적 많은 작업이다.

MIMD에서 가능한 연산 모델은 LAN으로 연결된 컴퓨터에서 서브루틴을 나누어 처리하는

분산 처리 모델로 부터 사칙연산 단위의 병렬처리기법인 데이터플로우(data flow) 모델에 이르기까지 광범위하다. 그러나 연산입자의 크기를 결정하는 것은 여러가지 복잡한 시스템 특성을 모두 고려하여야 한다. 즉, 연산입자의 크기를 작게하고 이를 많은 프로세서에 분할 할수록 프로세서간의 통신요구가 빈번하며, 또한 연산입자의 크기를 크게하면 프로그램중의 병렬처리 가능성이 줄어들어 병렬처리 컴퓨터의 가용자원을 충분히 이용하지 못하게 된다. 따라서 최적의 연산입자 크기를 결정하는 것은 MIMD에서의 효과적인 계산을 위해서는 필수적이다. 이에 대한 사항은 4절에서 다루기로 한다.

3.2.2 정보의 교환방식

전통적으로 MIMD는 메모리 사용 형태에 따라 두가지로 구분하여 왔다. 즉, 공유메모리 MIMD에서는 모든 변수를 공유메모리에 저장하므로 변수의 형식에 관계없이 모든 프로세서가 함께 참조할 수 있는 전역변수(global variable)로 취급된다. 이와 반면에 개별메모리(private memory) MIMD의 경우는 각 프로세서별로 마련된 개별 메모리에 변수를 저장하므로 모든 변수가 지역변수(local variable)로 취급된다. 그러나 근자에는 분산메모리 MIMD에서도 개별메모리를 공유메모리로 다루려는 경향을 보이고 있다. 예를 들어, 어떤 지역변수 α가 프로세서 P_i에 존재할 때, 다른 프로세서 $P_j(j \neq i)$가 α를 사용할 때 P_i에 존재하는 α를 자신의 메모리로 복사하여 온 후, 이를 사용하고 곧이어 복사본을 폐기한다면 결국 P_j는 α가 필요할 때마다 매번 P_i의 α값을 참조하여야 한다. 따라서 α는 비록 P_i의 개별 메모리에 위치하고 있지만 다른 프로세서에게는 전역변수로 간주될 수 있다. 근래에는 복사본의 폐기시기를 사용직후가 아니라 어떤 프로세서가 α를 갱신하는 시점으로 계속적인 사용가능성을 향상시키거나, 갱신된 α를 갱신되기 이전의 α값을 가지고 있던 모든 프로세서로 전송하여 항상 최신값을 개별 메모리에 유지하는 방법도 연구되고 있다. 이러한 방법을 메모리 일관성 문제라고 하며, 공유메모리 MIMD에서는 캐쉬에서의 일관성 문제, 분산 메모리형 MIMD에서는 개별메모리간의 일관성 문제로 다루어진다. 이에 대한 자세한 사항은 본 특집의 기사[10]를 참고하기 바란다.

3.3 확장성이 큰 시스템(scalable system)

최근들어 반도체 제조기술의 발달에 힘입어 값이 싸며, 우수한 성능의 프로세서 개발이 가능해짐에 따라 이들을 대규모로 연결하여 슈퍼컴퓨터의 성능을 능가하는 새로운 시스템을 구축하려는 연구가 추진되고 있다. 현재 Teraflops의 성능을 내는 시스템이 연구되고 있으며, 이러한 성능은 대량의 벡터 계산이 요구되는 여러 분야에서 막강한 능력을 발휘할 수 있다. 그러나 이러한 시스템이 광범위하게 이용되기 위해서는 뛰어 넘어야 할 장벽도 만만치 않다. 가령 수백개 이상의 프로세서가 조화롭게 동작하기 위해 필요한 네트워크의 선택문제, 원거리 메모리를 참조할 때 발생하는 지연시간 문제, 많은 수의 프로세서간에 발생되는 동기화의 문제, 캐쉬 및 메모리 일관성유지 문제 등에 대한 대책이 필요하다.

이중 매우 중요한 것이 네트워크의 선택으로, 적절한 네트워크를 선택하기 위해서는 네트워크 토폴로지(network topology), 라우팅 제어기법(routing control), 데이타 전달 방법(data transfer method), 사용의 용이성, 하드웨어 구현, 성능, 결함 허용 능력(fault-tolerance), 분할능력(partitionability), 확장성(scalability) 등이 고려되어야 한다.

그럼에도 불구하고 이 분야의 연구가 활발한 것은 시스템을 구성하는 비용이 저렴하면서도 계산 능력이 뛰어난 파급효과 때문일 것이다. 이 분야에서의 대표적인 연구과제로는 대규모 병렬처리 프로세서(MPP: Massively Parallel Processors)[14] 및 분산 공유메모리 시스템(distributed shared memory system)[17]에 관한 연구를 들 수 있다. 이러한 시스템들은 아직 개발단계이므로 성공여부는 확신할 수 없으나, 향후 병렬처리 컴퓨터의 연구방향과 기술혁신의 주축을 이루어 나갈 것임에는 틀림이 없다.

3.3.1 확장성(scalability)

고정된 네트워크에 연결된 MIMD에서 프로세서의 수를 늘리는 것은 네트워크의 확장성과 밀접한 관계가 있다. 예를 들어, 하이퍼큐브의 경우에는 프로세서의 수를 배수로 늘일 수 있으나 각 프로세서의 포트의 수가 하나씩 늘어나야 하므로 기존의 시스템을 확장하는 것은 어려운 문제이다. 메쉬의 경우에는 프로세서를 늘리어도 프로세서당 포트의 수는 일정하나 늘려나가는 프로세서의 수는 네트워크의 크기에 의존적이다.

임의의 프로세서 수를 허용하는 네트워크도 있으나 링을 제외하고는 실용적이지 못하고, 링의 경우에는 직경이 커서 대규모 시스템의 네트워크로는 적합하지 않다. 이러한 문제를 극복하기 위한 방법으로 다단계로 시스템을 구성하는 방안이 연구되고 있는데, 확장가능한 네트워크(SCI: Scalable Coherence Interface)을 사용하거나 계층구조의 클러스터(cluster) 네트워크 등에 관한 연구[11,13,18]가 그것이다. 또한 확장성이 향상되면서 프로세서간 통신시간의 지연을 방지하기 위한 분산 공유메모리 시스템에 관한 연구도 활발히 수행되고 있다.

클러스터 구조는 서로 상이한 방식의 네트워크를 계층으로 구성하는 방식이다. 따라서 많은 프로세서를 단일 네트워크로 연결하는 시스템에 비하여 네트워크의 부하를 분산시킬 수 있으므로 많은 프로세서를 가지는 시스템의 구성이 용이하다. 예를 들어, 대표적인 클러스터 시스템인 Cedar[11]는 8개의 프로세서로 구성되는 밀결합 MIMD를 하나의 클러스터로 구성하고 이들 클러스터를 오메가 네트워크(Omega network)을 통하여 공유메모리 구조로 연결하고 있다. 계층구조에 알맞은 연산의 형태는 클러스터내의 프로세서간 통신속도가 클러스터간 통신속도에 비하여 빠르므로 클러스터내의 통신유발 빈도가 크나 클러스터간 통신 횟수가 작은 연산의 형태가 적합하다.

분산 공유메모리 구조는 Stanford Dash[20]에서 발견된다. 많은 프로세서를 네트워크로 연결하면 네트워크의 부하가 증가하게 되고 결국 이것이 프로세서간 통신을 지연시켜 전체 시스템의 성능을 저하시키므로 프로세서의 수가 증가하더라도 네트워크의 부하를 증가시키지 않는 방안이 강구되어야 한다. 이러한 방안의 하나로 각 프로세서에게 개별메모리를 가지도록 하고, 네트워크의 통신부하를 최소로 하는 공유메모리 일관성 유지기법이 연구되었다. 즉, 공유메모리 일관성 문제[10]는 기존의 분산메모리를 가상의 공유메모리(virtual shared memory)로 간주하고 각 프로세서가 보유하고 있는 복사본을 효율적으로 처리하는 방법이다. 확장가능한 네트워크(SCI)[18]은 가상의 공유 메모리를 네트워크에서 지원하기 위하여 IEEE가 표준으로 채택한 네트워크로 프로세서 노드와 외부 네트워크와의 접속을 정의하고 있다. 프로세서 노드들은 SCI를 통하여 링 또는 크로스바 스위치 등과 같은 다양한 네트워크로 연결되어 클러스터를 구성할 수 있으며, 이들은 다시 외부의 네트워크에 연결하여 여러가지 대규모 네트워크을 구성할 수 있다.

3.3.2 대규모 프로세서를 위한 네트워크

대규모의 프로세서로 시스템을 구축하기 위해서는 이들을 상호연결하는 네트워크의 구조가 단순하고 확장하기 쉬워야 하며, 빠른 통신을 지원하며, 분산 공유메모리를 채용하고, 집적방식이 개선되어야 한다. 최근에는 이러한 필요성을 인식하여 확장성이 높은 일차원 링, 이차원 배열, 삼차원 배열이나 토러스(torus)메쉬구조 등이 많은 연구의 대상이 되고 있다. 이차원 메쉬를 이용한 시스템으로는 Stanford Dash[20], MIT Alewife[21], Intel Paragon[22], Caltech Mosaic[23], Wisconsin Multicube[24] 등이 있으며, 삼차원 메쉬로는 MIT J-Machie[25], Tera computer[26], Cray T3D[27] 등이 있다. KSR-1[28]은 계층구조를 기본 네트워크로 삼고 있다.

대규모 프로세서로 구성된 시스템에서 각 프로세서별로 상이한 프로그램을 마련하는 방법은 매우 번거러운 절차이다. 따라서 대규모 프로세서로 구성된 시스템에서는 벡터계산과 같이 단순한 작업이 반복되는 SIMD형의 계산분야에 알맞다. 그러나 SIMD형의 연산형태는 연산입자의 크기를 작게 설정하므로, 프로세서간 통신의 유발가능성이 커서 대규모 프로세서 시스템에서는 통신지연 시간이 급격히 증가할 우려가 있다.

따라서 통신지연 시간이 작은 시스템을 구성해야 한다. 실제로 MIT의 J-Machine[25]은 통신지연 시간이 2 μsec로 MIMD(Intel iPSC/1은 1msec 수준임)에 비하여 크게 단축되어 미소연산입자(fine-grain) 연산을 지원할 수 있다. Seitz[23]는 향후 병렬처리 컴퓨터의 발전 방향을 대규모의 확장가능하며, 미소연산입자 모델을 수용할 수 있는 컴퓨터일 것으로 진단하고 있다.

3.4 데이타 플로우(dataflow)와 리덕션 (reduction) 컴퓨터

MIMD나 SIMD에서는 작업의 수행순서가 프로그램에 명시된 제어순서를 따른다. 이러한 전통적인 방법을 이용하여 병렬처리를 수행하기 위해서는 프로그램상에 작업의 흐름을 표현해 주어야 한다. 그런데 프로그램으로부터 또는 수행하려는 어떤 작업으로 부터 병렬처리가 가능한 부분을 모두 찾아내고 병렬처리효과를 극대화하는 방안을 마련하는 것은 매우 어려운 일이다. 예를 들어 MIMD에서 메모리 참조시의 충돌을 회피하기 위하여 메모리 참조 순서를 프로그램 작성자가 일일이 확인하거나 각 프로세서간의 통신이 불규칙하게 일어나는 경우, 이를 정확하게 예측하고 이를 토대로 프로그램을 작성하는 것은 거의 불가능한 일이다.

이러한 문제를 해결하기 위한 대안으로 프로그램이나 작업내에 묵시적으로 포함된 병렬성을 컴퓨터가 스스로 발굴하여 병렬성을 최대한 유지하면서 프로그램을 수행하도록 하는 함수형 계산모델이 제안되었다. 또한, 함수형 계산모델은 프로그램의 수행에 따른 명령어 코드의 변환 여부에 따라 데이타 플로우 모델(dataflow model)과 리덕션 모델(reduction model)로 세분된다.

3.4.1 데이타 플로우 모델

데이타 플로우 모델[28]은 연산의 실행순서를 연산에 필요한 자료의 존재여부로 결정하는 데이타구동(data driven) 구조이다. 즉, 데이타의 흐름이 실행의 순서를 좌우하기 때문에 프로그램에는 동기화를 위한 표현이 불필요하고, 프로그램상의 모든 묵시적 병렬성이 모두 이용될 수 있다. 또한 실행의 결과가 실행 순서나 속도에 관계없이 항상 일정하기 때문에 고도의 병렬성과 정확성을 얻을 수 있는 장점도 있다. 데이타 플로우 컴퓨터를 개발하는 입장에서는 프로그램에 명시된 병렬성으로 시스템 자원을 이용할 수 있도록 하여야 하며, 또한 데이타의 존재여부를 연산의 실행순서로 치환하여야 하므로 von Neumann 방식의 컴퓨터 제작기술을 이용하는 경우에는 연산의 동기화를 이루게하는 메카니즘이 복잡해진다. MIT에서 개발한 시스템[28]과 일본의 ETL Sigma-1[29]에서는 명령어마다 필요한 입력자료가 준비되었음을 의미하는 토큰이 도달하기를 기다리다가 도달하면 해당되는 명령어를 수행하고 그 결과로 생성되는 자료에 표시한 토큰(tagged token)을 발생하여 다음 명령어에게 제공한다. 이 과정에서 다음에 수행할 명령어를 하드웨어로 지정해주는 방식이 정적 데이타 플로우 방식이며, 수행되어야 하는 명령어를 지정해서 토큰을 부여하는 방식이 동적 데이타 플로우 방식이다.

3.4.2 리덕션 컴퓨터

리덕션 모델[32]은 데이타플로우 모델과 반대의 성격을 가지고 있다. 즉, 어떤 함수는 그 결과를 요구하는 경우에만 불리워지는 요구 구동(demand driven) 방식으로 처리된다. 만일 요구된 어떤 함수가 실행시 입력자료가 필요하나 필요한 입력자료가 마련되어 있지 않은 경우에는 입력자료를 생성하는 함수를 호출한다. 이러한 과정을 반복하면 최종적으로, 필요한 자료가 준비된 함수부터 계산이 수행되고 계산결과를 함수가 불리었던 역순으로 되돌려지면 이 값을 필요로 하는 함수가 수행된다.

리덕션 모델을 구현할 때, 한번의 함수호출로 되돌려진 결과를 여러 함수에서 복사하여 사용하는 방식(그래프 리덕션 방식)과 매번 해당 결과를 필요로 하는 함수를 호출하여 실행하는 방법(스트링 리덕션 방식)이 있다. 후자의 방식은 많은 프로세서들이 계산에 참여할 가능성이 많아 병렬성이 높으나, 중복 계산으로 인한 자원의 낭비 및 지연시간의 발생을 초래할 수 있다. 전자의 경우에는 모든 수식을 그래프로 표현하여

중복된 계산을 배제할 수 있으므로 후자의 단점을 보완할 수 있다.

리덕션 수행모델은 프로그램상의 모든 단위연산명령을 병렬처리 단위로 하는 경우에는 병렬처리 단위간의 잦은 인자 전달로 인하여 통신부하가 급격히 증가된다. 이를 줄이기 위해서는 순차처리되는 부분을 통합하여 병렬처리 단위를 크게 하는 것이 바람직하다. 이러한 관점에서 리덕션 모델은 데이타 플로우 모델과는 달리 MIMD 연산 모델에서와 같이 중형 연산입자 모델이 보다 나은 성능을 발휘할 것이다.

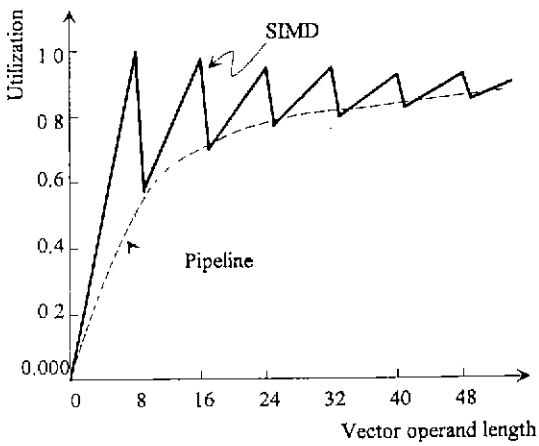


그림 4 벡터길이에 따른 벡터 프로세서와 배열 프로세서의 성능비교

4. 병렬처리 메카니즘

4.1 프로그램 분할과 스케쥴링

병렬처리는 한 작업을 여러개의 작업으로 나누어 서로 협력하면서 실행속도를 증가하는데 목적이 있다. 따라서 어떤 작업을 병렬처리 컴퓨터에서 수행하고자 할 때, 병렬로 수행할 작업들을 어떻게 분리할 것인가와 분할된 작업들을 연산소자나 프로세서에 어떤 방법으로 할당할 것인가를 고려하여야 한다. 이러한 문제를 각각 프로그램의 분할(partition)과 할당(allocation) 또는 스케쥴링(scheduling)이라고 한다. 분할과 할당은 독립적인 문제가 아니라 상호보완적이다. 그리고 하드웨어와 매우 밀접한 관계가 있다. 본고에서는 하드웨어와 관계없이 병렬성을 향상시키는 기법에 대한 사항은 다루지 않는다. 이에 대한 자료는 참고문헌[30]을 참조하라.

4.1.1 연산입자의 길이와 크기

프로그램을 분할하여 병렬 처리가 가능한 병렬 타스크의 크기를 연산입자라고 할 때, 연산입자의 크기는 벡터처리에서와 같이 단위연산을 하나의 연산단위로 처리하는 미소연산입자(fine grain)로부터 서부루틴 단위에 이르는 대단위 입자(coarse-grain)에 이르기까지 다양하다. 파이프라인을 이용하는 벡터처리는 단위연산을 반복해서 처리하는 루프를 벡터처리의 기본단위로 삼는다. 슈퍼벡터기능을 지닌 벡터 프로세서에서는 몇개의 단위연산을 순차적으로 처리하는

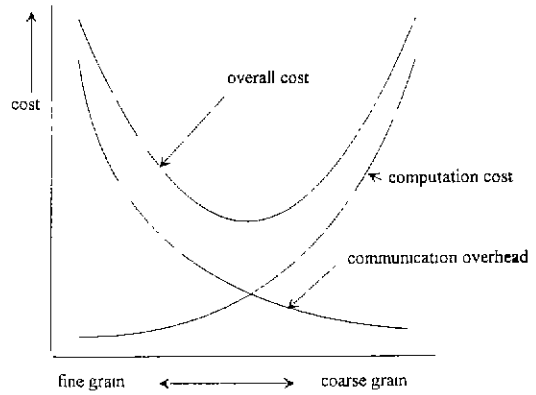


그림 5 연산입자 크기에 따른 병렬성과 통신비용 관계

문장을 반복하는 프로그램이 적합하다.

배열 프로세서에서는 모든 연산소자가 동시에 하나의 작업을 수행하도록 함으로써 효율을 높일 수 있으므로 루프내의 모든 기본 블럭(basic block)을 하나의 연산단위로 간주할 수 있으므로 벡터 프로세서에 비해서 연산입자의 크기가 상대적으로 크다. 그러나 배열 프로세서는 한번에 처리하는 연산의 수가 연산소자의 수로 결정되므로 긴 벡터는 연산소자의 수로 분할하여 여러차례 반복해서 작업을 수행하여야 한다. 따라서 처리할 벡터길이가 연산소자수의 정수배가 아니면 시스템의 효율이 낮아진다. 그림 4에서 벡터 프로세서는 벡터길이에 비례해서 속도가 꾸준히

높아지나, 배열 프로세서에서는 톱니형태로 나타난다.

MIMD에서는 연산입자간의 자료종속관계가 프로세서간의 통신으로 치환되므로 SIMD와 같이 입자크기를 작게하는 경우에는 프로세서간의 통신에 많은 비용이 소요될 것이다. 이와 반대로 연산입자를 크게하면 통신요구는 줄어드나 병렬처리 가능성이 줄어들어 병렬처리의 효과가 감소한다. 이러한 문제를 보완하기 위하여 두가지 연산 모델의 중간을 따르는 중형입자모델이 제안되었다[7,31]. 그림 5는 연산입자의 크기에 따른 프로그램의 병렬성과 잠재적 통신 비용의 관계를 나타낸 것이다.

그림 5에서 보는 바와 같이 입자의 크기가 작을수록 병렬처리의 가능성은 커지나, 통신 및 스케쥴링 비용도 높아진다. 따라서 MIMD에서는 연산입자의 크기와 통신비용을 조화시킴으로써 컴퓨터의 성능을 향상시킬 수 있다.

4.1.2 연산입자의 통합(packing)

병렬처리를 위한 프로그램을 마련하기 위해서는 주어진 하드웨어에 적합한 최적의 연산입자 크기를 결정하여야 한다. 본 절에서는 Kruatrachue[31]가 사용하였던 예제를 중심으로 입자의 크기가 병렬처리에 미치는 영향을 살펴보기로 하자. 그림 6(a)와 같이 17개의 연산을 수행해야 하는 프로그램을 두개의 프로세서를 가진 MIMD시스템에서 수행시킨다고 하자. 프로그램의 자료종속관계는 그림 6(b)와 같다. 여기서 프로그램 1~6까지는 각각 1단위시간이 필요한 연산이며, 7~17까지는 2단위시간이 소요되는 연산작업이라고 가정하고, 각 연산간의 자료종속관계를 해소하기 위하여 필요한 통신시간이 그림 6(b)의 화살표에 기록된 단위시간으로 가정하자. 이때 프로그램의 각 문장을 하나의 연산입자로 간주하는 미소연산입자 모델에서는 그림 6(c)와 같이 42 단위시간이 소요된다. 통신시간이 차지하는 비중을 줄이기 위해 통신비용을 증가시키는 작은 입자들을 모아서 하나의 커다란 입자로 성장시킨 중형 연산 입자 모델에 의한 실행결과는 그림 7(b)와 같이 38 단위시간이 소요되므로 후자의 방법이 전자의 방법에 비하여 유리함을 알 수 있다.후자의 방법은 그림 6(a)의 프로그램을 그림 7(a)와 같이 다섯개의 중형입자로 통합한

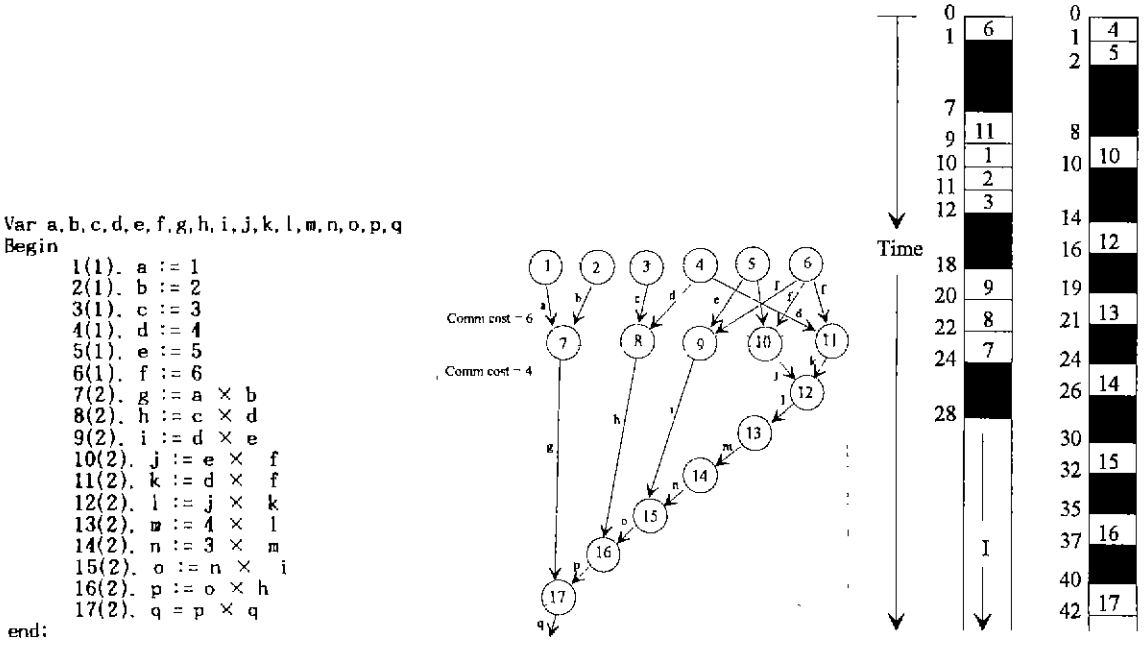


그림 6 예제 프로그램과 자료종속그래프

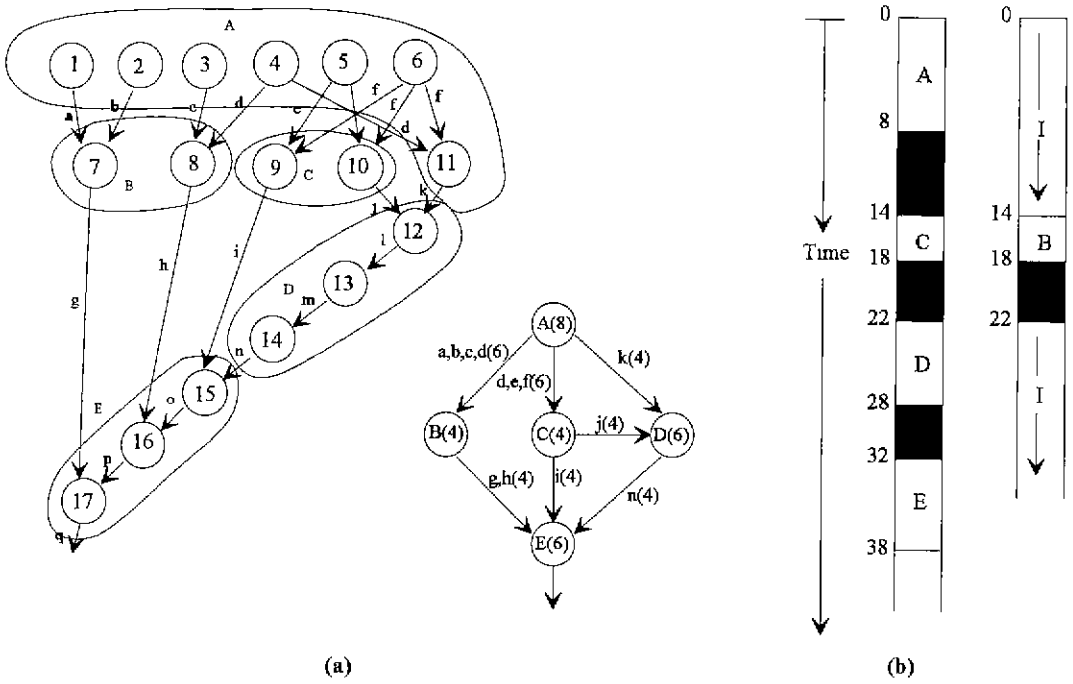


그림 7 중형연산입자 모델에 의한 병렬처리 결과

것이다. 그림 7에서 원안의 숫자는 통합된 프로그램의 문장번호를 의미한다. 또한 통합후의 통신비용은 화살표위에 표시하였다.

4.1.3 프로세서 스케쥴링

미소연산입자를 중형연산입자로 통합함으로 인하여 병렬처리 효과를 높일 수 있음을 그림 7로부터 알 수 있다. 또한 통합된 중형연산입자를 실제로 프로세서에 할당하는 과정에서도 통신에 따른 실행시간의 지연을 방지할 수 있다. MIMD에서 최적의 결과를 가져올 수 있는 프로세서 할당 방법은 NP문제임이 밝혀진 바 있으나, 경험적인(heuristic) 방법으로 준최적인 여러가지 프로세서 할당방법이 제안되었다. 그 중에서 Kruatrachue[31]의 방법은 할당시 통신으로 인하여 실행시간이 지연되는 경우 연산입자를 여러 프로세서에서 중복해서 실행시켜 시간을 줄일 수 있다면 하나의 연산입자를 다수의 프로세서에서 중복실행시키는 방법이다. 예를 들어

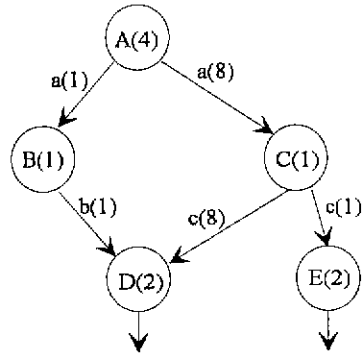


그림 8 예로 든 연산입자간의 자료종속성

주어진 연산입자간의 자료종속성이 그림 8과 같을 때, 기존의 방법으로는 그림 9(a)와 같이 할당되어 23 단위시간이 필요하나, 그림 9(b)와 같이 두개의 프로세서에서 동일한 연산입자를 중복실행하도록 함으로써 10단위시간에 모든 작업을 완료하는 프로세서 할당이 가능함을 알 수 있다.

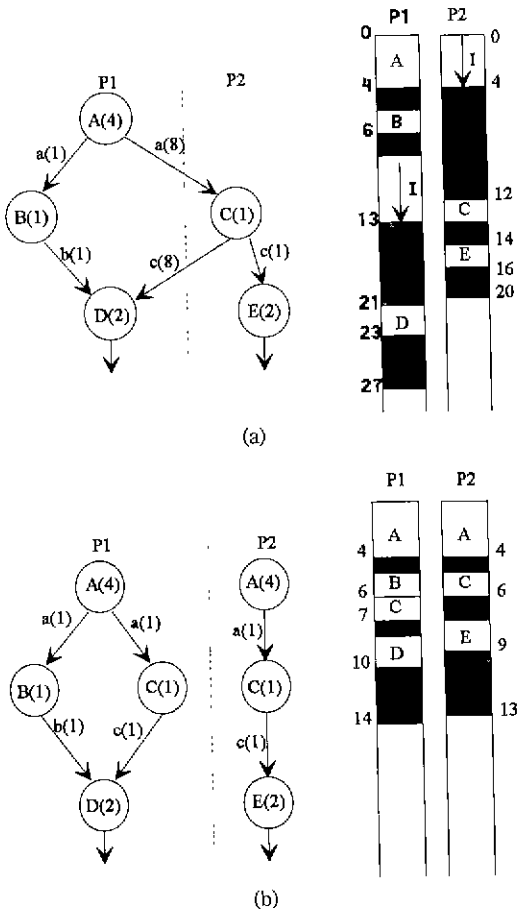


그림 9 Kruatrachue의 방법에 따른 할당

4.2 구동방식별 처리방법

프로그램에 명시된 순서에 따라 순차적으로 처리되는 제어 구동(control-driven)방식은 프로그래머가 지정한 제어 순서(control flow)에 의하여 처리된다. 이것은 단일 프로세서 뿐 아니라 대규모의 프로세서를 연결한 MIMD나 대규모 병렬처리 컴퓨터에서도 마찬가지이다. 이러한 컴퓨터에서는 여러개의 프로세서가 서로 다른 작업을 수행하므로 각 프로세서간의 작업 순서를 지정해주어야 한다.

참고문헌 [33]에 수록된 예와 같이 수식 $\alpha=(b+1)*(b-c)$를 MIMD에서 수행한다고 할 때, 공유메모리 MIMD에서는 그림 10(a)와 같이 프로세서 동기를 제어하는 fork-join기능을 이용하여

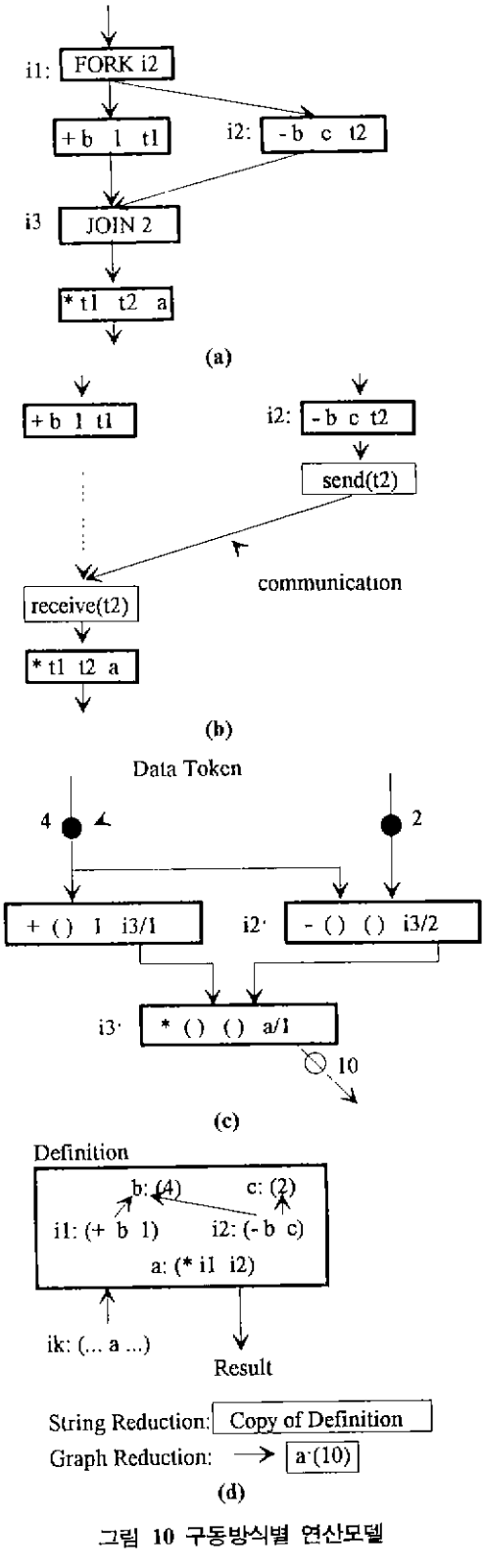


그림 10 구동방식별 연산모델

2개의 프로세서를 이용하는 프로그램의 작성이 가능하다. 분산 메모리 MIMD에서는 그림 10(b)와 같이 프로세서간 통신을 수행하는 통신 명령을 삽입하여 병렬처리 프로그램을 작성한다. 공공유메모리 MIMD와의 차이는 분산메모리 MIMD에서는 필요한 변수의 값이 직접 프로세서에게 전달되나 공유 메모리 MIMD에서는 프로그램의 수행 순서를 제어하는 제어명령어가 각 프로세서에서 수행되는 점이 다르다.

데이타 플로우 컴퓨터와 같은 데이타 구동형 구조에서는 메모리를 공유하지 않으므로 메모리를 공유함에 따른 부작용(side effect)이 없으며, 자료 이용가능 여부에 따라 연산이 수행되므로 명령어 수준의 미소연산입자 모델에 의한 병렬처리가 가능하다. 동일한 예제에 대하여 적용하면, 데이타 구동형 연산을 (b+1)과 (b−c)를 먼저 수행되고 나서 곱셈이 수행된다. 이러한 연산은 피연산자들을 필요로 할 때마다 계산된 값을 인출하여 연산을 계속 수행하는 eager evaluation에 의한다. 그림 10(c)는 데이타 구동형 연산과정을 보여준다.

요구 구동(demand-driven) 방식으로 연산을 수행하는 리덕션 컴퓨터에서는 top-down방식으로 프로그램이 실행된다. 동일한 예제를 리덕션 컴퓨터에서 수행하는 경우, 우선 α를 요구하면, 이것은 곧 곱셈을 위한 피연산자 (b+1)과 (b−c)를 요구한다. 그런데 (b+1)은 덧셈을 위한 피연산자 b와 1을 필요로 하며, 동시에 (b−c)는 뺄셈을 요구하고 피연산자 b와 c를 필요로 한다. 이 시점에서 b, c 및 1이 준비되어 있으므로 덧셈과 뺄셈이 동시에 수행된다. 이 작업이 종료되면 그 결과가 곱셈의 입력자료로 되돌려져 곱셈이 수행되고 그 결과가 α값으로 되돌려진다. 따라서 리덕션 모델에서는 연산이 불리워질 때마다 연산이 수행되는 lazy evaluation방식으로 처리된다. 그림 10(d)는 요구 구동형 연산과정을 보여준다.

5. 결 론

병렬 처리의 궁극적인 목표는 여러가지 컴퓨터 자원의 이용을 극대화 함으로써 고속 연산을 가능하게 하는 것이다. 본 고에서는 여러가지 병렬처리 컴퓨터 구조와 병렬처리의 개념을 살펴보았다. 병렬처리의 효과를 극대화하려면 시스템에 포함된 프로세서를 놀리지 않고 동시에 연산에 참여하도록 하는 프로그램을 마련하는 것이며, 다른 하나는 프로그램 수행 과정중에 일어날 수 있는 여러가지 비용을 최소화 하는 것이다.

파이프라인 프로세서를 효과적으로 이용하기 위해서는 프로그램 중에 조건문이나 분기 인스트럭션이 발생되는 횟수를 가급적 줄이고 벡터 연산의 길이를 늘려야한다. 배열 프로세서의 경우에는 경쟁적 메모리 참조의 가능성을 줄이는 프로그램이 되어야 한다. MIMD 컴퓨터에서는 프로세서간에 유발되는 통신 횟수가 가급적 적으면서 병렬성이 큰 프로그램을 마련하여야 효과가 크다.

확장성이 있는 컴퓨터에서도 통신시의 시간지연을 줄이기 위하여 하드웨어 및 소프트웨어의 도움을 받으려는 연구가 활발하다. 아직은 보다 나은 시스템을 개발하려는 노력이 광범위하게 진행중이므로 곧 상업성 있는 시스템들이 개발될 것이다.

데이타 플로우 컴퓨터나 리덕션 컴퓨터는 개념적으로 병렬성을 처리할 수 있는 장점이 있으나 아직도 실용화에는 많은 난관이 놓여있다. 그러나 앞으로 큰 발전이 이룩될 수 있을 것이다. 앞으로 개발되는 병렬처리 컴퓨터는 보다 많은 프로세서를 포함하는 대규모 시스템이 될 것이며, 이에 따라 다양한 병렬처리기법이 제안될 것이다.

복잡하고 많은 프로세서를 채용한 컴퓨터를 효과적으로 이용하는 프로그램을 작성하는 것도 매우 복잡하고 어려운 문제이다. 예전에는 미소연산입자 모델이나 큰 연산입자 모델의 한가지 모델을 따르는 일차원 병렬처리 기법으로 충분히 실용적인 병렬처리 프로그램을 작성할 수 있었으나, 앞으로 개발되는 병렬처리 컴퓨터등에서는 여러가지 병렬처리 기법이 혼합된 방법이 요구될 것이다. 다행히도 이러한 병렬처리 컴퓨터를 위한 병렬화 또는 벡터화 컴파일러가 계속 개발중에 있으므로 이들 컴퓨터에 적합한 병렬처리 프

로그램을 작성하는 절차도 간단해질 것이다.

참고문헌

[1] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Trans. Comput.*, C-**21**(9): 948~960, 1972.

[2] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, 1984.

[3] S-Y. Kung, K. S. Arun, R. Gal-Ezer and D. V. Bhaskar, "Wavefront array processor: language, architecture and applications," *IEEE Transactions on Computers*, C-**31**(11): 1054~1065, November, 1985.

[4] 박명순, 이진호, 양정일, 박성순, "VLIW 시스템을 위한 컴파일러 최적화 기법," 한국정보과학회지 제 12권 5호, 1994년 6월.

[5] G. H. Barnes, "The ILLIAC-IV computer," *IEEE Trans. Comput.*, C-17: 746~757, 1968.

[6] R. S. Valga, *Matrix Iterative Analysis*, Prentice-Hall, 1962.

[7] S. Kim, *The Implementation of a Parallel Computer for Loosely Coupled Multiprocessing Systems*, PhD thesis, North Carolina State University, 1989.

[8] 김석일, 김대식, "슈퍼 컴퓨터의 연산 모델과 자원할당", 병렬처리시스템 연구회지 제 1권 2호, pp. 34~50, 1990. 6

[9] S. Fortune and J. Wyllie, "Parallelism in Random Access Machines." *Proc. ACM Symp. Theory of Computing*, pp. 114~118, 1978

[10] 박병섭, 김성천, "병렬처리 시스템에서 캐쉬일관성에 대한 고찰" 한국정보과학회지 제 12권 5호, 1994년 6월.

[11] D. J. Kuck, E. S. Davidson, D. H. Lawrie, and A. H. Sameh, "Parallel Supercomputing Today-The Cedar Approach," *Science*, **231**(2), Feb. 1986.

[12] D. J. Kuck and R. A. Stokes, "The Burroughs Scientific Processor(BSP)," *IEEE Trans. Computers*, pp. 363~376, May 1982.

[13] 김기철, "상용 대규모 병렬처리 컴퓨터의 시스템 구조," 한국정보과학회지 제 12권 5호, 1994년 6월.

[14] K. E. Batcher, "Design of a Massively Parallel Processor," *IEEE Trans. Computers*, pp. 836~840, Sept. 1980.

[15] TMC, *The CM-2 Techical Summary*, Thinking Machines Corporation, Cambridge, MA, 1990.

[16] P.Christy. "Software to Support Massively Parallel Computing on the MasPar MP-1," *Digest of Papers Spring Compcon 90*, San Francisco, CA, Feb. 1990.

[17] A. Agrawal, R. Simoni, J. Hennessy, and M. Horowitz, "An Evaluation of Directory Schemes for Cache Coherence," *Proc. 15th Annu. Int. Symp Computer Arch.*, 1988.

[18] D. James, A.T. Laundrie, S. Gjessing, and G.S. Sohni, "Scalable Coherence Interface," *IEEE Computer*, **23**(6): 74~77, 1990.

[19] KSR, *KSR-1 Overview*, Internal Report, Kendall Square Research Coporation, 170 Tracer Lane, Waltham, MA 02154, 1991.

[20] D. Lenoski, J. Laudon, K. Gharachorloo, W.D. Weber. A. Gupta, J. Hennessy, M. Horowitz, and M. Lam, "The Standford Dash Multiprocessors," *IEEE Computer*, pp. 63~79, Mar. 1992

[21] A. Agarwal, D. Chaiken, G. D'Souza, K. Johnson, D. Kranz, J. Kubiatowicz, K. Kurihara, B. H. Lim, G. Maa, D. Nussbaum, M. Parkin, and D.A. Yeung, "The MIT Alewife Machine: A Large-Scale Distributed Memory Multiprocessor," *Proc. Workshop Multithreaded Computers, Supercomputing* 91, 1991.

[22] Intel, Paragon *XP/S Product Overview*, Supercomputer Systems Division, Intel Corporation, Beaverton, OR, USA, 1991.

[23] C. L. Seitz, *Mosaic C: An Experimental Fine-Grain Multicomputer*, Technical Report, California, Institute of Technology, Pasadena, CA, 1992.

[24] J. R. Goodman and P. Woest, "The Wisconsin Multicube: A New Large-Scale Cache-Coherent Multiprocessor," *Proc. 15th Annu. Int. Symp. Computer Arch.*, pp. 422~431, 1988.

[25] W. J. Dally, J. Fiske, J. Keen, R. Lethin, M. Noakes, P. Nuth, R. Davison, and G. Fyler, "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms," *IEEE Micro*, **12**(2): 23-39, Apr. 1992.

[26] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith, "The Computer System," *Proc. ACM Int. Conf. Supercomputing*, pp. 1~6, Amsterdam, The Netherlands, June 1990.

[27] Cray, *Cray/MPP Announcement*, Cray Research, Inc., Eagan, MN, 1992.

[28] Arvind and R. S. Nikhil, "Executing a Program on the MIT Tagged-Token Dataflow Architecture," *IEEE Trans. Computers*, **39**(3): 300~318, 1990.

[29] K. Hiraki, K. Nishida, S. Sekiguchi, T. Shimada, and T. Yiba, "The SIGMA-1 Dataflow Supercomputer: A Challenge for New Generation Supercomputing Systems," *Journ. Info. Processing*, **10**(4): 219~26, 1987.

[30] 이만호, "병렬화를 위한 루프구조의 변환," 한국정보과학회지 제 12권 5호, 1994년 6월.

[31] B. Kruatratue and T. Lewis, "Grain Size Determination for Parallel Processing," *IEEE Software*, **5**(1): 23~31, Jan. 1988.

[32] 김형식, 김명주, 전주식, "리덕션 컴퓨터의 소개," 병렬처리시스템 연구회지 제 1권 1호, pp. 4~17, 1993년 3월.

[33] G. S. Almasi, A. Gottlieb, *Highly Parallel Computing*, Benjamin/Cummings Pub., CA. USA., 1994.

김 석 일

1975 서울대학교 전기공학과 학사
1975 ~1990 국방과학연구소 선임연구원
1985 ~1989 North Carolina Satate University 공학박사
1990 ~현재 충북대학교 컴퓨터과학과 교수
1993 ~현재 국방과학연구소 위촉연구원
관심 분야 : 병렬처리 컴퓨터구조, 슈퍼컴퓨팅, 병렬처리언어

이 충 한

1994 충북대학교 전자계산학과 학사
1994 ~현재 충북대학교 전자계산학과 석사과정
관심 분야 : 병렬처리 컴퓨터구조, 병렬처리 알고리즘

지 승 현

1993 충북대학교 전자계산학과 학사
1993 ~현재 충북대학교 전자계산학과 석사과정
관심 분야 : 병렬처리 컴퓨터구조, 병렬처리 알고리즘, 운영체제론

김 대 식

1984 충북대학교 자연과학대학 계산통계학과 학사
1997 숭실대학교 공과대학 전자계산학과 석사
1987 ~현재 국방과학연구소 연구원
관심 분야 : 병렬처리 컴퓨터구조 및 운영체제, 병렬처리 컴파일러, 이미지 프로세싱, 인공지능