병렬 프로그래밍 개념 LINDA의 간편화

박영환°
한성 대 학 교 컴퓨터공학과
yhpark@ice.hansung.ac.kr

Simplification of Parallel Programming Concept LINDA

Young-Hwan Park°
Dept. of Computer Engineering, Hansung University

요 약

본 논문은 병렬 프로그래밍 개념 LINDA에서 read()와 in() 프리미티브의 역전에 따른 데드락 문제를 read() 프리미티브를 제거하고 in() 프리미티브와 튜플에 계수(counter) 필드를 추가하는 간편화를 통하여 해결하는 방법에 대하여 기술한다. 기존의 LINDA 개념에서 read() 와 in() 프리미티브의 차이는 전자는 튜플을 읽기만 하고 후자는 읽은 후 그 튜플을 지운다는 점에 있다. 결국 같은 튜플에 대하여 in() 프리미티브가 먼저 실행된다면 read() 프리미티브의 서비스를 요구한 프로세스는 한없이 기다리게 되는 문제가 있다. 따라서 각 프리미티브를 사용해야 하는 시점을 사용자가 주의 깊게 결정해야 하지만 이것이 병렬 프로그램의 개발에서는 그리 쉬운 일이 아니다. 따라서 본 논문에서는 read(0 와 in() 프리미티브 2 가지를 결합하여 in() 프리미티브 한가지와 튜플에 추가된 counter 필드를 이용하여 이 문제를 해결할 수 있는 방법을 소개한다.

1. 서론

4개의 프로세서를 포함하고 있는 PPDS[1], 저가의 PC와 고성능 통신용 하드웨어를 이용한 클러스터 컴퓨터 등 병렬 컴퓨터에의 접근이 점점 더 값싸지고 용이해 지고 있다. 또한 이러한 컴퓨터에서의 프로그램 개발을 용이하게 해주는 PVM[2], MPI[3] LINDA[4] 등도 소개되거나 개발되고 있다. 이러한 새로운 툴이나 개념 중 LINDA는 병렬 프로그램 개발상의 어려움을 극복하고자 예일대(Yale University)의 David Gelernter와 그 팀에 의하여 제안된 병렬 프로그램의 한 모델이다[4]. LINDA 개념을 통하여 프로세스에서 시간적인 그리고 공간적인 문제를 분리해 냄으로써 순차적 프로그램의 개발에 들어가는 노력과 같은 정도의 노력으로 병렬 프로그램을 개발할 수 있게 되었다[4].

이러한 LINDA 개념에 기초하여 C 언어와 , Fortran 언어 등이 병렬 프로그래밍 언어로 개발되었으며[5], 커널과 결합되어 병렬 컴퓨터를 직접 사용할 수 있게 해주는 병렬 커널들도 개발되었다[6][7].

2. LINDA

LINDA 모델은 튜플(tuple)과 튜플-스페이스(tuple-space) 그리고 out(), eval(), in(), read() 라는 4개의 기본적인 프리미티브들로 이루어져 있다.

2.1 튜플과 튜플-스페이스

LINDA 개념에서 튜플은 고유의 키(key) 필드와 데이터의 집합 필드로 구성되어 있고 튜플-스페이스는 튜플들로 구성되어 있는 일종의 공용 연상 기억장치(Shared Associative Memory)로, 이 튜플-스페이스에서는 주소가 아니라 바로 각 튜플의 고유 키를 통하여 원하는 튜플을 찾을 수 있다.

이와 같은 특징은 병렬 프로그램을 개발함에 있어 필수적인 프로세스 또는 프로세서간 통신(IPC : Inter Process Communication) 을 쉽게 구현할 수 있게 해준다. 즉 2 개 이상의 프로세스 간에 정보를 주고 받아야 하는 경우, LINDA 개념을 사용하지 않으면 이 정보를 저장할 주소를 관계된 모든 프로세스가 알고 있어야 하지만, 병렬 프로그램을 실행함에 있어서 이 주소를 병렬 컴퓨터에서 유지한다는 것은 쉬운 일이 아니다. 또한 병렬 컴퓨터의 구조가 바뀌면 프로그램을 다시 개발해

야 하는 문제점이 있다. 그러나 LINDA 개념에서는 주고 받아야 할 정보를 튜플에 저장하고, 이러한 튜플은 튜플-스페이스에 저장되어 있는데, 이 튜플-스페이스는 주소를 가지고 정보를 찾는 기존의 어드레서블 메모리(addressable memory) 가 아니라 열쇠(key)를 이용하여 정보를 찾는 공용 연상기억장치 이다.

2.1 LINDA의 4 프리미티브

원하는 튜플을 쓰거나 읽기 위하여 out(), eval() 과 in(), read()라는 4개의 프리미티브가 사용된다.

out(key, data)을 통하여 하나의 튜플을 튜플-스페이스에 삽입고, 이렇게 삽입하는 과정에서 이 튜플을 읽기 위하여 in()이나 read()를 실행시킨 프로세스가 있나 검사하여 이 들을 깨운 후 프로세스 레디 큐(ready queue)에 추가 시킨다. eval(key, func(e), true)은 기본적으로 out()과 같은 일을 하나, 우선 func(e)를 실행하여 그 결과가 사실이면 그 결과를 데이터로 하여 튜플을 생성하고 이 튜플을 튜플-스페이스에 삽입시킨다.

in()과 read()는 튜플-스페이스에서 원하는 튜플을 읽는 프리미티브들이다. 이 두 프리미티브의 또 다른 공통점은 원하는 튜플을 찾을 수 없는 경우, 그 프리미티브를 실행시킨 프로세스가 수면상태에 들어가고, 그 후 원하는 튜플을 삽입하는 out() 이나 eval()에 의해서 깨어나 원하는 튜플을 읽게된다. 두 프리미티브의 차이점은 read()는 튜플을 읽은 후 그 튜플을 지우지 않고 남겨놓는 반면 in()은 원하는 튜플을 읽은 후 그 튜플을 튜플-스페이스에서 지워버린다.

2.2 LINDA의 장점

LINDA 개념에서 프로세스로부터 시간적인 그리고 공간적인 문제를 분리해 냄으로써 순차적 프로그램의 개발에 들어가는 노력과 같은 정도의 노력으로 병렬 프로그램을 개발할 수 있다고 하였는데, 이러한 특성은 바로 튜플-스페이스와 위의 4가지 프리미티브 덕분이다. 즉 병렬처리에서 꼭 필요한 프로세스간 통신을 수행하는데 있어 필요한 정보의 입,출력이 주소가 아닌 키를 매개로 이루어지므로, 여러 개의 프로세스 개발 시 데이터의 공간적인 위치 문제를 고려함이 없이 프로그램을 할 수 있다. 또한 원하는 튜플이 없으면 프로세스가 자동적으로 수면상태에 들어가고, 그 원하던 튜플이 삽입되면 또한 자동적으로 깨어나게 됨으로써 프로세스간의 동기화를 고려하지 않고 병렬 프로그램을 개발할 수 있게 된다. 결과적으로 LINDA 개념을 통하여 병렬 응용프로그램을 개발함에 있어 공간적인 그리고 시간적인 제약을 뛰어넘을 수 있게 된다.

2.3 LINDA의 문제점

LINDA개념을 사용하여 병렬 프로그램을 개발하려면 in() 프리미티브와 read() 프리미티브의 순서를 정확히 조정해야만 하는 문제점이 있다. 이 문제점은 in()과 read()의 사용 목적에서 오는 것으로, read()는 정보를 단지 읽기만 하고 지우지는 않으므로 튜플-스페이스가 가득 찰 우려가 있다. 따라서 사용하지 않는 튜플은 튜플-스페이스에서 지워버릴 필요가 있는데 이러한 일을 하는 것이 in() 이다. 즉 하나의 튜플에 대하여 read()가 먼저 모두 수행되어야 하고 마지막으로 in()을 이용하여 이 튜플을 제거해야 한다. 예로 5개의 프로세스가 A라는 튜플 정보를 필요로 한다면, 우선 4개의 프로세스가 read()를 이용하여 정보를 읽어야 하고, 마지막 프로세스가 in()을 수행하여 정보를 읽고 튜플 A 를 삭제하여야 한다. 이때 in()을 수행하는 프로세스가 빨리 실행되어 read()를 수행해야 할 프로세스보다 먼저 in()을 호출하게 되면 read()를 수행한 프로세스는 영원히 튜플 A 를 기다려야 하는 데드락에 빠지게 된다.

이러한 문제에 빠지지 않기 위해서는 사용자가 주의 깊게 A라는 튜플을 이용하는 모든 프로세스의 수와 읽는 시점, 제거 시점을 고려하여 프로그램을 개발해야 한다.

3. LINDA 개념의 간편화

기존의 LINDA에서는 3개 이상의 프로세스가 데이터를 주고받아야 할 경우, 데이터를 주는 프로세스는 out()을, 데이터를 읽어야 하는 나머지 프로세스 중 하나를 제외한 모든 프로세스는 read()를, 그리고 데이터를 읽고 튜플-스페이스에서 지워야 하는 마지막 프로세스는 in()을 해야 한다. 이 경우 read()와 in()을 실행하는 프로세스 사이에 실행 순서가 역전되면, 즉 in()이 먼저 수행되어 데이터가 없어진 후에 read()가 실행되면, 이 read()를 실행한 프로세스는 읽어야 할 데이터가 없으므로 계속 기다리게 되는 데드락에 빠지게 된다.

이러한 문제점을 해결하기 위해서 본 논문에서는 read() 프리미티브를 제거하여 기존의 LINDA를 단순화 하였다. 하지만 read()와 in() 프리미티브 각각의 역할이 있으므로, 본 커널의 in() 프리미티브는 이 두 가지 역할을 동시에 수행해야 한다. 이러한 문제는 튜플을 읽어야할 프로세스의 숫자(counter)를 미리 정의하여 out() 프리미티브의 3 번째 파라미터에 설정해 줌으로써 해결하였다. 이 counter 값은 튜플에 저장되어 매번 in() 프리미티브가 원하는 튜플을 읽을 때마다 1씩 줄여나가서 기존의 read() 프리미티브 역할을 하고 마지막 in() 프리미티브가 기존의 in() 프리미티브 역할을

하여 읽고 난 후 튜플을 지우게 된다. 따라서 in() 프리미티브를 호출하는 프로세스들의 순서는 중요하지 않으므로 결국 HLINDA가 갖고있는 read()와 in() 프리미티브의 실행 순서 역전에 따른 데드락 문제를 해결할 수 있었다.

프로세스의 숫자를 미리 정의하는 것이 HLINDA에 비하여 부가적이고 번거로운 작업처럼 보이지만, 2.3절에서 언급하였듯이 LINDA의 in() 프리미피브를 호출할 정확한 시점을 찾기 위해서는 같은 튜플의 정보를 필요로 하는 모든 프로세스를 확인해야 하며 이중 가장 늦게 실행될 프로세스를 찾아내어 이 프로세스에서만 in()을 하고 나머지 선행 프로세스에서는 read()를 실행해야 한다. 즉 프로세스의 숫자를 미리 알고 있어야 하며 또한 실행 순서에 대한 정확한 추정 정보도 알고 있어야만 데드락을 피할 수 있다. 또한 이렇게 모든 것을 잘 추정하여 in()의 위치를 결정하였다 하더라도 실제 병렬처리에 있어서는 프로세스의 실행 순서가 바뀔 가능성이 충분히 있으므로 데드락에 빠질 가능성이 존재한다. 하지만 본 논문에서 제안하는 간편화된 LINDA 에서는 이러한 여러 고려사항 중 단지 같은 튜플의 정보를 필요로 하는 프로세스의 숫자 정보만을 이용하여 이것을 out() 프리미티브의 한 파라미터로 설정하고, in()이 수행될 때마다 이 카운터를 줄여감으로써 LINDA 개념에서 read() 프리미티브를 없앨 수 있고 이에 따라 구현된 LINDA의 크기도 줄일 수 있다.

4. 간편화된 LINDA 구현

4.1 튜플

기존 LINDA 개념에서 튜플은 일반적으로 열쇠(key) 필드와 자료(data) 필드로 구성되어 있으나 간편화된 LINDA 에서는 계수(counter) 필드가 하나 더 추가되어 있다. 이 필드에는 정수값이 저장되어 있는데 이 값은 이 튜플의 자료를 읽어야 하는 in() 프리미티브의 갯수를 나타낸다.

4.2 out() 프리미티브

이 프리미티브는 기존 LINDA 개념의 out() 프리미티브의 역할과 동일하다. 즉 튜플을 튜플-스페이스에 삽입한다. 그러나 매개변수에 계수정보가 아래와 같이 하나 더 추가되어 있다.

```
out(key, data, counter);
```

이 튜플을 기다리며 수면상태에 있는 프로세스 모두를 확인하여 깨운 후 정보를 넘겨주고 그 프로세스 숫자만큼 계수값에서 빼고 그 남은 값이 1보다 크거나 같으면 튜플을 튜플-스페이스에 삽입한다.

4.3 in() 프리미티브

이 프리미티브는 아래와 같은 형식으로 호출되며

```
in(key, &data);
```

찾는 튜플이 존재하지 않으면 이 프리미티브를 호출한 프로세스는 수면상태에 들어가며, 튜플의 템플리트만 튜플-스페이스에 삽입한다. 만일 튜플이 존재하면 정보를 읽어들인 후 계수값이 1 보다 크면 이 값을 하나 감소시키고, 1 이면 이 튜플을 튜플-스페이스에서 제거한다.

5. 결론

본 논문에서는 LINDA 개념에서 같은 튜플을 읽는 in() 과 read() 프리미티브의 처리 순서 역전에 따른 데드락 문제를 해결하기 위하여 간편화된 LINDA 개념을 소개하였다. in() 과 read() 프리미티브를 하나로 통합하고, 대신 계수 필드를 튜플에 추가하는 간단한 변화를 통하여 데드락 문제가 해소될 수 있음을 보였다. 해결해야 할 점으로는 계수값을 사용자가 모든 튜플에 대하여 미리 알고 있어야 한다는 문제이다. 이러한 문제는 소스코드를 분석하여 자동으로 찾아주는 방법을 고려하고 있다.

6. 참고 문헌

[1] "Parallel Processing Development System technical reference ", Texas Instruments, 1992.

[2] http://www.epm.ornl.gov/pvm

[3] http://www.mpi-fourm.org

[4] S. Ahuja, N. Carriero, and D. Gelernter, "Linda and Friends", Computer, Vol. 19, No. 8, Aug. 1986.

[5] N. Carriero and D. Gelernter, "How to Write Parallel Programs: A guide to the Perplexed", ACM Computing Survaeys, Vol.21, No.3, Sept., 1999, pp.321-357.

[6] N. Carriero and D. Gelernter, "The S/Net's Linda Kernel", ACM TOCS, May, 1986.

[7] E. Yoa, B. Jardin, Y.H. Park, and K.M. Hou, "Real-time Multiprocessor Kernel: Hierarchical LINDA", 4th International Conference on Signal Processing Application and Technology, Santa-Clara, California, USA, October, 1993.