特輯

대단위 병렬 처리 컴퓨터구조에 관한 고찰

金新德, 表三洙 *
光云大學校 컴퓨터工學科,
* 三星 電子情報 컴퓨터 本部

Ⅰ. 서론

컴퓨터 발전의 역사는 지속적인 성능 증대를 위한 노력과 연구에 대한 결과로서 컴퓨터 성능의 변천사로 설명 되어질수 있다. 그리고 컴퓨터의 성능 증대를 위하여 전형적인 방법으로써 고속 처리기 (processor)의 설계와 주변 부품에 대한 고속 동작 기술의 고안에 주력해 왔으며 지난 20년간 이를 위한 기본 기술 (technology)은 꾸준히 발전해 왔다. 그러나 컴퓨터 처리 능력의 증대에 대한 지속적인 요구가 기술의 발전 속도를 앞질러 왔으며 이제 거의 모든 처리 속도 (computing speed)의 물리적 한계에 도달 하고 있다. 즉 기존 고성능 컴퓨터들의 처리 속도가 거의 한계에 도달하고 있는 반면 과학과 공학 분야에서는 문제의 복잡성 증대로 컴퓨터의 보다 높은 처리 능력이 요구 되고 있다.

컴퓨터 구조란 컴퓨터 시스템내에 구성 되어진 각 동작 모듈의 구조로서 설명 되어지며 고성능을 이루기위한 효과적인 설계목표는 전체적으로 시스템 내부 각 모듈의 처리 능력들이 조화를 이루도록 (balanced system) 설계하는 것이다. 그리고 특정 모듈의 처리 능력 부족으로 초래되는 병목 부분 (unbalanced point)에 대한 궁극적인 두가지 해결 방법으로써, 그 모듈 부위에 대하여 고속 부품기술 개발 및 그 적용기술을 설계 하거나, 혹은 경제적인 방법으로써 해당 모듈을 여러개 사용하는 방법으로 병렬성을 이용하는 것이며, 이 두가지 결론적 해결책은 현재까지 지속적으로 사용되는 방법들이다. 즉 위의 두가지 방법에 의거 성능 증대를 위하여 사용되어온 예는 기본적인 IC 기술의 진보와 응용 기술, 컴파일러 기술의 최적화 기술, 멀티 프로그램밍, 스풀링 (spooling), pipelining, cache system, 및 low-level 병렬 처리 방법으로써의 superscalar, superpipelining, VLIW (very long instruction word)와 국부적 병렬성의 이용예로서 RAID (redundant array of inexpensive disks), 시스템 수준에서는 다중 처리 시스템과 병렬 처리 시스템을 들 수 있다.

컴퓨터의 성능은 크게 두가지 관점에서 설명 되어질수 있고 또 이 두가지 이유에서 시스템의 설계방식과 분류 배경이 쉽게 설명되어질 수 있다. 컴퓨터의 선으은 사용자 측면과 시스템 관리자 측면으로 나뉘어 지는데, 사용자는 주어진 업무 (job)의 처리속도 (execution time) 혹은 응답 속도 (response time)에 관심이 있고, 시스템 관리자는 주어진 단위 시간내에 시스템이 얼마나 많은 업무 (jobs)를 수행했는가, 즉 업무 처리량(throughput)에 더 많은 관심이 있다. 이러한 두가지 현상을 압축해 보면, 성능이란 주어진 업무의 처리 속도 그리고 업무들의 처리량으로 설명 되어진다. 결론적으로 컴퓨터의 성능을 증대시키기위해 두가지 다른 방법을 취할 수 있다는 것이며 그 첫번째로, 처리 속도 증대 (speedup)를 위한 목적으로 병렬 처리 (parallel processing) 방식이 사용되어져 왔고 이를 위한 구조로써 병렬 처리 컴퓨터들이 고안 사용 되어왔다. 이들 시스템들은 구조적으로 처리기들의 수에 물리적인 제약을 가지고있지 않은것이 특징이어서 시스템 확장이 용이하고 시스템 설계도 비교적 간단하며 대단위 병렬 처리 시스템이라고 불리운다. 두번째로 업무처리량의 증대를 위한 방법에서도 병렬성을 이용하게 되었으며 기존 소프트

웨어 베이스를 그대로 사용 가능케 하면서 다수 사용자(multi-user)의 지원으로 여러개의 업무 (jobs)를 효율적으로 처리할수 있도록 촛점이 맞추어져 왔다. 그리고 병렬성이 제공되어지는 잇점을 최대로 이용하기위해 단위 업무 처리 시간 의 단축 (speedup)에도 활용될수 있도록 구성 되어지고 한개의 공유된 메모리 주소 영역을 갖도록 구성 되어지며 일반적으로 다중 처리 (multi processor) 시스템 이라고 불리어 왔다. 그러나 이들 시스템 모두를 포함하여 병렬 처리 시스템이라고 한다.

병렬성을 이용하는 근본적인 이유는 기본적으로 우주 그 자체가 병렬성을 내포하고 있으므로 그 응용분야 또한 무궁무진 하기 때문이다. 그리고 보다 원칙적인 배경은 주어진 싯점에서의 활용 가능한 최신의 기술 (technology)로 얻을수 있는 컴퓨터의 성능은 그 기술을 바탕으로 얻을수 있는 최대의 성능이나, 역으로 그자체가 성능 증대의 한계점임을 역설한다고 할수 있겠다. 그러므로 이러한 기술적 제한을 극복하는 유일한 대안으로써 병렬 처리 시스템을 들수있다. 즉 주어진 기술적 바탕에서 다수의 처리기를 사용 함으로써 요구하는 계산 처리 능력을 구현할수 있는 유일하고 경제적인 방법이다. 시대적인 기술의 발전 (technological advance)을 최대로 활용 하면서 그 자체의 한계를 간단히 극복할수 있는 방법임을 주시해야 한다. 그리고 이러한 시스템들은 그 자체가 가장 간단한 에러 허용 (fault tolerant) 시스템들이다. 병렬 처리 방식이 오랜 역사에도 불구하고 현재 광범위하게 통용 되고 있지 못한 근본적인 이유는 병렬성의 이용이 구체적 병렬성 (explicit parallelism)과 함축적 병렬성 (implicit parallelism)으로 구분하고 있음이 그 구체적인 배경이다. 즉 사용자가 사용하기 편리 하도록 하기위한 응용 프로그램의 자동 병렬화의 미비에서 기인하나 강력한 컴파이러의 다각적인 연구에 의한 실용화가 실현됨으로써 광범위하게 사용될 것이다.

결과적으로 병렬 처리 컴퓨터들이 최고의 수퍼 컴퓨터로서 강력히 요구되어 오고있다. 이러한 병렬 처리 컴퓨터들은 수십에서 수천의 비교적 간단하고 값싼 처리기들로 구성 되어있다. 대단위 병렬 처리 컴퓨터들은 이론적으로 대단히 높은 속도를 가능하게 하나, 이를 실제적으로 실현 하는것은 어떻게 성공적으로 수백 수천의 처리기들을 효과적으로 통신하며 함께 동작 시키느냐에 달려있다. 본 기고문에서는 크게 두가지 성능 요인에 의거하여 간단하게 작업 처리 및 처리 속도 증대를 위한 다중 처리 시스템의 구조적 변화와 기술적 현황이 설명 되어지고, 많은 부분이 속도 증대를 위한 병렬 처리 시스템들의 구조적 특징과 발전 현황에 대해 보다 자세하게 기술되어진다. 그러나 병렬 처리 시스템은 위 두가지 계열의 시스템을 포함하므로 2장에서는 일반적인 구분 방법에 의거 이들 시스템을 분류하고, 작업처리 시스템 계열이 3 장에서 설명 되어지며, 4 장은 대단위 병렬 처리 시스템 계열이 다루어지고, 마지막으로 5 장에서 본 기고문의 요약과 병렬 처리 분야에 대한 전망에 대해 기술 되어진다.

Ⅱ. 병렬 처리 시스템의 분류

병렬성 (parallelism)은 광범위한 문제들의 처리 시간을 줄이기 위한 한 방법이며 여기에는 데이타에 의한 병렬성 (data parallelism)의 이용과 작업 처리 기능에 의한 병열성 (function parallelism)의 이용이 가능하다. [14] 병렬 처리 컴퓨터들은 시스템의 크기 (즉 처리기의 수)와 그 구성 방법 및 처리 방식에 따라 다양하게 분류되어 질수 있다. 즉 메모리 구성 방법 (즉 shared, nonshared, hierachical 등), 처리기들의 연결 방법 (즉 bus, mesh, hypercube, multistage cube 등) 등이 있으나, 보다 보편적인 방법은 Flynn [9] 이 제안한 방법으로서, 명령어와 데이타에 대한 제어 흐름에 의거하여, 처리기들이 다수의 데이타 (multiple data)에 대해 한개의 명령어 (single instruction)를 동기적으로 수행 하는가 혹은 다수의 데이타에 대해 다수의 명령어들 (multiple instruction)을 비 동기적으로 수행 하는가로 구분한다.

SIMD (Single Instruction - Multiple Data stream) [9] 구조의 병렬 처리 방식은 데이타의 병렬성 (data parallelism)을 이용하며 모든 처리기들은 주워진 데이타 (data set)의 각기 다른 부분 (element)들에 대하여 같은 연산을 수행 한다. SIMD 시스템에서는 처리기들의 제어를 위한 한개의 제어기 (CU: control unit)가 선택된 처리기들에게

한개의 명령어를 동기적으로 전송한다. 각 처리기는 자신의 데이타에 대하여 전송된 명령어를 lockstep으로 수행 한다. 현재 구성 되어진 SIMD 시스템들의 예는 CLIP4 [10], Thinking Machine사의 CM series [12, 19], DAP [13], IBM사의 GF11 [5], Illiac IV [6], MasPar [7], MPP [4] 와 STARAN [3] 등을 들수 있다. 그리고 SIMD 계열의 시스템들은 대부분 분산 메모리 구조로 구성 되어지며 수 십에서 수 만개의 처리기를 구성 할수있는 대단위 병렬 처리 시스템으로 분류 되어진다.

MIMD (Multiple Instruction - Multiple Data stream) [9] 구조의 병렬 처리 방식은 데이타의 병렬성은 물론 작업 처리 기능의 병렬성 (function parallelism)을 이용할 수 있으며 작업 처리 기능에 의한 병렬성의 이용시 각 처리기는 전체 업무 (task)를 수행 하기 위하여 서로 다른 부분 업무 (function or subtask)를 수행한다. 그러나 일반적으로 한개의 업무 (task)는 서로 독립적인 부분 업무들로 균등하게 나누어질 수 없으므로 처리기들은 마지막 처리기의 처리가 끝날 때까지 기다리게 된다. 그러므로 MIMD 시스템의 수행 시간은 각 처리기들 중에서 주어진 부분 업무를 가장 늦게 처리 하는 처리기의 수행 시간이 된다. 현재 구성 되어진 대형 MIMD 시스템의 예는 BBN Butterfly [8], Cm* [15], IBM RP3 [17], Intel iPSC [1] 와 i860xp를 사용한 Paragon, NCube series [11] 와 Ultracomputer [20], Thinking Machine사의 CM-5 [24], Kendall Square Research의 KSR1 [23] 등을 들수있다.

MIMD 시스템 계열은 처리기들간에 상호 통신을 어떻게 하는가에 따라 공유 메모리 (shared memory) 시스템과 메세지 전송 (message passing) 시스템으로 분류 할수 있다. 공유 메모리 시스템 계열은 한개의 공유된 메모리 주소 영역이 존재하며 이는 기존 작업 처리 시스템의 기본 요구를 충족 시켜주어 범용 시스템으로 분류할 수 있다. 또한 주어진 단위 업무에 대해서도 병렬 처리화 할수 있어서 다중 업무 처리는 물론 단위 업무 처리 속도 증대의 두가지 다른 영역을 제공해주는 시스템 계열이다. 1 장에서 설명 되어진 다중 처리 시스템은 이 계열에 속한다. 이에 반하여 메세지 전송 시스템 계열은 주로 분산 메모리 시스템에서 운용되는 방식으로 단위 업무 처리 속도의 증대가 주 목표이며 이계열의 시스템들은 위에서 지적한대로 대단위 병렬 처리 시스템으로 분류되어질수 있다. 그리고 SIMD 계열의 시스템들도 주로 분산 메모리 구조를 이루어 대단위 병렬 처리 시스템으로 분류 되어진다.

또한 SIMD와 MIMD 방식들의 장점을 따온 혼합형 시스템 (mixed-mode system)들도 한 부류로 자리잡고 있으며 이들 시스템들은 SIMD 혹은 MIMD mode를 수행하며 양 수행 mode 간에 명령어 수준 단위 (instruction level granularity)에서 SIMD/MIMD 수행 mode 변환이 가능하다. 혼합형 시스템의 예는 OPSILA [2], PASM [18], TRAC [16] 을 들수있다. 이외에도 VLIW (Very Long Instruction Word) 방식이나 data driven 형태의 data-flow방식, demand driven 형태의 reduction 모델 등을 포함하여 기타 많은 시스템 설계들이 몇몇 다른 시스템의 구조적 특징의 혼합형으로써 병렬 처리 컴퓨터의 구조는 다양하게 발전 되어가고 있다. 다음 장에서는 첫번째 지적한 다중 처리 시스템들의 현황과 구조적 특징에 대해서 보다 자세하게 다루어진다.

Ⅲ. 다중 업무 처리용 병렬 시스템

다중 처리 시스템 혹은 보다 포괄적으로 공유 메모리 MIMD 시스템이라 불리우는 이 분야는 기존 소프트웨어 베이스상에서 상향 호환성 (upward compatible)을 보이고 시스템 성능 향상의 두가지 목표를 제공하여 주기때문에 많은 컴퓨터 제조회사에서 다양한 컴퓨터를 설계 하여왔고 상용성을 인정 받아왔다. 이들 시스템들의 기본적인 요건으로 한개의 공유 메모리 주소를 제공하여 기존의 업무 처리 환경인 다수 사용자 (multi-user)와 다중 업무 처리 (multi-processing)은 물론 단위 업무 처리 속도 증대를 위한 병렬성을 제공하는 것이다. 이러한 공유 메모리형 MIMD 시스템들은 사용 되어지는 연결 구조 (interconnection network)의 형태에 따라 세분 되어질 수 있다. 공유 메모리 MIMD 시스템 구성에서 가장 간단한 처리기들의 연결 방식으로 한개의 공유 버스 (single bus-based)에 의한 구성 방식을 들수 있으며, 이들 시스템은 초기 다중 업무 처리용의 대표적인 시스템들로써 최대 수십개의 처리기를 한개의 공

유 버스 (common bus)에 연결하여 사용하는 형태로 사용 버스의 고정된 통신 대역폭 (bandwidth) 때문에 사용 가능한 처리기의 수에 물리적인 제약이 존재 한다. 이 부류에 속하는 예는 Encore의 Multimax, Sequent사의 Balance/Symmetry, Flexible사의 Flex32, Alliant Computer의 FX/8, Cray Y-MP, 그리고 SUN사의 각종 Server system등이 있다. 이들 시스템들은 공통적으로 버스의 사용을 줄이기 위해 처리기에 국부적 캐쉬 (local cache)를 사용하며 캐쉬 일관성 (cache coherency)를 유지 하기 위해 버스 스누핑 (snooping) 방법을 사용한다. 이들 시스템들의 근본적인 문제는 처리기의 사용수에 근본적인 제한이 존재한다는 점이다. 즉 시스템의 자원 (resource)으로서 한개의 공유 버스의 사용은 버스의 제한된 통신폭 (bandwidth)로 말미암아 기본적으로 시스템 확장을 제한하게 된다. 처리기의 수가 10개 정도의 수준에서는 시스템 성능 증대를 위한 두가지 목표를 성공적으로 이루었으나 확장성의 근본적 문제를 해결할수 없는 구조이다. Multiple bus-based 시스템들의 예는 Encore사의 Ultramax, CMU (Carnegie Mellon university)의 Cm* 등이 있으며 그외 많은 공유 메모리 MIMD 시스템들이 다단 연결 구조 (multistage interconnec-tion network)를 사용하고 있다.

1980년대의 다단 연결 구조의 공유 메모리형 시스템의 예로써 BBN Butterfly [8] 시스템은 Omega network의 환형 (wraparound) 형태로 구성되며 총 256 개의 processing node까지 구성 할수 있다. 각 node는 Motorola의 MC68020 처리기를 사용하며 4 Mbytes의 메모리, Processor Node Controller (PNC), 그리고 I/O 버스와 Omega network로 구성 되어 진다. 구조상으로 메모리는 각 processor node에 분산 되어 있으나 이들은 한개의 총체적인 address space를 이룬다. 특이점 으로써 Butterfly 시스템은 Chrysalis 오퍼레이팅 시스템에 의해 운영되며 두가지 프로그램밍 환경, 즉 cooperating sequential processes (메세지 전송 방식)과 Uniform system (Monitors 사용에 의거한 공유 메모리형 프로그램밍 모델)이 제공되어 진다.

또 다른 예로써 Ultracomputer [20] 는 fetch-and-add (F&A), synchronization primitive, 그리고 같은 메모리 위치에 대한 2개 이상의 처리 요구 (requests)를 조합 처리 할수 있는 진보된 interconnection network (enhanced Omega network)의 두가지 설계상의 특징을 가지고 있다. Ultracomputer는 N-개의 처리기, N-개의 메모리 모듈, 그리고 Omega network으로 구성 되어 진다. 각 처리 node 내에는 memory latency를 줄이기 위해 cache가 사용 되며 network상의 switch node들은 메세지들을 저장할 queue와 메세지들을 조합 결합할수 있는 처리 능력이 갖추어져 있다. 이러한 메세지들의 조합 처리 방법은 network traffic과 메모리 hot spot 문제들을 효과적으로 감소시켜 준다. IBM사의 RP3 [17] 는 단적으로 Ultracomputer의 super-set design으로 볼수 있다. 그러나 BBN Butterfly나 Ultracomputer등은 단일 주소 공간의 분산 메모리 구조로 주소 참조의 지역성 (locality of reference)도 이용하지 아니하며 메모리 내용의 일관성도 구체적으로 프로그램내에서 다루어져야하는 문제로 다소 비 효율적이다.

실질적으로 성능 증대의 두가지 기본 목표를 만족시키면서 확장성 (scalability)을 보장하는 시스템의 구조를 설계 가능하다면 상용과 과학 연산등의 특정 목적을 수행할수 있는 가장 이상적인 시스템이 될수 있으며 고성능 시스템 설계의 궁극적 목표가 될수 있다고 말할수 있겠다. 여기서 확장성이라 함은 사용 처리기의 수를 증대 시킬때 시스템의 성능도 거의 선형적 (near linear)으로 증대되는 시스템을 말한다. 그러나 기본적으로 자원이 공유 된다는것 자체가 이의 실현을 어렵게 만드는 역설적인 제약점이기 때문에 간단히 해결될 문제가 아닐 것이다. 그리고 현재까지 이러한 의미에서 확장성을 고려한 공유 메모리 시스템의 구조는 그리 성공적이라 할수 없다. 시스템의 특성과 시대적 의미를 연관 시키는 것은 각 분야에서의 성능 증대를 이루기 위한 새로운 시스템의 구조적 진보를 시대적으로 구분하여 단계적인 발전을 총체적으로 설명 할수 있다. 이러한 배경에서 다중 업무 처리용 병렬 시스템 분야는 크게 1980년 대의 10개 정도의 처리기를 사용하는 소형 규모 (small scale) 시스템으로 특징 지을수 있으며 1990년대인 현재 수 백개의 처리기를 사용하는 중형 규모 (mid-scale) 시스템으로 분류되어 진다. 소형 규모 시스템들은 위에서 언급이 되었고 이들 시스템들의 근본적인 문제인 시스템 공유 버스의 통신폭 확장을 위한

다른 구조적 특징을 갖는 최근의 중형 규모 시스템을 살펴 보기로 하자. 이 분야는 상용 시스템으로써 광범위하게 사용되어질수 있기 때문에 확장성이 보장되는 구조의 시스템 설계는 상당한 파급 효과가 있다.

최근의 중형 규모의 다중 처리 시스템 설계의 예로는 학계에서 고안된 MIT의 Alewife[22], Stanford의 DASH[21] 등이 있으며, 상용 시스템으로는 Encore의 GigaMax, Kedall Square Research의 KSR1[23] 등이 있다. 이들 시스템중 DASH와 KSR1에 대해서만 간단히 구조적 특징을 살펴 보겠다. DASH 시스템은 Directory Architecture for SHared memory의 약어로 각각의 node는 기존의 단일 공유 버스와 스누핑을 사용하는 다중 처리 시스템을 사용하며 상위 레벨에서 이들 각각의 node를 그림 1 에서와 같이 Caltech Mesh로 연결하여 전체적인 시스템을 구성한다. 이는 단일 공유 주소 영역을 갖는 분산 메모리 구조를 의미하며 캐쉬 일관성 문제를 해결하기위해 분산 디렉토리 프로토콜 (distributed directory protocol)을 사용한다. 이 시스템 구성의 특징은 기존 단일 공유 버스 다중 처리 시스템을 node로 구성하여 상위 레벨에서의 각 node을 연결할 수 있는 설계의 간편성을 이룬 시스템의 예라 하겠다.

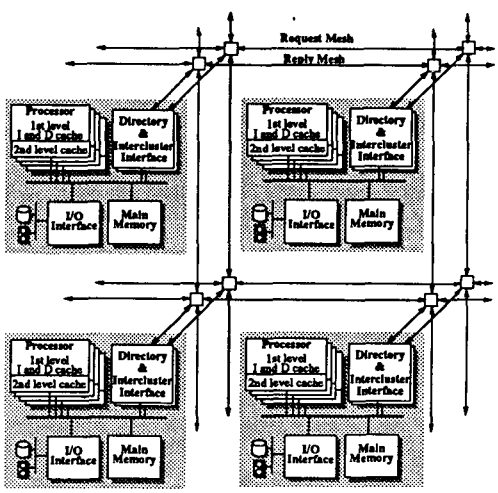

그림 1. DASH 시스템의 구조

Kendall Square Research사의 KSR1 은 병렬 처리 능력은 물론 다중 업무 처리 능력을 보유하면서 확장성까지 제공하는 시대적 의미를 부여 할수 있는 시스템으로 평가하고 싶다. 이 시스템의 구조는 자체 특허를 보유한 AllCache 메모리를 사용함으로써 컴퓨팅 파워에서의 확장성을 허용한다. AllCache 메모리는 기존의 순차적 공유 메모리 프로그래밍 모델을 확장성 병렬 처리 시스템에 적용 가능하게 해주는 구조이다. 그림 2 에서와 같이 각각의 프로세싱 node는 64-bit superscalar RISC 처리기를 사용하며 처리기들은 국부적으로 ring 구조로 연결 되어 있고 이러한 ring으로 연결 되어진 처리기 그룹들은 계층적으로 상위 ring구조에 의해 연결 되어진다. 그림 2는 2-level 계층 구조로 연결된 시스템의 예이며 첫번째 level에서 32개, 두번째 level에서 1066개의 처리기를 연결한다.

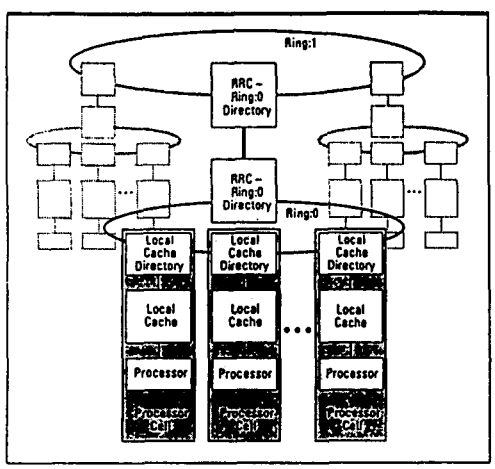

그림 2. KSR1의 구성도

Ⅳ. 대단위 병렬 처리 시스템

1. 시스템의 구조적 특징

시스템 이용의 주 활용 목적이 단위 업무의 처리 속도 증대 (speedup)로써 대부분의 병렬 처리 시스템의 포괄적 영역이 이에 속한다. 일반적으로 시스템 확장성이 용이 하여 응용 범위에 따라 시스템을 확장 구성할수 있으며 처리기의 수가 증가함에 따라 그 성능 (speedup)도 선형적으로 증가한다. 시스템의 각 모듈, 즉 처리기들, 메모리, I/O, 및 내부 연결 구조

(interconnection network)와 통신 대역폭 (communication bandwidth)등 모든 부위에서 확장성 (scalability)이 보장 되어지는 구조를 갖고 있다. 또한 이들 대단위 병렬 처리 시스템들의 또 하나의 특징은 효율적인 가격 대 성능 비를 보여준다. 이에 속하는 시스템 계열은 앞장에서 지적한 바와 같이 크게 데이타 병렬성의 이용 및 작업 처리 기능에 대한 병렬성을 이용 하는 두가지 방식으로 구분되어 구조적으로 SIMD 시스템과 비공유 (분산) 메모리 MIMD 시스템으로 각각 분류되어 질수 있다. 일반적으로 시스템 메모리 구성상, 비공유 메모리 시스템들은 처리기들간의 상호 통신 방법이 메세지 전송에 의하여 이루어 지므로 구분 없이 사용 하겠다.

MIMD 구조는 데이타의 병렬성과 업무 처리 기능의 병렬성을 모두 이용할수 있으므로 보다 일반적인 응용 분야에 사용될수 있는 특성을 가진 반면 SIMD 구조들은 데이타의 병렬성 만을 이용하여 주로 과학 연산과 같은 특별한 응용 분야에 사용되어 왔다. 일반적으로 프로그램의 유연도 (flexibility)나 적용성 (applicability) 면에서 MIMD 구조는 동시 수행하는 병렬 연산에 대하여 운용상 거의 제약이 없는 반면 SIMD 구조는 적용 알고리즘에 따라 병렬 처리상의 제약이 있으며 그 성능도 수행 알고리즘에 의해 큰 영향을 받는다. SIMD 시스템의 동작은 근본적으로 동기 처리 (synchronous processing) 방식이라고 표현할 수 있으며 MIMD 시스템의 동작은 비동기 처리 (asynchronous processing) 방식으로 표현할 수 있다.

운용상의 특징으로 SIMD mode의 병렬성은 선택된 한개의 프로그램에 대한 명령어들을 모든 처리기에 전송 수행함을 특징으로 하며 각 처리기들은 자신의 데이터 (local data)에 대해 전송 되어진 명령어를 수행한다. 또한 한개의 제어 흐름 (control thread)이 존재하기 때문에 오직 한개의 프로그램이 제어기 (CU)내에 저장되어 있으며 제어기는 프로그램 수행 제어 (control flow) 명령어들의 자체 수행과 병렬 데이타 처리 명령어들을 각 처리기에 전송 수행하게 한다. 이에비해 MIMD mode의 병렬성은 각각의 처리기가 자신의 내부 데이터에 대하여 개개의 프로그램 수행을 특징으로 하며 여러개의 제어 흐름 (control thread)들이 존재하여 각 처리기들은 서로 비 동기적으로 동작 한다.이러한 구조적 특징에 대하여 몇가지 구조상의 장단점을 비교 하면 다음과 같다.

시스템 설계의 복잡도 측면에서 볼때 SIMD 구조의 설계는 MIMD 구조 보다 훨씬 간단하다. MIMD 시스템의 업무 처리의 병렬성 이용은 수행되어질 프로그램이 몇몇 부분 업무 (subtask)들로 나뉘어져 처리기들에 의해 처리 되어야 하므로 어떻게 각 프로세스 (process 혹은, 한개의 처리기에 할당된 개개의 subtask)를 생성/종료 (create/terminate) 해야 하는가, 어떻게 프로세스들간에 통신을 하고 동기를 해야 하는가, 등 하드에어 및 소프트웨어 상의 모든 면에서 시스템 구성에 대한 복잡성을 내포 하고 있다.

메모리의 효율성 면에서 볼때 SIMD 구조에서는 전체 시스템에 대해 하나의 공통 명령어 저장 메모리가 제어기 (CU) 내에 존재한다. 반면에 MIMD 구조에서는 프로그램의 전부 혹은 일부가 처리기의 자체 메모리 (local memory, 비공유 메모리형 시스템의 경우) 혹은 자체 국부적 캐쉬 (local cache, 공유 메모리형 시스템의 경우)에 존재함으로 SIMD 구조가 메모리 공간 사용 면에서 더 효율적이다. 처리기들의 구성 문제에서는 SIMD 구조에서는 MIMD 구조에서 필요치 아니한 제어기 사용 (CU cost)가 요구 되어지나 일반적으로 SIMD 명령어들은 제어기에 의해 해석 (decode) 되고 그 제어 신호 (control signal)들이 처리기들에게 전송 되므로 SIMD 구조의 처리기들은 명령어 해독 (instruction decode)을 위한 별도의 하드에어가 필요하지 않다.

SIMD 구조와 MIMD 구조간의 또 하나의 구분은 동기 부담 (overhead)를 들수 있다. SIMD 구조에서는 프로그램 수행시 동기는 명령어 수행 자체에 이미 내포 되어 있으므로 별도의 동기 수단이 필요치 않으나 MIMD 구조에서는 처리기들간에 프로그램 수행을 위한 동기가 요구 되어지며 별도의 동기 수단이 제공 되어야 한다. 특히 처리기들간의 통신이 필요 할때 SIMD의 내재된 동기 (implicit synchronization)의 장점은 더욱 명백해 진다. 이 경우 SIMD 구조의 각 처리기들은 한개의 프로그램을 병렬로 동기적 처리를 하기 때문에 각 처리기들은 어느 처리기로 부터 메세지를 받았는지 이 메세지가 무엇을 의미 하는지 알고 있기 때문에 MIMD 구조의 처리기들간의 데이터 전송시 필요한 동기 수단이나 각 처리기들간의 상호 인식 방법 (identification pro-

tocol)이 필요 없으므로 데이터 전송 속도도 빠르다. 그러므로 MIMD 구조에서는 동기 부담 비용 (synchronization overhead cost)이 요구 되어지며 이러한 부담 (overhead)은 MIMD 수행 방식의 프로그램밍시 유연성 (flexibility)의 보상으로 나타나게 된다.

마지막으로 프로그램 개발시의 문제점을 들수 있다. 기본적으로 MIMD 구조의 작업 처리 기능의 병렬성은 일반 사용자가 최적의 구조를 갖는 프로그램 개발이 어렵고 더나아가 intelligent compiler에 의해 병렬성이 구현 되어 진다고 하더라도 프로그램 오류 수정 (debugging)시 일반 사용자 에게는 쉽게 이해 또는 처리되기 어렵다. 이는 MIMD 수행 방식에서 처리기들이 서로 상관성이 있는 프로그램 혹은 프로그램의 일부를 독립적이고 비동기적으로 수행 하기 때문에 한개의 오류를 발견 하기 위해서 각 처리기 (혹은 process)의 개별 프로그램에 대한 검사가 요구되어지며 수행처리 되는 처리기들의 수에 따라 그 복잡성은 급격히 증대되기 때문이다. 이에 반해 SIMD 수행 방식에서의 프로그램 오류 수정은 MIMD에 비해 훨씬 용이하다. 그 이유는 모든 처리기들이 한개의 프로그램을 동기적으로 수행 함으로써 기존 순차적 프로그램에 대한 오류 수정을 하는 것과 유사 하기 때문이다.

이러한 기본적인 구조적 특징을 배경으로 다음 절에서는 SIMD, MIMD 시스템들의 발전 추이와 대표적인 시스템의 구조를 시스템 동작 운용 관점에서 간단히 살펴본다.

2. SIMD 구조

SIMD 구조는 전형적으로 N 개의 처리기들과 한개의 제어기 (CU), 그리고 interconnection network로 구성 되어진다. 일반적인 SIMD 구조는 PE-to-PE 구성으로 한개의 PE (processing element)는 처리기와 자체 메모리로 구성 되어지며 일반적으로 one-bit 처리기들로 구성되어 있다. 이러한 SIMD 시스템들은 2-dimensional array 배치상에 bit-serial 처리기를 사용하는 구조적 특성으로 화상 처리 (image processing)나 화상 인식 (pattern recognition)등의 응용 분야에 널리 사용 되어져 왔다.

일반적으로 제어기는 수행 제어 명령어 (예를 들면 loop controls, branches 등과 같은 scalar 연산)를 수행하며 데이터 처리 명령어를 처리기들에게 전송하고 모든 선택된 (active) PE들은 각기 자체의 데이터에 대해 동시에 전송 명령어를 수행한다. 그러므로 이 모델에서는 오직 한개의 프로그램이 존재하며 이 프로그램의 일부 (즉 scalar 연산)는 제어기에 의해 수행되고 그 나머지 부분 (병렬 연산)은 병렬 처리기들에 의해 수행된다. 이러한 한줄기의 명령어적 (single instruction stream) 특성 때문에 기존의 Fortran이나 C 컴파일러들에 대해 부분적 수정 보완으로 SIMD의 병렬성을 이용할수 있다. SIMD 시스템들의 제어기는 PE 명령어의 전송은 물론 모든 scalar 연산을 할수 있도록, 혹은 제한된 scalar 연산 능력을 갖도록 설계 되어 있다. 후자의 경우에는 FE (Front End) 시스템을 사용하여 SIMD 프로그램의 시행과 대부분의 scalar 연산을 수행 하도록 되어 있다.

SIMD 시스템들의 발전사는 표 1과 같으며 대표적인 시스템의 예들이 시대적으로 간략히 그 구조와 운용상의 특징에 의거 아래과 같이 설명 되어진다.

표 1. SIMD Machines

연도	시스템 명칭	규 모	제작상태
1972	Illiac IV	8×8	prototype
	STARAN	256×1	〃
1976	DAP	32×32	〃
1980	CLIP4	96×96	〃
	DAP	64×64	commercial
1983	MPP	128×128	〃
1985	CM-1/2	65536	〃
	GF-11	24×24	〃
1990	MasPar	16384	〃

1972년 병렬 처리 시스템의 hardware-first 방식으로써 256 개의 PE로 구성된 Illiac IV [6]는 미국 NASA에서 1983년 Goodyear사로 부터 MPP를 들여올 때까지 사용된 전형적인 SIMD 시스템이다. Illiac IV의 주요 구성은 CU, PEs, I/O subsystem과 B6700 FE 컴퓨터이며 한개의 CU가 64-PE arrays를 제어하여 한 그룹을 이루며 총 4개의 그룹으로 256 개의 PE를 조합 구성 한다. MPP (Massively Parallel Processor) [4] 시스템은 Array Control Unit (ACU), PE Array Unit (ARU), Program

and Data Management Unit (PDMU), 그리고 interconnection network로 구성 되어 진다. ARU는 128x128 기본 PE array들과 128x4의 예비 PE array로 구성 되어 진다. DAP (Distributed Array of Processor)[13] 시스템은 Master Control Unit (MCU), Host Connection Unit (HCU), PE array와 interconnection network으로 구성 되어 있다. DAP 500(600) 시스템은 32x32 (64x64)의 one-bit processor array로 총 1,024 (4,096) 개의 PE로 구성 된다. HCU는 DAP 시스템과 호스트 컴퓨터 (즉 FE 시스템)간의 연결을 제공 하며 FE 시스템은 프로그램의 개발, loading, initiating, 그리고 DAP 프로그램의 제어를 위해 사용 된다.

Connection Machine (CM-1, CM-2)은 Thinking Machine사에서 개발된 SIMD machine[12, 19] 이다. CM은 65,536개의 실제 (physical) PE array로 구성 되어 있으며 가상 처리기 (virtual processor) 이용을 위한 구조가 제공되어 실제보다 많은 가상 PE의 사용을 가능하게 한다. 시스템의 구성은 시스템 소프트웨어의 개발과 동작 환경을 제공해 주는 FE 컴퓨터와 데이터 병렬 처리 연산을 위한 65,536 개의 PE, 제어기의 역활을 하는 4 개의 sequencer, 병렬 입출력 시스템과 interconnection network로 이루어 진다. 또한 복잡한 부동 소수점 연산이 요구되는 과학 계산 (scientific computing) 응용 분야를 위해 CM-2의 각 PE는 one-bit 처리와 부동 소수점 처리가 가능한 혼합 형태를 이룬다. CM은 CU를 채용하지 않고 그대신 FE 컴퓨터가 전체 시스템의 제어를 맡고 있고 모든 프로그램들은 FE 컴퓨터에 저장되며 모든 CM 프로그램의 수행은 FE 시스템에 의해 제어 되어 진다. Single data에 관련된 명령어 (scalar code)는 FE 컴퓨터에서 수행 되어지며 parallel data array에 관련된 명령어 (parallel code)는 CM에서의 수행을 위해 전송 되어 진다. Scalar data는 FE 컴퓨터에 저장 되며 parallel data는 FE 컴퓨터상에서 I/O command에 의거 PE memory에 분산 처리 된다. 결과적으로 CM은 FE 컴퓨터의 instruction set과 I/O 처리 능력을 확장 하는 방식을 취하는 구조 이다. FE 컴퓨터가 CM에게 제공하는 명령어는 Paris 라고 불리우며 Paris 명령어들은 CM의 sequencer에 의해 처리 되어 진다. Sequencer는 각각의 Paris instruction을 일련의 초보적인 연산 (nanoinstructions)으로 변환하여 PE들에게 전송 수행하게 한다. 즉 CM의 sequencer는 Paris 명령어의 해석 및 전송을 맡고 있다.

MasPar의 MP-1은 시스템을 구성 하는 PE들의 수에 대해 확장이 용이한 (scalable) 구조를 갖고 있으며 기존 SIMD 시스템들과는 달리 최적의 컴파일러 기술을 제공하는 RISC (Reduced Instruction Set Computer)형의 명령어 세트 (instruction set) 설계에 기초 하고 있다. MasPar의 MP-1 계열은 MP1100과 MP1200으로 MP1100은 4,096 개의 PE, MP1200은 16,384 개의 PE로 구성되어 있으며 각 처리기는 4-bit RISC 처리기를 사용 하고 있다. MasPar 시스템의 구성은 그림 3 에서와 같이 FE, Array Control Unit (ACU), 그리고 PE array의 3 부분으로 나누어 진다. FE 시스템은 MP-1에 대한 운영과 network 환경을 제공해 준다. ACU는 FE 와 PE array간의 상호 작용을 제어하며 scalar 연산을 수행 하여 FE 컴퓨터와 함께 PE array를 제어 한다. MP-1 시스템은 FE 명령어와 ACU 명령어 줄기 (instruction stream)와 같이 2 개의 명령어 줄기를 갖는다. 모든 PE 명령어들은 ACU 메모리에 저장되어 위의 두 명령어 줄기에 의거 2 개의 기본적인 프로그래밍 모델 (synchronous model과 asynchronous model)이 제공 되어 진다.

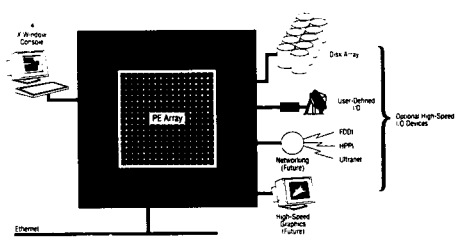

그림 3. MasPar 시스템의 구성도

Synchronous model에서는 FE 시스템과 ACU/PE array의 어느 한 부분이 시간적으로 어느 한 시점에 선택 동작 된다. 예로 UNIX의 remote procedure call과 유사한 한개의 subroutine call은 FE 시스템과 ACU 간의 직접적인 control flow transfer를 허용 한다. 그러나 asynchronous model에서는 FE와 ACU가 함께 동작 될수 있다.

또한 MP-1 compiler는 2 block의 code, 즉 FE 컴퓨터에 대한 code와 ACU 를 위한 code를 만들어 낸다. ACU는 PE array 와 FE/PE 상호 동작 그리고 I/O subsystem을 control 한다.

3. MIMD 구조

MIMD 시스템들의 설계시 고려 되어야할 요인들은 프로세스의 생성과 종료 방법(process creation/termination mechanism), 처리기들 간의 상호 통신 (interprocessor communication) 과 동기 방법(syncronization mechanism)등을 들수 있다. 이러한 MIMD 시스템은 메모리 구성 방법에 따라 공유 메모리형 시스템 (shared memory system)과 비공유 메모리형 시스템 (noshared memory system)으로 구분되며 시스템의 자원을 처리하는 방식 (즉 상호 통신 방법)에 따라 공유 메모리형 방식(shared memory)과 메세지 전송 방식 (message passing)으로 구분 할수 있다. 공유 메모리형 시스템에 대해서는 2 장에서 다중 처리 시스템으로서 설명되었으므로, 본 장에서는 대단위 병렬 처리 시스템과 연계하여 비공유 메모리 구조에 입각한 메세지 전송 방식의 MIMD 시스템이 다루어진다. 메세지 전송 방식은 공유 메모리형 시스템에서의 사용이 가능 하나 (예: BBN Butterfly와 RP3) 주로 비공유 메모리형 시스템에 적용 사용되며 (예: NCube series, iPSC series등), 동기적 처리와 비동기적 처리 방식으로 구분 되며 처리기들간에 엄격한 독립성이 존재한다. 공유 메모리형 방식은 전체적으로 공유된 한개의 주소공간 (physical address space)을 제공 하며 사용자적인 측면에서 볼때 보다 병렬 처리화가 용이하고 프로그램밍시 거의 제약이 없으나 시스템 설계의 측면에서 memory contention과 cache coherency (local cache 사용시) 등 보다 많은 어려운 문제점을 내포 하고 있다. 결과적으로 병렬 처리화의 어려움(burden)을 공유 메모리형 방식은 시스템 설계자에게, 메세지 전송 방식은 사용자에게 부가 하게 된다.

비공유 메모리형 MIMD 시스템들은 주로 메세지 전송 방식을 사용 하므로 비교적 구성이 간단 하고 각 처리기들의 제어가 용이 하나 운영상의 제약이 따른다. 대표적인 시스템으로는 Intel사의 iPSC hyercube [1] 와 NCube사의 NCube-2/10 [11] 을 들 수 있다. 비공유 메모리형 시스템의 가장 중요한 특징은 시스템 node의 확장성을 들수 있다. 즉 비공유 메모리형 시스템의 확장시 시스템에 연결되는 처리기들의 수에는 근본적으로 제약이 없고 확장시의 문제점도 적으며 데이터 병렬성의 이용에도 광범위하게 사용될 수 있다.

Intel사의 iPSC 시스템은 32, 64, 혹은 128 개의 Intel 80286 microprocessor들이 hypercube network에 의해 연결 되어지며 각 node는 host 시스템인 cube manager에 연결 되어 있다. 각 처리 node들간의 data 전송은 메세지 전송 방식에 의해서만 이루어질 수 있다. Hypercube의 연결 방식의 장점으로는 시스템 확장이 용이하며 (cube구조의 dimension 확장에 의해 간단하게 증가 시킬수 있음) 많은 병렬 응용 알고리즘들이 hypercube network에 쉽게 mapping이 될수 있다. 응용 프로그램은 cube manager에서 개발, 컴파일 되어 각 node에 down load되는 방식을 취한다. 응용 프로그램 개발시, 적용 알고리즘은 독립적으로 운영될 프로세스들의 구분을 명백히 하고 상호 작용 (send/receive)을 정확 하게 정의해 주어야 한다. Intel iPSC는 80386을 사용한 iPSC/2 [1] 로 발전 되었고 최근에는 i860xp 처리기를 기반으로 하는 Paragon이 발표 되었다. Intel Paragon 시스템은 메세지 전송을위한 메세지 전용 처리기를 사용하여 전송 지연을 줄이며 2-D Mesh 내부 연결 구조를 사용 각 node에 MRC (Message Routing Controller)를 설치하고 연결 지점에 각 처리기를 연결한다. 각 node의 구성은 기존의 iPSC series와 유사하다.

NCube사의 NCube/10 시스템은 Cosmic cube구조를 기반으로한 MIMD 시스템으로 16 개 로부터 1,024 개의 node 까지 확장이 가능하다. 각 node는 독립적인 32 bit 처리기, 자체 메모리, 그리고 각 node간의 interconnection network로 구성 된다. Processor node간의 통신은 전적으로 메세지 전송 방식에 의하여 이루어 지며 시스템의 구성과 동작은 iPSC 시스템과 유사하다.

최근 발표된 CM-5는 기존의 SIMD 시스템인 CM-1/2의 상향 호환을 위하여 SIMD 수행 방식과 MIMD 수행 방식의 선택 사용이 가능 하다. CM-5는 MIMD 시스템에서와 같이 각 처리기는 각기의 명령어를 인출, 수행할수 있으며, SIMD 시스템과 같이 instruction broadcasting을 위한 구조도 가지고

있다. 또한 CM-5는 source language level에서 기존의 CM-1/2와 호환성을 가지며 MIMD 프로그램의 수행시는 메세지 전송 방식을 사용 한다. CM-5의 각 node는 SPARC 처리기를 사용 하며 소프트웨어, I/O, 그리고 내부 연결 구조를 포함 하는 모든 부분에서 최대 16,000 개의 node까지 확장 가능 하며 각 처리 node는 32/64 bit vector unit을 선택하여 vector 연산 속도를 증대 시킬수 있다. 그림 4 에서와 같이 각 node의 처리기들은 CP (control processor)에 의해 제어 되어진다. 즉 프로그램 로딩 (loading)은 CP에서 이루어지고 CP는 병렬 처리기들에 일련의 수행 명령어들을 전송하고 수행하게한다. 또한 시스템사용의 효율성을 극대화 하기위해 병렬 처리기들은 여러개의 그룹으로 나뉘어질수 (partitionable) 있으며 partition manager라고 불리우는 독립 CP가 각 partition을 조정하고 있다. 그림 4 에서 보여지는 바와같이 시스템의 연결 구조로서 3 가지의 다른 구조를 제공한다. 즉 데이타 전송을 위한 내부 연결 구조로써 fat tree연결 구조가 사용되며 처리기들간의 병렬동작에 대한 컨트롤 (control)을 위하여 binary tree 연결 구조가 사용되어지고, 자가 진단용의을 위한 진단 연결 구조가 제공된다.

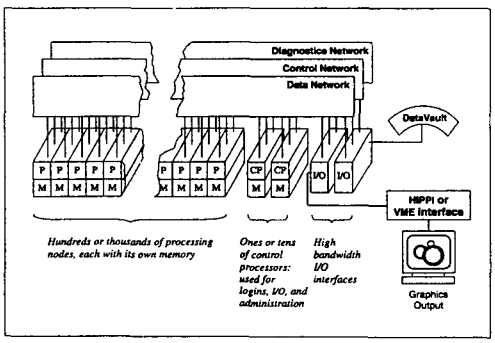

그림 4. Thinking Machine사의 CM-5 구조

Ⅳ. 요약 및 결론

현대의 과학과 산업을 주도 하게될 고성능 컴퓨터 개발에 대한 지속적인 요구와 그 중요성 때문에 다양한 병렬 처리 시스템들의 개발과 그 상용화가 점점 증대하고 현재에도 많은 연구와 투자가 이루어지고 있다. 또한 현재의 급속한 반도체 기술의 발달로 고성능의 off-the-shelf 처리기들의 경제적인 구입이 가능해지고, 메모리 가격의 지속적인 감소로 시스템의 메모리 구성 비용이 전체 시스템 구성 가격에 미치는 영향이 줄어듬에 따라 대단위 병렬 처리 시스템의 구성이 더욱 보편화 되어가고 있다. 이와 아울러 시스템의 확장성을 높이기 위하여 고속의 통신 대역폭를 갖는 내부 연결 구조 (interconnection network)가 요구되어 지며, 이를 위하여 보다 강력한 network의 구성 방법과 고속의 optical communication network도 연구 중이다. 이러한 추세속에서 많은 병렬 처리 컴퓨터 회사들은 성능 증대라는 공통 목표를 위해 다중업무 처리의 증대와 병렬성을 지원하는 확장형 공유메모리 MIMD 시스템계열과 단위 업무처리 속도 증대의 목표로써 응용범위에서의 보편성을 지원하는 대단위 분산 MIMD 시스템계열의 설계에 많은 노력을 하고있으며, 또한 데이터의 병렬성을 지원하며 경제적인 시스템 설계가 가능한 대단위 SIMD 병렬 처리 시스템들도 간과 되어질수는 없다. 또한 병렬성을 효율적으로 이용하기 위한 새로운 병렬 컴파일링 기법과 오퍼레이팅 시스템의 개발은 물론 각종 병렬 처리 소프트웨어의 개발 등에도 많은 연구가 이루어지고 있다. 이에 관련하여 본 기고문에서는 시스템 성능 측면에서 컴퓨터의 구조를 분류하고 다중 처리 시스템과 처리 속도 증대를 위한 대단위 병렬 처리 시스템들의 소개와 시스템들의 특징을 간단히 살펴 보았으며, 그러나 전자의 경우에는 대단위 병렬 처리 시스템으로 분류하기에는 미흡하지만 대단위 구성을 목표로하고 있으므로 대단위 병렬 처리 시스템의 한 분야로 본 기고에서 언급되어졌다.

병렬 처리 분야는 선진국에서 다양한 방식의 시스템들에 대해 집중 연구, 개발이 되고있는 분야로 근본적으로 고성능의 컴퓨팅 능력을 실현하기 위함으로써 그 미래의 구조적 경향도 1 장에서 언급한 시스템 성능의 두가지 기본 원칙에 의거 속도 증대와 다중 업무 처리량의 증대를 목표로 꾸준히 발전할 것이며 선형적 확장성의 구현이 시스템 설계의 주요 목표가 될것이다. 그러므로 대단위 다중 처리형 구조와 대단위 병렬 처리형 컴퓨터 구조의 두 주요 영역이 컴퓨팅 환경의 변화와 함께 지속적으로 발전해 나갈 것이

다. 그외에도 특정 목적을 위한 특수 응용 분야도 점점 세분화하여 최대의 처리 속도를 얻기위한 특정 목적의 고성능 컴퓨터의 개발도 꾸준히 발전할 것이다.

우리의 컴퓨팅 환경은 기존의 고속 scalar machine, vector machine (기존 supercomputer), 그리고 다양한 병렬 처리 시스템과 알고리즘 지향 (algorithm specific) machine (예: FFT machine, graphic accelerator) 등 점점 복잡 다양해 지고 이러한 컴퓨터들은 고속 network에 의해 연결 되어져 총체적으로 이기종 연산 (heterogeneous computing) 환경을 이루게 된다. 또한 기술의 혁신으로 각종 컴퓨터들 간에 고속 통신이 가능해짐에 따라 다양한 특성의 서로 다른 시스템 (heterogeneous machine)들의 상호 접속을 통해 이를 효율적으로 이용하는 방법이 최근에 많은 관심을 모으고 있으며 이를 Superconcurrenct Processing이라 부른다. 기본적인 idea는 일반적으로 복잡한 task들은 각기 다른 연산적 특성을 갖는 subtask들로 구성될 수 있으며 이기종 컴퓨터들중 어느 한 기종이 이러한 광범위한 문제에 대하여 ideal하지 않다는 것이다. 즉 각 subtask에 대해 가장 최적의 성능을 제공 하는 machine의 구조가 다르기 때문이다. 결론적으로 superconcurrent processing 방법은 주어진 문제에 대하여 가장 적절하고 효과적인 machine들을 선택 수행 하도록 하여 응용 프로그램의 처리 시간을 최소화 하는 것이다. 현재 IBM 이나 Cray, 그외 여러 연구소에서 이문제에대한 연구를 하고있다.

이와 같이 혁신적인 병렬 처리 시스템들의 개발과 이의 응용에 대한 연구는 물론 이분야에 대한 보다 포괄적인 이해로 고성능 컴퓨터의 향후 구조적 경향을 파악하여 옳바른 방향으로 국내의 컴퓨터 산업이 발전되어 나갈수 있도록 많은 관심과 노력을 기울여야할 것으로 사료된다.

參 考 文 獻

[1] R. Arlauskas, iPSC/2 system: a second generation hypercube, " 3rd *Conf. Hypercube Computers and Applications*, Jan. 1988, pp. 38-42.

[2] M. Auguin and F. Boeri, "The OPSILA computer," in *Parallel Languages and Architectures*, M. Consard, ed., Elsivier Science Publishers, Holland, 1986, pp. 143-153.

[3] K. E. Batcher, STARAN parallel processor system hardware, " *AFIPS 1974 Nat'l Computer Conf.*, May 1974, pp. 405-410.

[4] K. E. Batcher, Bit serial parallel processing systems,' ' *IEEE Trans. Computers*, vol. C-29, no. 5, May 1982, pp. 377-384.

[5] J. Beetem, M. Denneau, and D. Weingarten, GF11 - A supercomputer for scientific applications,' ' *Proc. of the 12th Int'l Symp. on Computer Architecture, IEEE Computer Society*, Boston, 1985.

[6] W. J. Bouknight et al., The Illiac IV system, " *Proc. of the IEEE*, vol. 60, no. 4, Apr. 1972, pp. 369-388.

[7] T. Blank, "The MasPar MP-1 architecture," *IEEE Compcon*, Feb. 1990, pp. 20-24.

[8] W. Crowther, J. Goodhue, R. Thomas, and T. Blackadar, "Performance measure ments on a 128-node butterfly parallel processor," *1985 Int'l Conf. Parallel Processing*, Aug. 1985, pp. 531-540.

[9] M. J. Flynn, "Very high-speed computing systems," *Proc. of the IEEE*, vol. 54, no. 12, Dec. 1966, pp. 1901-1909.

[10] T. J. Fountain, "CLIP4: process report," in *Language and Architectures for Image Processing*, M. J. B. Duff and S. Levialdi, eds., Academic Press, London, England, 1981, pp. 281-291.

[11] J. P. Hayes and T. Mudge, "Hypercube supercomputers," *Proc. of the IEEE*, vol. 77, no. 12, Dec. 1989, pp. 1829-

1841.

[12] W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.

[13] D. J. Hunt, "AMT DAP - a processor array in a workstation environment," *Computer Systems Science and Engineering*, vol. 4, no. 2, Apr. 1989, pp. 107-114.

[14] L. H. Jamieson, "Characterizing parallel algorithms," in *The Characteristics of Parallel Algorithms*, L. H. Jamieson, D. B. Gannon, and R. J. Douglass, eds., MIT Press, Cambridge, MA, 1987, pp. 65-100.

[15] A. K. Jones et al., "Software management of Cm* - a distributed multiprocessor," *AFIPS 1977 National Computer Conf.*, June 1977, pp. 657-663.

[16] G. J. Lipovski and M. Malek, Parallel Computing: *Theory and Comparisons*, John Wiley and Sons, Inc., New York, NY, 1987.

[17] G. F. Pfister et al., "The IBM research parallel processor prototype (RP3): introduction and architecture," *1985 Int'l Conf. Parallel Processing*, Aug. 1985, pp. 764-771.

[18] H. J. Siegel et al., "An overview of the PASM parallel processing system," in *Computer Architecture*, D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, eds., *IEEE Computer Society Press*, Washington, DC, 1987, pp. 387-407.

[19] L. W. Tucker and G. G. Robertson, "Architecture and applications of the Connection Machine," *Computer*, vol. 21, no. 8, Aug. 1988, pp. 26-38.

[20] A. Gottlieb et al., "The NYU Ultracomputer - Designing an MIMD shared memory parallel computer," *IEEE Trans. on Computers*, C-32, no. 2, Feb. 1983, pp. 175-189.

[21] J. Hennessy et al., "Overview and Status of the Stanford DASH Multiprocessor," *Proc. of the 5th ACM Symp. on Principles of Distributed Computing.*, Apr. 1991, pp. 229-239.

[22] A. Agrawal et al., "APRIL: A processor architecture for multipro-cessing," *17th Int'l Symp. Computer Architectures*, 1990, IEEE Computer Soc. Press, Los Alamos, Calif., pp. 104-114.

[23] Kendall Square Research, "KSR1 technical manual," 1992.

[24] Thinking Machine Corporation, "CM5 Technical summary," Nov. 1992.

筆 者 紹 介

金 新 德

1982年 2月 연세대학교 전자공학과 (학사)
1987年 12月 University of Oklahoma 전기공학과 (석사)
1991年 12月 Purdue University 전기공학과 (박사)

1982年 ~ 1986年 삼성전자 computer R & D
1988年 삼성전자 computer R & D
1988年 ~ 1991年 Research Assitant in the Parallel Processing Laboratary of Electrical Engineering at Purdue Univ.
1992年 ~ 1993年 삼성 종합기술원 정보시스템
1993年 ~ 현재 광운대학교 컴퓨터공학과 조교수

주관심분야 : parallel computer architecture, parallel algorithm design and mapping, parallel language constructs, and heterogeneous processing.

表 三 洙

1974年 서울공대 전자공학과 (학사)
1976年 한국과학원 전기 및 전자공학과 (석사)
1985年 Carnegie-Mellon University 전기 및 컴퓨터공학과 (박사)

1976 年 ~ 1980 年 한국원자력연구소 연구원
1980 年 인하 대학교 전기공학과 대우 교수
1984 年 ~ 1986 年 Syracuse University 전기 및 컴퓨터공학과 조교수 겸 New York State Center for Advanced Technology in Computer Applications and Software Engineering 연구원
1986 年 ~ 1990 年 University of Kentucky 전기공학과 조교수
1990 年 ~ 현재 삼성전자 정보 컴퓨터 본부 연구위원