□기술해설□

병렬처리를 위한 병렬언어

충남대학교 최숙영* · 유관종**

목 차

1. 서 론

최근에 병렬처리 시스템의 하드웨어 분야가 급격한 발전을 한것에 비해 이러한 시스템에서 수행되는 병렬 소프트웨어에 대한 연구는 크게 빈약한 실정이다.

병렬처리의 구현을 위한 병렬 프로그래밍에 대한 연구는 크게 두가지 관점으로 구분할 수 있다. 먼저 기존의 순차형 언어로 작성된 프로그램을 병렬 컴파일러를 이용하여 병렬 프로그램으로 변환하는 관점이다[10, 25]. 이 방법은 기존의 순차형 소프트웨어를 그대로 이용할 수 있고 사용자로 하여금 병렬 프로그램 작성에 관한 부담을 덜어줄 수 있기 때문에 이상적인 방법처럼 보일 수 있다. 그러나 실제로 병렬 컴파일러에 의해 수행되는 병렬화 부분은 극히 제한적이고 데이타 종속성(data dependency)과 같은 복잡한 문제를 해결해야 하기 때문에 병렬 처리를 수행하는 관점에서는 효율적이라 할 수 없다. 다른 관점은 병렬 처리에 적합한 병렬 프로그래밍 언어를 사용하여 사용자가 직접 병렬 프로그램을 작성하는 관점이다. 이러한 병렬 프로그래밍 언어는 다시 다음과 같이 분류될 수 있다. 첫번째 부류는 기존의 순차형 언어에 병렬성을 추가하여 언어를 확장시킨 것으로써 Concurrent C, Concurrent Pascal, PARLOG, HPF, CC++[8, 21, 17, 20] 등이 존재한다. 두번째 부류는 새로운 병렬 프로그래밍 언어를 개발하는 것으로 언어 설계시 병렬 구조의 제공에 역점을 둔 Occam, Ada, POOL-T [11, 16, 22] 등이 있다. 이러한 관점은 병렬처리를 효과적으로 수행할 수는 있으나 병렬성의 고려, 프로세스간의 통신과 동기화와 같은 문제를 사용자가 직접 고려해야 하기 때문에 사용자는 많은 부담감을 갖는다.

이와 같이 각 관점들은 장·단점들을 지니고 있으며, 이중 어느 한 관점이 병렬처리를 수행하기 위한 이상적인 방법이라고는 할 수 없다. 두가지 관점들을 기준으로 하여 계속 많은 연구들이 수행되고 있다.

본 고에서는 후자의 관점인 병렬처리를 위한 병렬 프로그래밍 언어에 대해서 살펴보고자 한

*학생회원
**종신회원

다. 어떤 응용에 관해 병렬처리를 수행하기 위하여 병렬 프로그래밍 언어를 선택할 때에는 다음과 같은 사항을 고려해야 한다. 그 언어가 그 문제에 적합한 것인지, 또한 주어진 하드웨어 시스템에서 그 언어가 효율적으로 구현될 수 있는지 등을 고려해야 한다. 각각의 병렬 프로그래밍 언어들은 여러가지 요인들에 의해 분류될 수 있다. 본 고에서는 각 언어가 추구하고 있는 병렬성의 관점과 통신 및 동기화 방법들에 따라 각 언어들을 분류함으로써 각 언어의 특징들을 살펴보고자 한다.

2. 병렬 프로그래밍 언어의 분류 방법

병렬 프로그램이란 동시에 수행가능한 작업들 중, 프로세스들이 서로간에 상호작용(interaction)을 함으로써 수행된다고 볼 수 있다. 이렇게 동시에 수행가능한 작업들의 단위를 어떻게 다루는가, 또한 각 병렬로 수행되는 작업들 사이의 통신과 동기화가 어떻게 수행되는가에 따라 여러 병렬처리 언어의 특징들을 구분할 수 있다 .본 장에서는 병렬 프로그래밍 언어들이 추구하는 병렬성들의 여러 관점과 통신 방법들을 살펴보고자 한다.

2.1 병렬성의 관점

2.1.1 절차적 언어와 선언적 언어

프로그래밍 언어는 크게 절차적(imperative) 언어와 선언적(declarative) 언어로 분류할 수 있다[13]. 폰노이만 구조를 바탕으로 하는 대부분의 순차형 언어는 절차적 언어로써 주어진 문제를 해결하기 위해 수행되는 과정을 프로그램상에 기술해야 한다. 선언적 언어는 절차적 언어와는 달리 해결하고자 하는 문제가 무엇인지를 기술하는 것이다. 대부분의 프로그래머들은 절차적 언어에 익숙해져 있다. 병렬 처리 시스템이 발전됨에 따라 병렬 처리를 효과적으로 수행하기 위하여 기존의 절차적 언어에 병렬성을 추가한 병렬 언어들이 많이 개발되었다. 이러한 언어들로써 Concurrent C, Concurrent Pascal, HPF, Fortran D 등이 존재한다. 또한 언어 설계시 병렬성을 고려하여 만든 병렬 절차적 언어로써 Ada, Occam등이 있다. 대부분의 이러한 절차적 언어들은 병렬성을 명시적(explicit)으로 표현하고 있으며, 큰 단위(coarse-grain)의 병렬성을 제공하고 있다.

선언적 언어에는 함수형 언어(functional language), 논리형 언어(logic language) 및 데이타플로우 언어(dataflow language) 등이 존재한다. 이러한 언어들의 특징은 언어 자체에 병렬성이 내재되어 있기 때문에 암시적인(implicit) 병렬성을 얻을 수 있다. 함수형 언어의 장점으로는 부작용(side effect)이 존재하지 않기 때문에 함수(function)들이 실행되는 순서에 따라 영향을 미치지 않는다. 따라서, 한 식(expression)에서 수행되는 함수들은 서로 영향을 미치지 않기 때문에 병렬로 수행할 수 있다. 함수형 언어에 병렬 프로그래밍을 접근하려는 언어들로써 Multilisp, Concurrent Lisp, ParAlfl[8, 18] 등이 있다. 논리형 언어는 추론과정에서 병렬성을 얻을 수 있으며, AND 병렬성과 OR 병렬성을 지원하고 있다. AND 병렬성은 현재 실행중인 절(clause)의 몸체부를 구성하는 부목표(sub-goal)들을 병렬로 실행하는 것이고, OR 병렬성은 주어진 목표(goal)를 만족하는 절을 찾기 위해 여러개의 절들을 병렬로 시도하는 것이다. 이러한 논리형 언어에 병렬 프로그래밍을 접근하려는 언어로써 Concurrent Prolog, Prolog, PARLOG, P-PROLGO, BRAVE[8, 17] 등이 있다. 이러한 선언적 언어들의 특징은 암시적인 병렬성과 미세한 단위(fine-grain)의 병렬성을 제공하고 있다는 점이다.

2.1.2 언어구조의 확장과 라이브러리 지원

병렬성을 지원하는 방법은 기존의 언어에 명시적인 병렬 언어 구조를 추가하여 언어를 확장시키는 방법과 라이브러리(libraries) 형태로 지원하는 방법으로 구분할 수 있다[2, 20]. 전자의 경우 추가된 병렬 구조는 컴파일러에 의해 적당한 저급 수준의 연산자들로 변환된다. 병렬 구조의 종류로 병렬 블럭의 시작/끝(parbegin/parend)과 스레드(thread)를 생성하는(fork/join)문, 데이타를 교환하는 통신 연산자(send/receive) 등을 사용하여 프로그램을 작성할 수 있다. 이러한 구조를 가지고 프로그램을 작성하

는 것은 프로그램상에 병렬성을 명백하게 제시할 수 있는 장점이 있는 반면에, 기존의 순차형 언어 구조에 병렬 구조를 논리적이고 효과적으로 통합시키는 방법은 어려운 작업이다. Concurrent C, Parallel Pascal, Multilisp, Fortran M 등은 이 범주에 속하는 언어들이다.

위와 같은 방법과는 달리 기존의 순차형 언어에 병렬성을 분리된 라이브러리 형태로 지원하는 방법이 있다. 프로그래머는 스레드를 생성하거나 다루기 위해 라이브러리에 의해 정의된 함수를 이용할 수 있다. 이러한 라이브러리 루틴들은 언어와의 독립성을 유지할 수 있기 때문에 시스템이 바뀌더라도 라이브러리 루틴들을 재작성 한다거나 재번역할 필요가 없이 라이브러리만을 새로운 버젼으로 쉽게 대치하면 된다. 그러나 라이브러리 루틴과 프로그램의 구조/의미상에 분명히 정의된 관계가 존재하지 않기 때문에 디버깅하는 작업이 까다롭다. 또한 이러한 병렬 라이브러리들은 시스템 구성과 밀접한 관련이 있기 때문에 한 프로그램을 다른 시스템상에서 수행할 때는 비효율적일 수도 있고 원하지 않은 결과를 얻을 수도 있다. 이러한 관점을 대표하는 언어에는 P4, PVM, Linda 등이 있다.

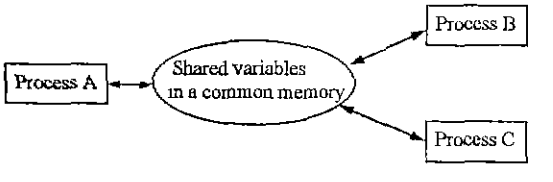
2.1.3 제어 병렬성과 자료 병렬성

병렬처리가 수행되는 내용에 따라 제어 병렬성(control parallelism)과 자료 병렬성(data parallelism)으로 구분할 수 있다[12, 20]. 제어 병렬성은 병렬 시스템의 각 프로세서상에서 서로 다른 연산들이 동시에 수행되는 것으로, 파이프라인(pipeline)은 대표적인 제어 병렬성의 예이다. 이러한 제어 병렬성은 대부분 MIMD 컴퓨터에서 구현하기 쉽다. 실세계의 많은 문제들은 문제의 특성상 병렬로 수행할 수 있는 서로 독립적인 모듈들로 구성될 수 있기 때문에 제어 병렬성을 이용하여 효과적으로 프로그램을 작성할 수 있다. 이 병렬성의 관점에서는 병렬성의 단위를 어떻게 다룰것인가가 고려될 수 있는 사항이다. 여러 수준에서의 병렬성들(명령어(instruction), 프로세스(process), 오브젝트(object), 식(expression), 절(clause), 타스크(task)이 존재하며, 병렬 프로그래밍 언어에 따라 지원하는 병렬성의 단위가 달라진다.

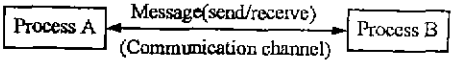
자료 병렬성은 각각의 프로세서상에서 각각 다른 데이타들을 가지고 동일한 연산을 수행하는 것이다. 이러한 자료 병렬성은 SPMD 컴퓨터와 SIMD 컴퓨터의 lockstep 연산으로 쉽게 구현될 수 있다. 자료 병렬성은 동기적 제어하에 수행되는 미세한 단위(fine-grain)의 병렬성을 구현할 수 있으며 높은 병렬성을 얻을 수 있다. 자료 병렬성을 지원하는 프로그램은 각 프로세서상에 서로 다른 데이타를 분배하는 작업이 요구되는데, 이러한 작업은 두단계로 구분할 수 있다. 먼저, 첫번째 단계는 수행될 데이타를 가상 프로세서들(virtual processors) 상에 분배하는 작업이고, 두번째 단계는 가상 프로세서들에 분배된 데이터를 어느 시점에서 실제 프로세서들(physical processors)에 할당(map)시키는 과정이다. 대부분의 자료 병렬성을 지원하는 언어들은 시스템의 구조적 특성을 프로그래머로부터 숨기기 때문에, 프로세서간에 명시적인 메시지 전달을 위한 프로그램에 비해 대체적으로 프로그램 작성이 쉽다. 그러나 사용자 입장에서 이러한 프로그램 작성의 용이함은 컴파일러의 작업을 그만큼 복잡하고 어렵게 만들 수 있다. 이러한 자료 병렬성을 지원하는 대표적인 언어에는 C*, Fortran D, Fortran 90, HPF 등이 있다.

2.2. 통신 방법 및 동기화 기법

병렬 프로그램에서 각 프로세스간의 통신 방법은 공유 변수(shared-variable)를 이용하는



(a) 공유 변수를 이용한 통신



(b) 메시지 전송을 이용한 통신

그림 1 프로세스간의 통신 방법

방법과 메시지 전송(message-passing)을 이용하는 방법이 존재하며, 그림 1은 이 방법들의 수행과정을 나타낸다.

2.2.1 공유 변수 방법

공유 변수 방법은 각 프로세스간의 통신을 위해 공유 메모리(shared memory)를 사용하는 방법으로 공유 메모리 시스템에서 구현이 쉽다. 공유 변수를 이용하는 프로그래밍은 기본적으로 공유 변수들을 정의하고 프로세스들을 생성하는 구조가 필요하다. 또한 어떤 단위 시간에 하나 이상의 프로세스가 공유 변수를 접근하게 되면 원하지 않는 결과를 얻을 수 있기 때문에, 여러 프로세스가 동시에 공유 데이타를 접근하지 못하도록 상호 배제(mutual exclusion)와 동기화(synchronization)가 요구된다. 이를 구현하기 위한 방법으로 락(lock)과 barrier synchronization을 사용하며, 좀 더 고수준(high-level)의 동기화 기법으로 세마포어(semaphore)와 모니터(monitor)를 사용하기도 한다. 기존의 순차형 언어인 Fortran과 C언어에 공유 변수를 기본으로 하는 병렬성을 추가하려는 연구들로써 병렬로 실행되는 프로세스들의 시작과 끝을 표시하는 구조, 각 프로세스간의 동기화를 제공하는 구조, 프로세스들간의 공유 변수를 나타내는 프리미티브(primitive)들이 추가되었다. 프로세스간의 통신 방법으로 공유 변수를 사용하는 대표적인 언어에는 Concurrent Pascal, COOL, Sequent C 등이 있다.

2.2.2 메시지 전달 방법

메시지 전달 방법은 각각의 프로세스들이 지역(local) 변수를 가지고 있고 프로세스간의 통신은 메시지를 전달함으로써 수행된다. 메시지 전달에 의한 통신은 공유 변수를 사용하는 것보다 통신 시간이 지연될 수 있다. 메시지 전달을 이용하는 프로그래밍의 기본 요소에는 송신 프로세스와 수신 프로세스에 의해 사용되는 send() receive() 연산자가 있다.

메시지 전달은 수행되는 방식의 차이에 따라 동기식 메시지 전송(synchronous message passing), 비동기식 메시지 전송(asynchronus message passing), 랑데부(rendezvous), 원격 프로시쥬어 호출(remote procedure call) 등으로 분류할 수 있다[8, 13].

동기식 메시지 전송에서는 송신자가 메시지를 보낸 후 수신자가 그 메시지를 받을 때까지 수행을 멈추고 기다리는 것으로 송신자와 수신자 사이에 동기화가 이루어진다. 이러한 방식에 의거하여 수행되는 언어는 CSP에 기반을 둔 Occam이다.

비동기식 메시지 전송은 송신자가 메시지를 보낸 후 수신자가 메시지를 받을 때까지 기다릴 필요없이 다음 작업을 수행한다. 이러한 방식은 송신자와 수신자간에 메시지를 저장하기 위한 버퍼(buffer)를 필요로 한다. 이러한 방식에 기초한 언어에는 Concurrent C가 있다.

랑데부(rendezvous)는 Ada에서 사용되는 동기화 및 통신 방식이다[22]. 랑데부는 양방향(two-way) 통신 방식으로 수행되며, 통신이 이루어지는 작업들간에는 엔트리(entry)라 불리는 동기화 지점이 포함된다. 이 방식은 다음과 같은 세가지 개념으로 구성된다 : entry declaration, entry call, accept statement. 두 프로세스간의 랑데부는 한 프로세스가 다른 프로세

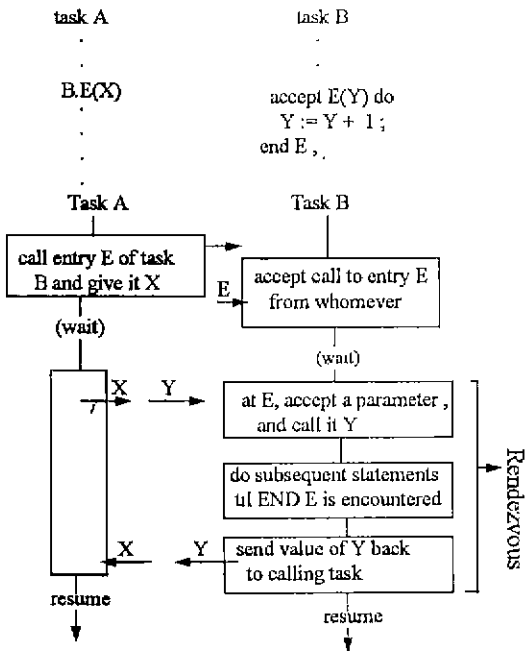


그림 2 Ada의 랑데뷰 과정

스의 엔트리를 호출할 때 발생하고 다른 프로세스는 그 엔트리를 위해 aceept statement를 수행한다. 수행방식은 동기식으로 두개의 프로세스 r과 s가 랑데부하기 위해서는 랑데부 시점에서 r은 s의 엔트리를 호출할 수 있어야 하며, s는 r의 호출을 받을 준비가 되어 있어야 한다. 만약 랑데부에 참여하는 프로세스 중 어느 하나가 준비되어 있지 않으면 지연된다. 그림 2는 Ada의 랑데부에 대한 수행 과정을 보여주고 있다.

원격 프로시쥬어 호출(remote procedure call)은 양방향 통신 방식으로 랑데부 방식과 유사하게 수행되며 한 프로세스 r이 다른 프로세스 s의 원격 프로시쥬어를 호출했을 경우, r에 의해서 제공되는 프로시쥬어의 입력 인수(input parameter)가 s에 넘겨진다. s는 실행 요구를 받고 프로시쥬어를 수행하고 난 다음, 그 결과를 r에 보낸다. 이때 s가 프로시쥬어를 수행하는 동안에 r은 블럭(block)되며, 결과값의 도착에 의해 r은 수행을 재개하게 된다. 랑데부와 원격 프로시쥬어 호출은 언어와 독립적으로 사용될 수 있기 때문에 여러 언어(Fortran, C, Pascal)에서 구현되고 있으며 시스템상에 패킷으로 제공되고 있다. 그림 3은 원격 프로시쥬어 호출에 대한 흐름도를 보여주고 있다.

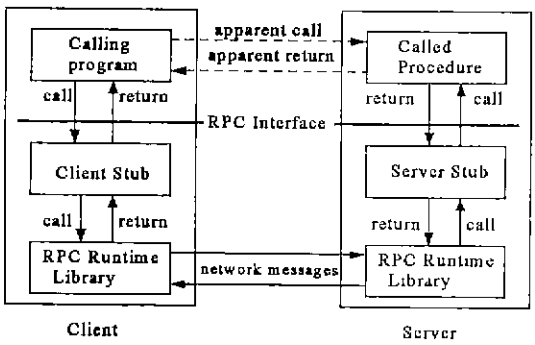


그림 3 원격 프로 시쥬어 호출의 구성도

3. 대표적인 병렬 프로그래밍 언어들

이 장에서는 앞에서 살펴본 병렬 프로그래밍 언어의 여러 관점들을 대표하는 병렬 프로그래밍 모델에 따라 병렬 언어들을 소개한다. 이 장에서 고려하는 병렬 프로그래밍 모델들은 메시지 전송 모델, 공유 변수 모델, 자료 병렬성 모델, 함수형 모델, 논리형 모델, 객체지향 모델 등이다.

3.1 Occam(메시지 전송 모델)

Occam언어는 메시지 전송 방식을 이용하는 대표적인 병렬 언어로써 CSP에 기반을 둔 언어이다[11]. CSP는 Hoare[24]에 의해서 개발되었으며 완전한 병렬 언어로써 개발된 것이 아니라, 다른 병렬 언어를 개발하는데 필요한 기본 요소를 제공하고 있다. CSP에서 모든 프로세스는 동기적인 send 명령과 receive 명령을 사용하여 통신이 이루어진다.

Occam은 Inmos사의 트랜스퓨터(transputer)를 프로세서로 구성하는 시스템을 위해 설계되었으며, 트랜스퓨터의 어셈블리 언어라 할 수 있다. Occam은 CSP와는 달리 두 프로세스간의 단방향(one-way) 링크로 구성되는 채널(channel)을 통하여 통신이 수행된다. Occam에서 기본적으로 사용하는 개념은 프로세스와 통신 채널이다. 프로세스란 문장들의 그룹이 될 수도 있고 프로세스들의 모임이 될 수 있다. 채널은 두 프로세스들 사이에 메시지 전송을 이용하여 통신을 가능하게 해주는 통신 링크이다. Occam은 다음과 같은 세개의 기본 프로세스들로 구성된다.

assignment : (x : = x + 1)
input : (chan? y)
output : (chan! x)

Occam은 위의 기본 프로세스들을 바탕으로 하여 프로세스의 병렬 실행과 순차 실행을 명시적으로 표현할 수 있는 다음과 같은 프로그램 제어 구조를 제공한다.

sequential constuctor(SEQ)
parallel constuctor(PAR)
alternative constructor(ALT)
replicator(FOR)

ALT 구조는 여러 프로세스들이 실행가능한

경우 그중 하나만을 임의로 선택하는 비결정적(*nondeterminal*) 특징을 갖고 있다. FOR는 n개의 프로세스를 생성하는 구조로써 SEQ, PAR, ALT 구조와 혼합하여 사용될 수 있다. 다음은 1024개의 FIFO buffer 프로세스를 병렬로 생성하는 프로그램이다.

```
[1025] CHAN OF INT Buffer :
PAR index = 0 FOR 1024
SEQ
  Buffer[index] ? Item
  Buffer[index + 1] ! Item
```

3.2 Concurrent Pascal(공유 변수 모델)

Concurrent Pascal은 프로세스간의 통신이 공유 변수를 기초로 하여 수행되는 대표적인 언어이다[21]. 이 언어는 순차형 Pascal에 프로세스, 클래스(class), 모니터(monitor)의 개념을 추가하여 만든 공유 메모리 시스템을 위한 병렬 언어이다. 프로세스는 다른 프로세서들과 동시에 수행될 수 있는 하나의 순차형 프로그램이며, 클래스는 한 프로세스나 모니터에 의해 수행될 수 있는 연산자들과 자료 구조를 정의한다.

모니터는 공유 자료 구조와 프로세스들에 의해서 수행될 수 있는 연산자들을 정의한다. 또한 모니터는 프로세스들 사이에 공유되는 데이타를 보호하는 메카니즘으로 공유 데이타에 대한 접근을 동기화 한다. 즉, 모니터는 프로시쥬어를 호출하는 수단과 다른 외부의 프로시쥬어에 의해 발생되는 다양한 요구들을 스케쥴링하는 수단을 제공함으로써, 각 프로세스들이 공유되는 자원에 접근하는데 충돌이 발생하지 않도록 감시하는 역할을 한다. Concurrent Pascal에서는 이러한 모니터에 관련된 연산자로써 delay와 continue를 두고 있으며, 자료 구조로 queue를 정의하여 사용한다.

3.3 C*(자료 병렬성 모델)

C*은 C언어에 자료 병렬성을 추가하여 확장된 언어이다[12, 14]. 이 언어는 CM-2 병렬 컴퓨터를 위해 설계되었으며, 자료 병렬성을 지원하기 위한 특징들을 제공한다. C*은 두가지 형태의 변수들(scalar/parallel)을 제공하고 있다. Scalar 변수는 C의 일반적인 변수와 동일하고, 호스트 프로세서(host processor)에 할당되어 진다.

Parallel 변수는 모든 노드 프로세서들(node processors)에 할당되며 시스템의 프로세스들의 수만큼 요소들을 가질 수 있다. Parallel 변수는 type뿐만 아니라 shape를 가지고 있는데, shape란 병렬 데이타가 논리적으로 구성되는 한 형태를 의미한다. Shape에서 차원의 수를 rank라 부르고, 각 차원의 프로세서 갯수를 position이라 부른다. 한 shape는 주어진 문제의 데이타들에 대한 가장 적절한 조직 형태를 반영해야 한다. 한 shape가 정의된 후, 그 shape에 대한 parallel 변수들을 정의할 수 있다. 다음 문장은 type이 정수이고 shape의 *rank*가 1이고 position이 8192인 ring에 대한 parallel 변수 count를 정의한 것이다.

```
shape[8192] ring :
int : ring count
```

Parallel 연산자는 parallel 변수와 관련이 있다. 예를 들어, 할당문 x + = y에서 x와 y가 parallel 변수일 때 y의 각 요소의 값을 동일한 형태인 x의 각 요소의 값에 더하게 된다. Parallel 연산자는 대부분 with 문과 같이 사용된다. With문은 현재의 선택함으로써 parallel 변수들에 대한 각요소의 연산이 병렬로 수행될 수 있도록 한다. 또한 where문을 두고 있는데, 이 문은 parallel 변수의 부분 요소들을 지정할 수 있도록 한다. 다음은 with문과 where문을 사용한 예제로써, 0으로 나누어지는 것을 막기 위해 where 문을 사용하고 있다.

```
shape[8192] ring ;
int : ring x, y, z
with(ring){
where(z ! = 0)
x = y/z}
```

C*에서 프로세서간의 데이타 통신은 parallel 변수를 사용한 *left indexing*에 의해 수행되

며, parallel left indexing이라 부른다. Parallel left indexing는 parallel index의 값 요소에 저장된 값에 기초해서 parallel 변수의 요소를 재정렬하는 것으로, send와 get 연산자를 제공하고 있다. 만약 index, dest, source가 parallel 변수이고 각 rank가 1이라고 할때, send와 get 연산자에 대한 일반 형식과 수행 과정은 그림 4와 같다.

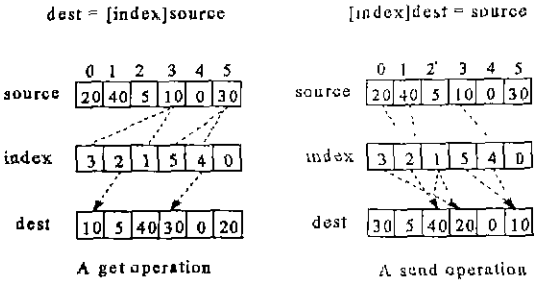


그림 4 send/get 통신 연산자의 형식과 수행

3.4 Multilisp(함수형 모델)

Multilisp는 Halstead[18]에 의해 제안되었으며, 기존의 함수형 언어인 Lisp에 명시적인 병렬성의 개념을 포함시킨 병렬 언어이다. Multilisp는 공유 메모리 모델에 기초를 두고 있다.

이 언어의 기본 개념은 프로그램의 융통성과 표현력을 높이기 위해 부작용(side effect)을 허용하는 점과, 병렬성을 명시적으로 표현하고 있다는 점이다. 병렬성을 동기화를 제공하기 위한 구조로써 PCALL과 future를 제공한다.

PCALL은 식(expression)들이 병렬로 계산되는(evaluate)것을 표현한다. future는 좀 더 강력한 병렬성을 지원하는 구조로써(future X)의 의미는 식 X의 값을 위해 future를 반환함과 동시에 X를 계산하기 시작한다. X의 계산에 의해 하나의 값이 생성되면 그 값은 future로써 대치된다. 이러한 future의 성격은 변수의 개념과 유사함을 알 수 있다. 예를 들어 다음의 문장에서 PCALL의 실행은 A와 B를 병렬로 계산한 다음 cons를 수행하기 위해 A와 B를

(PCALL cons A B)

(cons (future A) (future B)

위한 conscell를 생성하게 된다. 그 다음 이 생성된 conscell에 A와 B의 계산된 값을 넣게 된다. 그러나, future의 경우에는 좀 더 병렬성을 얻을 수 있다. 실제로 cons 연산을 수행하기 위해 conscell를 생성할 때는 A와 B의 계산된 값이 필요 없으므로, future의 특성을 이용하여 conscell을 생성하는 과정과 A와 B를 계산하는 과정을 병렬로 수행할 수 있다. A와 B의 계산된 값을 conscell에 바로 넣게 됨으로써 PCALL에 비해 좀 더 병렬성을 얻을 수 있다.

원래의 함수형 언어에서는 부작용을 허용하고 있지 않지만, Multilisp는 언어의 융통성을 부여하기 위하여 부작용을 지원하고 있으며, 명시적인 병렬성 구조를 제공함으로써 부작용으로 인해 방해되는 병렬성을 해결하고 있다.

3.5 PARLOG(논리형 모델)

PARLOG는 Imperial 대학의 Keith Clark와 Steve Gregory[17]에 개발된 병렬 언어이다. 이 언어는 다른 병렬 논리 언어와 같이 AND/OR 병렬성에 기초하고 있다.

AND 병렬성은 절(clause)의 몸체부를 구성하는 다수개의 부목표(sub-goal)을 동시에 실행하기 위해 시도하는 것이다. 모든 부목표들이 동시에 만족되거나 혹은 그 중 하나만이라도 실패될때 끝나게 되므로 AND 병렬성이라 부른다.

OR 병렬성은 주어진 목표를 만족하는 여러 개의 절이 존재할 때 동시에 실행되어 그 중 하나가 만족되면 끝나게 된다. PARLOG는 절의 부목표들을 연결하는 두개의 다른 접속 연산자를 두고 있다. 그 중 ','는 두개의 부목표가 병렬로 실행되는 것을 의미하고, '&'는 순차적으로 실행되는 것을 의미한다. 또한 한 릴레이션(relation)에 관한 절들은 '.'과 ';'에 의해 구분될 수 있다. 한 목표에 일치되는 절을 발견하려 할 때 '.'에 구분되는 모든 절들은 병렬로 수행될 수 있는데 이것을 OR 병렬성이라 한다. 다음 예에서

(1) A←(B & C), (D & E) ;

(2) A←F, G.
(3) A←H & J.

절 1은 (B & G)와 (D & E)를 병렬로 실행하면서 첫번째로 시도될 것이다. 만약 절 1이 실패되면 절 2와 3이 병렬로 시도될 것이다.

PARLOG에서 프로세스들은 공유 논리 변수를 통하여 통신이 수행되고, 비결정성을 위해 가아드 혼절(Guarded Horn Clause)을 사용한다.

3.6 POOL-T(객체지향 모델)

객체지향 모델은 객체를 하나의 독립적인 모듈로 보고 있으며, 객체들간의 정보교환은 메시지를 통하여 수행되도록 하고 있다. 객체지향 모델에서 한 시스템은 객체들로 구성되고, 각 객체들간의 상호작용에 의해 시스템이 수행되도록 하고 있다. 이러한 객체지향 개념의 특징들은 병렬처리 시스템을 자연스럽게 지원할 수 있다. 객체지향 개념의 장점들을 병렬처리 시스템에 도입하려는 시도로써 많은 객체지향 언어(ABCL/1, POOL, SINA..)들이 개발되었다.

POOL-T는 Esprit 프로젝트[4, 16]에서 개발된 병렬 객체 언어로써 DOOM이라는 분산 객체지향 시스템(distributed object-oriented system) 상에서 수행되는 언어이다. DOOM은 각각 개별 메모리(private memory)를 갖는 다수의 프로세서들이 패킷 스위칭(packet switching) 네트워크에 의해 연결되어 있다. POOL-T에서 하나의 프로그램은 메시지 전송에 의해 통신이 이루어지는 다수 객체들로 구성된다. 객체들은 데이터와 프로시쥬어로 구성되며, 생성시 동적인(active) 상태가 된다. 클래스는 객체들의 공통적인 특징을 정의하며 정적(static)인 특성을 지닌다.

POOL-T에서는 모듈의 개념으로써 unit을 제공하는데 다음과 같은 세가지 종류가 있다: root, specification, implementation. Specification unit과 implementation unit은 쌍으로 존재하며 여러개의 class 정의를 포함한다. Specification unit은 다른 unit에 의해 접근될 수 있는 class와 method를 기술한다. POOL-T에서는 root unit에 있는 마지막 클래스의 한 인스턴스(instance)를 암시적으로 생성하게 됨으로써 프로그램의 수행이 시작된다. 여기서 root unit은 전형적인 주 프로그램과 같이 작동된다. 수행시 여러 객체들이 동적으로 생성되어 병렬로 수행하지만 객체안에서는 모든 수행이 순차적으로 실행된다.

메시지 전송 방식은 동기화 방식이며, 메시지의 송신자는 항상 응답이 올 때까지 기다리게 되는데 이러한 과정을 랑데부라 부른다. 메시지의 송신과 수신은 명시적으로 수행되며 다음과 같은 형식을 가진다.

```
V ! put(56)              (송신 메시지 형식)
ANSWER(put, get)   (수신 메시지 형식)
```

송신 메시지에서 V는 수신 객체명이고 put은 수행해야 할 메시지이다. 수신 메시지의 두 메소드 중에서 먼저 도착한 메시지가 수행된다.

POOL-T에서의 실행은 하나의 동적 객체로 시작되며 메시지를 받아서 응답한 경우에도 동적인 상태로 남아있게 되므로 수행을 계속할 수 있다. 이것은 응답을 한 후에 post-processing section을 둠으로써 가능하다. 이때 송신자는 대기하지 않으므로 병렬성을 얻을 수 있다. 다음은 이러한 예로써, Bounded Buffer 문제를 위한 프로그램에서 Buffer 클래스에 정의된 put 메소드에 대한 기술이다.

```
METHOD put(elt : Integer) Buffer :
 contents ! put(in, elt) ## rendezvous 문장
 RETURN SELF
POST            ## post-processing section
 in←(in//size) +1
 number←number +1
END put
```

3.7 Linda

Linda는 Yale대학의 David Gelernter와 그의 동료들[9]에 의해서 개발되었다. 이 언어는 공유 변수나 메시지 전달에 기초한 프로그래밍 모델이 아니고 튜플 공간(TS : Tuple Space)라는 독특한 통신 메카니즘을 사용한다. 그림 5에서 볼 수 있는 것처럼 TS를 이용하여 프로세스

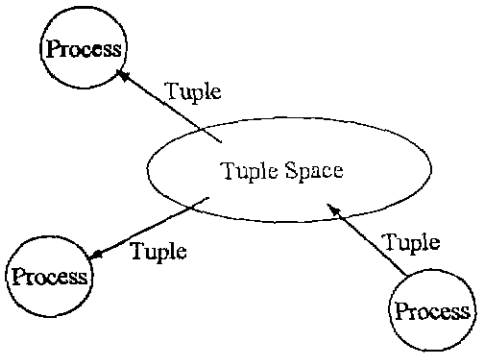


그림 5 Linda의 프로그래밍 모델

간의 통신이 단순하게 이루어짐을 볼 수 있다. Linda는 병렬처리를 수행하기 위해 프로그래머가 복잡한 병렬 계산과 동시성의 관점들을 고려하지 않음으로써, 병렬 프로그램을 작성하는데 따르는 수고를 줄여보자는 것을 기본 목표로 삼았다. TS는 개념적으로 볼 때 한 프로그램에 속하는 모든 프로세스들에 의해 공유되는 하나의 공유 메모리이다. TS의 한 요소는 튜플이라 불리우며 파스칼의 레코드와 유사하다.

Linda는 TS에 다음과 같은 연산자들을 정의한다.

read : TS에 포함된 한 튜플을 읽는다.
in : TS에 있는 한 튜플을 읽고 꺼낸다.
out : TS에 튜플을 삽입한다.
eval : 한 튜플을 실행하기 위해 프로세스를 생성한다.

프로세스들은 TS에 새로운 튜플을 삽입하고, TS에 존재하는 튜플을 꺼내거나 읽음으로써 서로 통신하게 된다. 한 튜플은 값이나 타입에 의해 표시될 수 있다. 예를 들어 다음과 같이 age가 정수형이고 married가 부울형태일 때

read("jones", var age, var married)
in ("jones", var age, var married)

만약 TS에 튜플 ["jones", 31, true]이 존재할 때 위의 두 연산자는 이 튜플을 읽거나 꺼낼 수 있다. 이 두 연산에서 변수 age는 두번째 필드의 값 31이 할당되고 married는 마지막 필드의 값 true를 얻는다. TS에서 튜플들을 찾으려 할 때 여러개의 일치되는 튜플들이 존재할 때는, 그 중 하나가 임의로 선택된다. 만약에 일치되는 것이 없는 경우에는 다른 프로세스가 그 해당 튜플을 넣을 때까지 멈추고 기다린다. Linda는 메시지 전송을 통해 수행되는 프로세스간의 통신 방법과는 대조적으로 익명(anonymous)의 방법이 사용된다. 즉, TS로부터 한 튜플을 읽는 프로세스는 어떤 프로세스가 그 튜플을 넣었는지 모르고 있으며, 고려하지 않아도 된다.

Linda의 장점으로는 기존의 언어에 기본적인 Linda의 메카니즘을 추가하여 쉽게 병렬 프로그래밍을 수행할 수 있다는 점이며, C-linda는 그러한 예에 속한다.

4. 결 론

병렬 처리 시스템을 효과적으로 이용하기 위한 많은 병렬 프로그래밍 언어들이 개발되었다. 병렬처리 언어에는 크게 기존의 순차형 언어에 병렬성을 추가하여 확장한 언어들과 병렬처리를 목적으로 새롭게 개발된 병렬 언어들이 존재한다. 전자의 경우는 순차형 언어에 부분적인 병렬구조를 추가하였기 때문에 사용자가 언어를 익히는데 많은 부담감이 따르지 않는 대신, 기존의 언어 구조와 효과적으로 통합하는 일은 어려운 작업이다. 후자의 관점은 사용자 입장에서 기존의 순차형 언어와는 다른 새로운 병렬 프로그래밍 언어를 익히는 작업은 매우 부담스러울 수 있으나, 효과적인 병렬 프로그램을 작성할 수 있다는 장점이 존재한다. 이러한 두가지 관점의 병렬 프로그래밍 언어들은 추구하는 병렬성의 성격과 단위, 프로세스들 사이의 통신 방법과 동기화 기법에 따라 좀 더 자세하게 분류될 수 있다.

본고에서는 병렬 프로그래밍 언어를 이와 같은 관점에서 살펴보았다. 병렬 프로그래밍 언어는 그 언어가 고려하는 계산 모델과 추구하는 병렬성, 통신 방법 등에 따라 각각 다른 특징들을 지니고 있기 때문에 어떤 응용문제에 적합한 언어가 다른 응용문제에서는 적합하지 않을 수가 있다. 따라서, 여러가지 응용 문제에 모두

적합한 이상적인 병렬 프로그래밍 언어는 존재하지 않으며, 수행되는 병렬 시스템의 성격과 해결하고자 하는 응용 문제에 따라 각기 이용될 수 있는 언어는 달라진다.

참고문헌

[1] A.H.Karp, "Programing for Parallelism," *IEEE Computer*, pp. 43-56, May 1987.

[2] C.M.Pancake and D.Bergmark, "Do Parallel Languages Respond to the Needs of Scientific Programmers?," *IEEE Computer*, pp. 13-23, Dec. 1990.

[3] A.S.Grimshaw, "Easy-to-Use Object-Oriented Parallel Processing with Mentat," *IEEE Computer*, pp. 39-51, May 1993.

[4] W.J.H.J.Bronnenberg, et al., "DOOM : Decentralized Object-Oriented Machine," *IEEE Micro*, pp. 52-69, Oct. 1987.

[5] S.S.Yau, X.Jia and D.H.Bae, "PROOF : A Parallel Object-Oriented Funtional Computational Model," *Journal of Parallel & Distributed Computing*, Vol. 12, No. 3, pp. 202-212, July 1992.

[6] E.V.Krishnamurthy, *Parallel Processing : Principles and Practice*, pp. 271-299, Addison-Wesley, 1989.

[7] M.Karaorman and J.Bruno, "Introducing Concurrency to A Sequential Language," *Communication of the ACM*, Vol. 36, No. 9, pp. 103-116, Sept. 1993.

[8] H.E.Bal, J.G.Steiner and A.S.Tanenbaum, "Programming Language for Distributed Computing Systems," *ACM Computing Surveys*, Vol. 21, No. 3, pp.2601-322, Sept. 1989.

[9] N.Carrero and D.Gelernter, "Linda in Context," *Communication of the ACM*, Vol. 32, No. 4, pp. 444-458, Apr. 1989.

[10] C.M.Chase, A.L.Cheung and M.R.Smith, "Paragon : A Parallel Programming Environment for Scientific Applications Using Communication Structures," *Journl of Parallel & Distributed Computing*, Vol. 16, No. 2, pp. 79-91, Oct. 1992.

[11] J.Galletly, *OCCAM 2*, Pitman, 1990.

[12] V.Kumar, et al., *Introduction to Parallel Computing*, pp. 525-569, Benjamin/Cummings, 1994.

[13] G. S. Almasi, *Highly Parallel Computing*, 2nd Ed., pp. 224-287, Benjamin/Cummings, 1994.

[14] M.J.Quinn, *Parallel Computing : Theory and Practice*, pp. 91-130, McGraw-Hill, 1994.

[15] K.Hwang, *Advanced Computer Architecture : Parallelism, Scalability, Programmability*, pp. 547-613, McGraw-Hill, 1993.

[16] P.America, "POOL-T : A Parallel Object-Oriented Language," *In Object-Oriented Concurrent Programming*, A.Yonezawa, N. Toroto, Eds., MIT Press, pp. 199-220, 1987.

[17] K.Clark and S.Gregory, "PARLOG : Parallel Programming in Ligic," *ACM Trans. on Programming Language System*, Vol. 8, No. 1, pp. 1-49, Jan. 1986.

[18] R.H.Halsted, "Multilisp : A Language for Concurrent Symbolic Computation," *ACM Trans. on Programming Language System*, Vol. 7, No. 4, pp. 501-538, Oct. 1985.

[19] A.Yonezawa, et al., "Object-Oriented Concurrent Programming in ABCL/1," *In the Proceeding of OOPSLA'86*, pp. 258-268, Sept. 1986.

[20] D.B.Loveman, "High Performance Fortran," *IEEE Parallel & Distributed Technology*, Vol. 1, No. 1, pp. 25-42, Feb. 1993.

[21] P. B. Hansen, "The Programming Language Concurrent Pascal," *IEEE Trans. on Software Engineering*, Vol. se-1, No. 2, pp. 199-207, 1975.

[22] A.Burns, *Concurrent Programming in Ada*, Cambridge University Press, 1986.

[23] P.J.Hatcher, at al., "Data-Parallel Programming on MIMD Computer," *IEEE Trans. on Parallel Distributed System*, Vol. 3, No. 2, pp. 377-383, July 1991.

[24] Hoare, C.A.R, "Communicating Sequential Processes," *Communication of ACM*, Vol. 21, No. 8, pp. 660-677, 1978.

[25] J.R.Allen and K. Kennedy, "Automatic Translation of Fortran Programs to Vec-

tor Form," *ACM Trans. on Programming Language and System*, pp. 491-542, Oct. 1987.
[26] R.S.Chin and S.T.Chanson, "Distributed Object-Oriented Programming System," *ACM Computing Surveys*, Vol. 23, No. 1, 91-124, Mar. 1991.

최 숙 영

1988 전북대학교 전산통계학과 학사
1991 전북대학교 전산통계학과 석사
1992~현재 충남대학교 컴퓨터과학과 박사과정
관심분야 : 병렬처리, 프로그래밍 언어, 객체지향 시스템

유 관 종

1976 서울대학교 계산통계학과 학사
1978 서울대학교 계산통계학과 전산학전공 석사
1984 서울대학교 계산통계학과 전산학전공 박사
1989~1990 캘리포니아 대학(IRVINE) 방문 교수
1979~현재 충남대학교 컴퓨터과학과 교수
관심분야 : 병렬처리, 프로그래밍 언어, 컴파일러 설계