계층적 튜플 스페이스를 갖는 린다모델의 설계 및 사상 기법

이건영, 최민호, 원영선, 홍만표
아주대학교 정보 및 컴퓨터공학부

Design and Mapping of Hierarchical Tuple Space Linda Model

Geonyoung Lee, Minho Choi, Youngsun Weon, Manpyo Hong
Ajou University Information & Computer Engineering

요 약

범 구조적 병렬 컴퓨팅 모델인 린다 모델은 비결합성, 확장성, 부하균등, 동기화 문제 해결과 같은 많은 특성을 가지고 있는 반면, 튜플 스페이스로의 메시지 집중 현상으로 전체 시스템의 성능을 저하시키는 결정적인 문제를 안고 있다. 따라서 본 논문에서는 계층적 튜플 스페이스를 제공함으로써 기존 린다 모델의 장점을 모두 수용함과 동시에 메시지 집중 현상을 줄일 수 있는 계층적 튜플 스페이스 린다 모델을 제안한다. 또한 계층적 튜플 스페이스를 구성하기 위한 병렬 코드 생성 방법을 모색하고, 그 과정에서의 분석결과 및 프로그램 성능 분석을 통해 실제 시스템에 균등하게 사상함으로써 전체적인 프로세서의 효율적 수행을 가능하게 하는 사상 기법을 제안한다

1. 서론

고성능 컴퓨터의 개발과 병렬 프로그램 개념의 보편화에도 불구하고 병렬 컴퓨팅은 컴퓨터 산업에서 아직 주된 흐름이 되지 못하고 있다. 그 이유 중의 하나가 병렬 컴퓨터 구조의 다양성과 소프트웨어들이 특정 하드웨어 구조에 종속되는 경향을 보이기 때문이었다. 따라서 새로운 병렬 계산 모델은 병렬 컴퓨터 구조와는 독립적이고 (architecture-independent) 범 구조적인 계산 모델(UPCM:Universal Parallel Computational Model)로서의 역할을 하여 기존의 순차 컴퓨터들에서 폰노이만 계산 모델이 담당했던 역할을 병렬 컴퓨터 분야에서 수행함으로써 다양한 구조를 갖는 병렬 컴퓨터들과 병렬 프로그램들이 효율적으로 연결될 수 있도록 하는 것이어야 한다. Yale대학의 D.H.Gelernter에 의해 제시된 린다 모델은 이를 해결하기 위한 프로그래밍 모델의 한 종류로 특정 하드웨어 구조에 종속되지 않기 때문에 범 구조적 병렬 모델로서 다양한 연구가 진행되고 있다[1,2]

린다 모델은 논리적 공유메모리인 튜플 스페이스(tuple space), 튜플 스페이스에 보관되는 단위인 튜플(tuple) 그리고 튜플 스페이스를 통해 튜플들을 조작하기 위한 몇 개의 기본연산들로 구성된다. 또한 구현 방식에 따라 기존의 병렬 구조의 양극단에 있다고 생각할 수 있는 공유 메모리 구조와 메시지 전달 구조를 모두 포용할 수 있으며, 비 결합성 프로그래밍 환경 제공, 확장성, 부하균등, 동기화 문제 해결과 같은 많은 장점을 가지고 있다[1,2]. 그러나 튜플 스페이스를 소프트웨어적으로 구현하는데는 많은 문제점을 가지고 있다.

특히 프로세서 노드의 개수가 늘어남에 따라 튜플 스페이스에 대한 요구는 집중되고 이에 따른 메시지 오버헤드는 오히려 전체 시스템의 성능을 저하시키는 요인이 되고 있다.

따라서 본 논문에서는 린다 모델의 장점을 모두 수용하면서 동시에 단점을 보완한 계층적 튜플 스페이스(Hierarchical Tuple Space : HTS) 린다 모델을 제안한다. 새로 제안한 HTS 린다 모델은 기존 린다 모델의 튜플 스페이스를 지역성을 살린 여러개의 튜플 스페이스로 나누고 각각의 튜플 스페이스를 병렬로 수행하도록 함으로써 메시지의 집중된 부하를 균등하게 하여 전체적인 효율을 높일 수 있다. 또한 HTS 린다 모델을 위한 병렬코드 생성 방법을 모색하고, 실제 시스템에 균등하게 사상함으로써 전체적인 프로세서의 효율적 수행을 가능하게 하는 사상 방법을 제안한다.

본 논문은 1998년 한국과학재단 특정과제(95-0100-19-01-3)의 연구비 일부를 지원 받았음.

2. HTS 린다 모델

기존의 린다 모델을 확장한 HTS 린다 모델은 튜플, 튜플 스페이스, 튜플 매칭, 기본 연산과 같은 기본 린다의 개념을 모두 수용함과 동시에, 모듈과 태스크의 개념, 계층적 튜플 스페이스, 부가적 연산들이 추가된 형태이다.

2.1 모듈(Module)과 태스크(Task)

HTS 린다 모델에서는 하나의 독립적인 린다 프로그램을 모듈로, 하나의 모듈을 구성하는 프로세스를 태스크로 정의한다. 즉 모듈은 하나의 튜플 스페이스를 통해 여러 프로세스들이 서로 데이터를 주고 받으며 수행해 나가는 프로그램을 의미한다. 또한 하나의 모듈은 하나의 튜플 스페이스와 여러 태스크들을 가지고 문제를 해결해 나가게 된다. 물론 하나의 모듈은 다시 서브 모듈로 나누어 질 수도 있으며, 이때의 각 서브 모듈들 역시 린다 프로그램이 된다.

2.2 계층적 튜플 스페이스

린다 모델에서 프로그램을 수행하기 위해서는 반드시 가상 머신이 만들어져 있어야 한다. 이때, 머신 심에는 항상 하나의 튜플 스페이스가 존재하게 되는데 이를 초기 튜플 스페이스(Initial Tuple Space)라 부른다.

먼저, 하나의 프로그램이 실행되면 초기 튜플 스페이스를 통해서 튜플들을 주고 받으며 작업을 수행한다. 수행 중 병렬 처리 부분에 도달하면 초기 튜플 스페이스는 모듈들의 수 만큼 자식 튜플 스페이스를 생성한다. 초기 튜플 스페이스로부터 생성된 계층적 튜플 스페이스의 예를 살펴보면 그림 1과 같다.

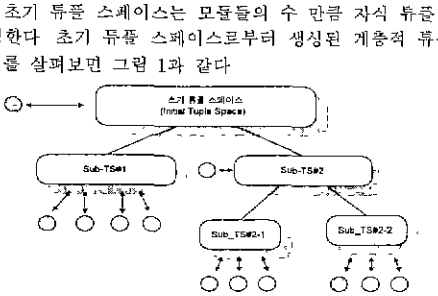


그림 1. 계층적 튜플 스페이스

이와 같이 생성된 계층적 튜플 스페이스로부터 한 모듈의 수행이 끝나면 결과 값을 부모 튜플 스페이스에게 넘겨주고 자신의 튜플 스페이스와 모든 태스크들은 종료한다. 결국 모듈들간의 동기화는 자동

으로 이루어지게 되고, 모든 프로그램이 종료되면 초기 튜플 스페이스만이 남게 되고 곧 다른 프로그램에서 사용된다.

2.3 부가적 연산

HTS 린다 모델은 그 특성 상 하나의 튜플 스페이스로 부터의 실행 결과를 부모 튜플 스페이스로 넘겨줘야 하는 작업을 필요로 하므로 부가적 연산의 추가를 요한다. 따라서, 하나의 린다 모듈에서 부모 튜플 스페이스로 결과 튜플 t를 넘겨줘야 할 경우에 사용되는 연산 out_up(t)를 추가하였다.

3. HTS 린다 모델을 위한 병렬코드 생성

HTS 린다 모델에서의 하나의 프로그램은 여러 개의 모듈로 구성되고, 각각의 모듈들은 여러 개의 태스크들로 이루어진다 이때 병렬성이 존재하는 모듈들은 전처리기(Pre-processor)에 의해 코드 생성 시 찾아지고, 각각의 모듈들은 하나의 린다 프로그램이 되어 수행된다 또한 하나의 모듈 내에서도 병렬성이 존재하는 서브 모듈이 존재한다면 병렬로 처리하도록 코드를 생성할 수 있다. 즉, HTS 린다 모델에서는 자식 튜플 스페이스를 생성하는 부분을 명시하기 위해서 미리 소스 프로그램을 처리해 데이터 종속성을 분석(data dependence analysis)하여 모듈간의 병렬성(parallelism)을 찾아내 자식 튜플 스페이스를 생성하는 코드가 삽입된 병렬코드를 생성해야 한다. 실제로 순차 프로그램이 들어오면 전체적인 의존(Dependence) 관계를 분석하여 병렬로 처리할 부분은 PAR로 묶고 순차적으로 처리할 부분은 SEQ로 묶음으로써 변형된 소스 프로그램을 생성하게 되며, 이때 새로 삽입되는 코드 PAR와 SEQ는 OCCAM에서 사용되는 명령어이다 이렇게 생성된 소스 프로그램을 컴파일 하여 HTS 린다 모델을 구현한 가상 머신에서 실행하면, PAR 명령어를 만났을 때 그 안에 있는 모듈의 개수만큼 자식 튜플 스페이스를 만들어 준다.

3.1 루프를 포함하지 않은 프로그램에 대한 병렬코드 생성

병렬코드 생성은 두 단계로 이루어진다. 첫 번째 단계에서는 전체 소스 프로그램을 매크로 명령어(macro instruction) 단위로 구분하여 매크로 명령어 사이의 데이터 종속성을 분석한 후 병렬코드를 생성한다. 여기서 매크로 명령어는 in 명령어를 기준으로 구분한 블럭을 말한다. 다음 단계에서는 하나의 매크로 명령어를 구성하고 있는 마이크로 명령어(micro instruction) 사이에서의 데이터 종속성을 분석하여 병렬코드를 생성한다 여기서 마이크로 명령어란 프로그램을 구성하는 하나의 명령어를 말한다

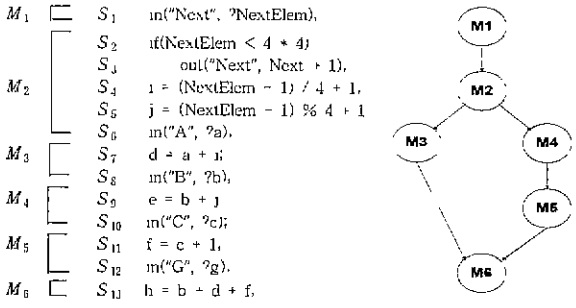
병렬 코드 생성 과정을 예를 들어 살펴보면 그림 2와 같다 먼저, 매크로 명령어 간의 데이터 종속성을 분석하기 위해 모든 매크로 명령어 간의 사용-정의 고리(use-definition chain)를 통해 데이터 종속성 그래프를 생성한다 그 결과에 의해 1단계 전처리 병렬코드를 생성한다. 다음 각 매크로 명령어 내에서 마이크로 명령어 간의 데이터 종속성을 분석한다 이때, 데이터 종속성이 존재하지 않는 명령어들은 병렬 실행이 가능하여 PAR로 묶이게 되고 그 결과에 의해 2단계 전처리 병렬코드가 생성된다

3.2 루프에 대한 병렬코드 생성

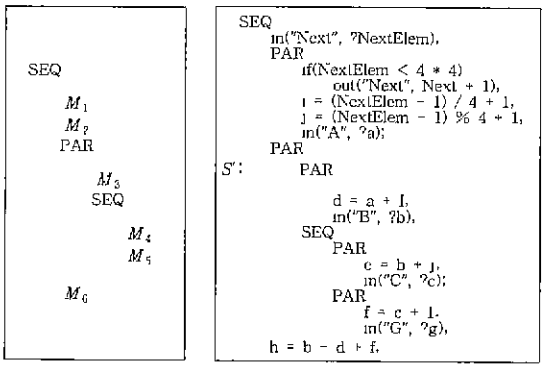
HTS 린다 모델에서는 전처리기(Pre-processor)를 통해 소스 프로그램을 분석하여 PAR와 SEQ 명령어가 추가된 소스 프로그램을 생성한다. 여기서의 PAR와 SEQ는 OCCAM 에서 사용되는 명령어이고, OCCAM에서 루프(loop)는 하나의 명령어로 간주되기 때문에 루프가 PAR에 있든 SEQ에 있든 관계없이 하나의 프로세서에 의해서 순차적으로 처리된다 그러나 루프는 프로그램을 작성하는데 있어서 많이 사용되어지는 모듈이고, 이들 루프를 병렬화한다면 전체 프로그램에 상당 부분을 차지하게 된다. 그러므로 루프에 대해 병렬성을 찾는 것은 대단히 중요한 문제이고, 루프를 병렬화시킬 수 있다면 프로그램의 실행 속도를 현저히 개선시킬 수 있을 것이다 이러한 병렬 루프를 위하여 PARLOOP 라는 명령어를 다음과 같이 정의한다 PARLOOP는 인덱스 변수를 정의할 수 있고, 인덱스 변수는 상한(upper bound)과 하한(lower bound)을 가지고 있다. 또한 PARLOOP는 ENDPAR 명령어로 범위를 나타낸다.

PARLOOP *index_variable* = *low_bound*, *upper_bound*, [step] *loop_body* ENDPAR

HTS 린다 모델에서 소스 프로그램을 전처리로 분석할 때 루프에 대해서는 특별히 PARLOOP 구문으로 변형해야 한다 PARLOOP를



(a) 린다 프로그램 예제 (b) 데이터 종속성 그래프



(c) 1단계 전처리 결과 (d) 2단계 전처리 결과

그림 2. 병렬코드 생성과정

이용하여 루프 내의 iteration을 병렬적으로 수행시키기 위해서는 루프 내의 명령어들이 서로 독립적이어야만 하는데, 만일 이들 명령어들 간에 데이터 종속성이 존재한다면 순차적 빈복 실행을 여러 프로세스에 의한 병렬 실행으로 변환하기 위해 프로세서 간에 동기화를 시켜 주기 위한 메커니즘이 포함되어져야 한다. 본 논문에서는 이러한 동기화 메커니즘을 위하여 추가적인 프로그램 언어의 확장없이 린다에서 제공하는 in과 out 오퍼레이션을 사용한다 in에 의해서 프로세스는 블록킹 될 수 있고 out은 in에 의해 블록킹된 프로세스가 재실행 되도록 할 수 있기 때문에 두 명령어를 가지고 동기화 메커니즘을 구현할 수 있다. 그림 3은 이러한 동기화 메카니즘을 고려한 병렬코드 생성 예이다.

```
DO I = 1, 100
   A(I) = B(I) * C(I) + D(I)
   B(I) = C(I) / D(I-1) + A(I-3)
   IF C(I) < 0 THEN
      C(I) = A(I) * B(I)
   ENDIF
END DO
```

```
PARLOOP I = 1, 100
   A(I) = B(I) * C(I) + D(I)
   OUT("flag" I, 1)
   IN("flag" I-3, ?tmp)
   B(I) = C(I) / D(I-1) + A(I-3)
   IF C(I) < 0 THEN
      C(I) = A(I) * B(I)
   ENDIF
END PAR
```

그림 3 루프에 대한 병렬코드 생성

이와 같이, 순차적인 루프 실행을 PARLOOP를 이용해 병렬 루프 실행으로 변형한다면 높은 성능 향상을 기대할 수 있다

4. 성능 분석을 통한 사상 기법

병렬화된 프로그램의 각 모듈을 병렬 컴퓨터에 효율적으로 사상하기 위해서는 HTS 린다 모델에서 각 레벨의 튜플 스페이스들이 동시에 생성되고 종료되어야 한다. 즉 각 모듈에서의 실행시간 예측을 통해서 프로세서 수를 적절히 할당함으로써 실제 시스템에 효율적으로 사상할 수 있다 먼저, 병렬 프로그램이 주어지면 그 프로그램을 위한 HTS 린다 모델이 생성된다. 다음 HTS 린다 모델에 의해 생성된 모듈들의 실행시간을 성능 분석기를 통해 측정한다. 이는 각각의 프로세서에게 작업을 할당함에 있어서 작업균등을 맞춤으로써 전체적인 병렬 프로그램의 처리시간을 최소화하는데 그 목적을 두고 있다.

각 모듈에서의 카리 시간이 결정되면 그 결과에 의해 프로세서 할당 트리를 생성하고, 실제 프로세서에 사상을 한다

4.1 생성된 각 모듈의 실행시간 예측

각 모듈의 실행시간을 예측하기 위한 성능 분석기는 프로파일러(profiler), 정적 성능 예측(static performance predictor), 동적 성능 예측(dynamic performance predictor)으로 구성된다.

프로파일러는 프로그램을 실행함으로써 각 문장의 실행횟수를 구하여 프로파일 자료(profile data)를 만들어 낸다. 이때, 각 문장의 실행횟수를 구하기 위해서는 반드시 프로그램 전체가 필요한 것은 아니다 즉 프로파일러의 속도향상(speed up)을 위해서 프로그램을 실행하기 위한 최소한의 문장만이 필요하게 된다. 따라서 최소한의 문장을 구하기 위해 프로그램 슬라이싱(program slicing)기법을 이용하며, 이때 HTS 린다 모델의 병렬코드를 위해 전처리기에서 생성된 의존성 분석 결과를 이용한다

정적 예측기는 프로파일 자료를 참조로 각 연산의 실행시간과 통신시간을 예측하여 기계 기술 테이블(machine description table)을 만들고 그 테이블을 이용하여 정적 성능 자료를 생성한다.

동적 예측기는 실제 기계에서 실행시켜서 성능을 예측하는 방법으로 프로파일 자료와 정적 성능 자료를 입력정보로 하여 동적 성능 자료를 생성하게 된다 따라서 동적 예측기는 프로그램의 크기가 큰 경우에는 성능을 예측하기 위해 너무 많은 시간이 소요되므로, skeleton 프로그램을 생성함으로써 이 문제를 해결하였다. 즉, skeleton 프로그램을 실행시킴으로써 원래의 프로그램을 실행 시켰을 때 얻을 수 있는 정보와 거의 유사한 정보를 얻음과 동시에 거의 20~30배 빠른 속도 향상 결과를 얻을 수 있다

skeleton 프로그램을 만드는 과정은 다음과 같다

첫째, 루프의 반복을 '1'부터 '반복횟수'로 변형한다 루프의 반복횟수는 프로파일러의 결과로 생성된 프로파일 자료를 참조한다. 둘째, 조건문이 있는 경우 조건문을 실행 횟수만큼 먼저 실행한다 조건문의 실행횟수 역시 프로파일 자료를 참조한다. 셋째, 루프와 조건문의 최대공약수를 구한 후 루프의 반복횟수와 조건문의 반복횟수를 동시에 최대공약수로 나눈다. 그런데 만일 최대공약수가 1인 경우 skeleton 프로그램은 원래의 프로그램과 동일한 실행시간을 요하게 되므로 속도 향상을 기대할 수 없다. 그래서 최대 공약수가 1인 경우의 skeleton 프로그램 생성 방법에 대한 연구가 진행 중에 있다

4.2 사상

성능 분석기에 의해 각 모듈의 실행시간이 정해지면 그 결과에 의해 각 모듈당 적절한 수의 프로세서를 할당한다

먼저, HTS 린다 모델의 성능 분석을 통해 각 노드의 실행시간이 예측되면 각각의 실행시간에 의해 프로세서 할당 트리를 생성한다

시스템의 사용 가능한 프로세서의 수를 N이라 하고, HTS_A를 계층적 튜플 스페이스, HTS_A의 레벨을 l로, 각 레벨의 노드의 집합을 $set(N_l)$, 각 레벨의 노드 $N_l(i)$의 실행시간을 $C(N_l(i))$로 나타내면, 각 레벨의 노드의 실행시간의 합은 다음과 같다

$$TC(l) = \sum_{i=1}^{N_l} C(N_l(i))$$

즉, 동일한 레벨의 노드들은 서로간에 의존성이 존재하지 않으므로 동시에 병렬적인 수행이 가능하다 따라서 각 레벨의 노드들은 시스템의 프로세서들을 실행시간에 비례하여 모두 나누어 사용할 수 있다

각 프로세서의 실행시간을 최대한 동일하게 작용함으로서 전체적인 병렬 프로그램의 실행시간을 최소로 할 수 있는 효율적인 프로세서 할당 방법은 다음과 같다

```
while HTS_A ≠ ∅ {
    for all N_l(i) in set(N_l) {
        P(N_l(i)) = [ N × C(N_l(i)) / TC(l) ]
    } end for
    HTS_A = HTS_A - set(N_l)
} end while
```

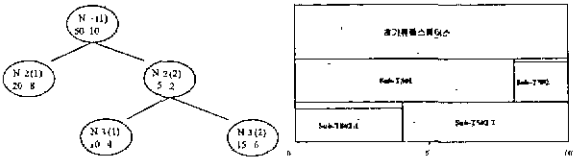
여기에서 $P(N_l(i))$는 각 노드 당 프로세서 할당 수이다

즉, 이는 병렬적으로 수행 가능한 동일한 레벨의 노드들에게 최대한의 프로세서 할당을 취하는 방법이다. 프로세서 할당 트리는 HTS 린다 모델의 실행시간 분석 결과에, 실행시간에 비례하여 할당되는 프로세서의 수를 추가한 형태의 결과를 나타낸다.

그림 4는 전체 프로세서의 수 N이 10인 경우의 프로세서 할당 방법을 보이고 있다. 먼저, 그림 1과 같은 HTS 린다 모델에서의 튜플 스페이스의 성능 분석 결과가 (a) 같을 경우 이를 토대로 (b) 같은 프로세서 할당 트리를 생성한다 (c)는 전체 프로세서에 대한 각 레벨 별 프로세서 할당 트리의 결과를 도식화한 결과이다.

	성능분석결과
초기튜플스페이스	50
Sub-TS#1	20
Sub-TS#2	5
Sub-TS#2-1	10
Sub-TS#2-2	15

(a) 튜플 스페이스의 성능 분석 결과



(b) 프로세서 할당 트리 (c) 각 레벨별 프로세서 할당 결과

그림 4 프로세서 할당 방법(N=10 인 경우)

할당트리를 생성하는 기본 규칙은 각 레벨의 노드에 할당되는 프로세서의 수를 각 노드의 실행시간에 비례하여 결정하는 것이다 따라서 각 레벨의 노드들은 최대한 동일한 실행시간을 갖는다 각 노드당 할당될 프로세서의 수가 결정되면 그 값에 의해 실제 시스템에 사상한다 이때 실제 병렬 컴퓨터 구조의 특성 통시시간 그리고 계산시간을 고려하여 결정된 프로세서의 수에 맞게 적절한 형태로 사상을 시킨다

5. 결론

린다 모델은 범 구조적 병렬 컴퓨팅 모델로서 많은 장점을 가지고 있는 반면, 튜플 스페이스를 유지하는데 필요한 메시지 오버헤드와 튜플 스페이스로의 메시지 집중현상으로 전체 시스템의 성능 저하를 가져온다.

본 논문에서는 기존의 린다 모델이 갖는 장점을 모두 수용하면서 동시에 문제점을 보완한 HTS 린다 모델을 제안한다. 제안한 HTS 린다 모델은 기존 린다 모델의 튜플 스페이스를 지역성을 살린 여러 개의 튜플 스페이스로 나누고 각각의 튜플 스페이스를 병렬로 수행하도록 하였다 따라서 메시지 집중 현상을 줄임으로써 전체 시스템의 성능을 향상시켰다 또한 HTS 린다 모델을 위한 병렬코드 생성 방법을 모색하고, 그 과정에서 발생한 분석 결과 및 프로그램 성능 분석 결과를 이용하여 실제 시스템에 균등하게 사상함으로써 전체적인 프로세서의 효율적 수행을 가능하게 하는 사상 방법을 제안하였다.

참고문헌

[1] D Gelernter. "Generative Communication in Linda", ACM Trans. on Programming Language and Systems, Vol7. No 1 Jan 1985, pp. 80-112

[2] S. Ahuja, N. Carriero, D. Gelernter, "Linda and Friends", IEEE Computer, Aug. 1986, pp. 26-34

[3] N. Carriero, D Gelernter, "Linda in Context", CACM Vol 32, No 4, Apr 1989.

[4] H. Zima, B Chapman, "Supercompilers for Parallel and Vector Computers", ACM press, 1990.

[5] Michael Wolfe, "Optimizing Supercompilers for Supercomputers", MIT press, 1990

[6] D Han, "Performance Predictor for HPF Compilers", ICPADS'97, Korea, Dec. 1997.