스레드를 이용한 함수 병렬성 추출

김현철° 이성우 류시룡 유기영
경 북 대 학 교 컴 퓨 터 공 학 과
{hckim, swlee}@purple.knu.ac.kr

Exploration of Functional Parallelism using threads

Hyun-Chul Kim° Sung-Woo Lee Shi-Ryong Ryu Kee-Young Yoo
Dept. of Computer Engineering, Kyungpook University

요 약

본 논문에서는 프로그램을 루프 구조에 근거하여 계층적으로 표현한 HTG(Hierarchical Task Graph)의 복합 노드 태스크들을 공유 메모리 다중처리기 환경에서의 효율적 수행을 위한 새로운 스케쥴링 기법을 제안한다. 단일처리기의 멀티스레드 구조를 비롯한 여러 플랫폼에 적용하기 위해 자바의 스레드를 사용하여 구현하였으며, 기존의 HTG의 함수 병렬성을 위한 비트 벡터 알고리즘과 성능을 비교 분석하였다. 실험 결과에서 보듯이, 제안된 기법이 비트 벡트 방법에 비해 수행 시간 측면에서 효율적임을 알 수 있으며 또한, 좋은 부하 균형을 유지하였다.

1. 서론

지금까지 관심을 가진 많은 연구는 병렬 루프에서의 자료 병렬성 추출에 관한 것들이다[1,2,3,4].다양한 새로운 구조의 병렬 시스템이 개발되고 서브루틴간의 종속성 분석에 관해 연구함에 따라, 루프 외에 태스크 레벨에서의 함수 병렬성을 위한 스케쥴링 기법에 관한 연구의 중요성을 인식하게 되었다[5,6,7,8].프로그램에서 함수 병렬성의 양은 자료 병렬성 보다 적지만, 고수준의 파이프라인, 멀티스레드, 슈퍼스칼라, VLIW 구조에서는 유용하다[2,6,7,8].

본 논문에서는 공유 메모리 다중처리기 환경에서 HTG의 복합 노드를 표현한 비순환 태스크 그래프(Acyclic Task Graph, 이하 ATG)에서의 함수 병렬성 추출을 위한 새로운 기법을 제안한다. 함수 병렬성의 대상이 되는 태스크는 그레인의 크기가 작아 빈번한 스케쥴링이 요구되며, 고성능의 병렬 시스템이 아닌 단일처리기의 멀티스레드 구조에서도 수행 가능하기에 제안된 기법을 스레드 레벨에서 구현하였다. 성능 평가는 HTG의 함수 병렬성을 위한 기존의 비트 벡트 알고리즘을 동일한 형태인 스레드로 구현 후, 다양한 플랫폼에서 비교 분석하였다.

2. 관련 연구

프로그램의 중간 표현인 태스크 그래프를 계층적으로 표현하는 대부분의 방법들은 인터벌을 기본으로 계층 구조를 형성하지만, HTG는 루프에 의해 계층이 정의된다[6,7,8]. HTG는 단순, 복합, 단순 루프, 복합 루프의 네 종류 노드들과 이들간의 자료와 제어 종속 관계를 표현한 간선으로 구성되는 방향성 비순환 그래프이다[6,7].그림 1은 예제 프로그램과 이에 해당하는 HTG를 나타낸 것이다. 여기서, 노드를 포함한 그림자를 가진 타원은 하나의 계층 혹은 레벨을 표현하며, 간선의 숫자는 제어 흐름에 따른 경로를 나타낸다. 그림에서 B와 C는 단순 루프, D는 복합 루프 노드이다. 그리고, 노드 1, 2, 10은 단순 노드이며, E는 서브루틴을 표현한 복합 노드이다. 이러한 HTG의 생성 방법과 병렬성 추출에 관한 연구는 Girkar[6,7]에 의해 상세히 다루어졌으며, 노드들은 성김도(granularity)의 크기에 따라 서로 다른 레벨로 구성될 수 있다.

본 논문에서 제안된 성김도가 작은 태스크의 함수 병렬성을 위한 스케쥴링 기법의 대상이 되는 ATG는 HTG의 복합 노드를 표현한 그래프이다. 기존의 연구로는 Moreira[8]가 제안한 비트 벡트 알고리즘이 있다.ATG의 동적 스케쥴링은 종속성 이론과 Girkar[6,7]의 수행 조건들에 기초를 두고 있다. 각 노드는 수행 태그를 가지며 값이 참이 될 때 그 노드는 수행을 위한 준비 상태가 된다. 노드 v_i의 수행 태그 ε는 아래와 같이 표현된다[6,7,8].

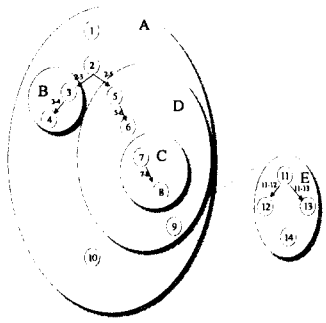
$$\varepsilon(v_i) = \varepsilon_c(v_i) \wedge \varepsilon_d(v_i)$$

여기서, $\varepsilon_c(v_i)$는 v_i의 제어 종속 조건, $\varepsilon_d(v_i)$는 자료 종속 조건의 만족 여부를 나타내며, v_i의 수행 조건은 자료 종속성이 만족되어야 하고(AND 조건) 제어 종속성 또한 참이 되어야 한다.

```
1  V1 =V2
2  if (C1.gt.100) then
3       do 4  i = 1, n
4            V3 =V3 + i
   else
5     do 9 j = 1, n
6          V4 = V4 + j
7          do 8 k = 1, l
8               V5 = V5 +k
9          T = mul(j)
10 stop

   function mul(x)
11 if (x.le.10) then
12   mul = 10
   else
13   mul = x * 10
14 return
   end
```

(a) 예제 프로그램



(b) 예제 프로그램의 HTG

3. 함수 병렬성을 위한 제안된 스케쥴링 기법

본 논문에서 제안한 함수 병렬성 추출을 위한 자동 스케쥴링 기법은 알고리즘 1과 같다. 알고리즘에서 사용되는 자료구조들은 기억 장소 크기를 줄이기 위해 필요한 정보를 비트 형태로 저장하였으며, 실제 구현은 자바의 불린 자료형을 사용하였다. 제안된 스케쥴링 기법에서 프로세서는 크게 세 가지 기능을 수행한다. 태스크를 가져오며(알고리즘 1의 입구부) 가져온 태스크의 코드를 실행한다(수행부). 그리고, 작업 완료된 노드의 후행 노드들 중에 수행 가능한 태스크를 작업 큐에 집어넣는다(출구부). 출구부는 노드 v_i의 수행 후, 경로 b가 선택되었을 때에 스케쥴링을 위해 추가된다. 이것은 크게 세 부분으로 나누어진다. 현재 수행이 끝난 노드에게만 종속적인 후행 노드들(v_j)의 실행을 위해 작업 큐에 넣는 부분(줄1-3), v_i가 작업 완료됨을 종속적 후행 노드들에게 알리는 부분(줄4-7), 그리고, v_i외에 다른 것들과도 종속 관계가 있는 후행 노드들(v_k)의 수행 가능 여부를 검사하여 조건이 만족된 태스크를 작업 큐에 넣은 부분이다(줄8-12). 줄1-3은 경로 b에 대해 후행 노드가 종속성이 하나뿐이라면, 이러한 노드들은 바로 수행 가능하기에 비교 연산(줄9)을 행하지 않고 작업 큐에 들어가게 된다. 비트 벡트 알고리즘에서는 v_i의 모든 후행 노드들의 수행가능 여부를 검사하기에 많은 비교 연산을 필요로 한다.알고리즘에서 표현 $Succ^*(v_i, b)$는 v_i에 종속적인 노드들이며, *one_successor*는 후행 노드 중에서 v_i에 대해서만 종속관계를 가지는 노드들이다. 줄4-7은 프로세서의 태스크 실행이 끝났음을 후행 노드들에게 반영하기 위해 현재 각 노드가 만족된 자료와 제어 종속성 개수를 담고있는 *DepCount*에 접근하여 스케쥴링 연산(*Fetch_Add*)에 의해 현재의 *DepCount* 값에 프로세서가 작업한 노드 v_i의 $AddBit_{v_i,b}$값을 이용하여 비트별 덧셈을 하여 갱신한다. $AddBit_{v_i,b}$는 v_i가 경로 b로 분기 할

알고리즘 1. 제안된 스케쥴링 기법

```
Entry Block :
     lock(TaskQueue)
        get the task from front of the queue
     unlock(TaskQueue)
Execute Block :
     execute the task
Exit Block :
1. for each one_successor  v_j ∈ Succ*(v_i, b) do
2.          enqueue  v_j for execution
3. end for
4. lock(DepCount)
5.     DepCount = Fetch_Add(DepCount, AddBit_{v_i,b})
6.     Temp = DepCount
7. unlock(DepCount)
8. for each successor  v_k ∈ Succ*(v_i, b) − v_j do
9.    if (Temp && MaskBit_{v_k}) == CountBit_{v_k}) then
10.         enqueue  v_k for execution
11.   end if
12. end for
```

시, 후행 노드들이 v_i의 종속성을 만족했음을 나타내는 정보를 담고 있다. 줄5에 의해 v_i의 후행 노드들의 현재 만족된 종속성 개수가 1증가된다. 이것은 후행 노드 입장에서 보면 자신이 고려해야 할 여러 종속 관계 중 하나가 만족됨을 의미하며, 모든 종속 관계가 만족 될 때 실행을 위한 준비 상태가 된다. 줄8-12는 v_i의 후행 노드들 중에 만족해야 할 종속 관계가 두 개이상인 태스크(v_k)들의 수행 가능 여부를 검사 후, 조건이 만족되면 큐에 집어넣는다. 현재 *DepCount*값을 저장한 *Temp* 변수와 후행 노드 v_k의 정보만을 추출하기 위한 마스크 값인 $MaskBit_{v_k}$를 비트별 AND 연산 한 결과와 v_k가 만족해야 할 전체 종속 개수를 비트 형태로 담고있는 $CountBit_{v_k}$와 비교하여 같으면 작업 큐에 집어넣는다.

4. 구현 및 비교 분석

여러 플랫폼에서 수행하기 위해 JDK 1.2.2를 이용하여 프로세서의 셀프 스케쥴링 기능을 스레드에서 수행 되도록 하였다. 임계 영역의 동기화 구현은 잠금(locking)을 위한 synchronized 키워드와 동기화 메소드를 이용하였다. 실험에 사용된 환경은 다음과 같다.

- 시스템A ;펜티엄 프로 200MHz,64MB
- 시스템B ;SUN Ultra 140(ultraSPARC 167MHz,132MB)
- 시스템C ;SUN Ultra Enterprise 450 (ultraSPARC-II 248MHz, 1048MB)
- 시스템D ;삼보 STD workstation 820(60MHz X 2,64MB)
- 시스템E ;Dell Precision workstation 420 (Pentium-III 733MHz X 2, 512MB-PC800)

실험에 사용된 응용 프로그램의 파라메터 값으로 작업 노드의 실행비용(e)은 10,20,50,100,500,1000,5000msec,

표1. 파라메터 값에 따른 수행시간 (msec)

n	e	t	비트	제안	n	e	t	비트	제안
시스템C					시스템A				
200	10	2	2052	2043.5	11	5	4	232	228
		8	792	778		10	4	307.5	280
		20	937.8	822.4		100	4	1256	1190
		30	787.6	589.17		500	1	5594	5592
	100	2	11074	11045			2	5594	5568
		4	5582.3	5564.3			6	5712	5646
		6	3791	3758.3	20	10	2	235	233.75
		8	2896	2865.3			4	166.25	163.75
		10	2475.7	2425.3		30	2	598.75	596.25
		100	1121.7	928			4	302.5	302.5
	1000	4	50611	50566	시스템B				
		50	6277.5	6177	11	10	4	307.5	280
		200	5573.5	5452.5		500	4	5662	5650
		1000	6435	5575.5	100	100	1	11012	11008
시스템D							2	5525.2	5511.1
100	100	2	5535.8	5533.3			4	2834.8	2805
		3	3796.3	3787.9			6	1956.6	1926.2
		5	2298	2274.9		5000	4	125335	125304
시스템E					200	10	100	765.67	626
100	100	1	10091	10086			200	1421.7	766.67
		3	3446.6	3437.3		100	4	5600.5	5592.5
		5	2045.3	2034.5			100	1278.7	889
		20	651.6	640.7			200	1683.3	1011.3
		50	853.2	760		1000	100	5794.5	5267.5
		100	1454.6	1389			200	6416.5	5613
	5000	20	30047	30039			1000	11905	6865

ATG 노드의 수(n)는 10,11,20,50,100,200개이다.그리고 그래프의 전체 간선의 수는 자료와 제어 종속 간선이 각각 10, 3개인 13개에서 최대 260개로 하였고,하나의 노드가 가지는 최대 종속 간선의 수는 세 개인 응용에 대해 실험을 하였다. 시스템 파라메터 값으로, 스레드 수(t)는 1개에서 응용에 따라 최대 200개까지 랜덤하게 생성하였으며, 사용된 프로세서 수(p)는 병렬 수행할 태스크의 성김도가 작기 때문에 최대 2개까지 하였다.

1)수행시간 ; 동적 할당기법의 스케쥴링 오버헤드를 v 포함한 수행 시간의 측정 결과는 표 1과 같다. 표의 시간 단위는 밀리초(msec)이며,각 값은 10번 수행한 결과의 평균 시간이다. 표에서 보듯이,다양한 실험 환경에서 대부분의 경우 제안된 할당 기법이 비트 벡트 알고리즘에 비해 수행 시간이 짧아 효율적임을 알 수 있다. 스레드 수를 증가하여 수행 시간을 감소시킬 수 있었지만, 그렇지 못한 경우도 있었다. 스레드가 어떻게 동작하는가는 플랫폼마다 다르고 플랫폼간의 성능 차이 때문에 스레드의 실행 속도가 다르다[9]. 따라서, 성능 향상을 위한 적정한 스레드 개수는 특정 응용과 시스템에서 대해 반복된 실험에 의해 알 수 있었다. 사용된 프로세서 수에 따른 성능 향상 또한, 시스템과 응용에 종속적임을 알 수 있었다.

태스크 수를 2배 증가함에 따라 약 2배의 수행 시간을 보였으며, 작업량의 크기, 즉 노드의 연산 시간을 변화시킨 결과,예로서,시스템 A에서 노드 수가 11개인 ATG에 대해 노드 당 연산 시간을 5,10,100, 500ms로 변화시켜 제안된 알고리즘을 이용해 실험한 결과, 수행 시간이 각각 228,280,1190,5614ms로 길어짐을 확인할 수 있었다.

2)부하균형 ; 제안된 기법과 비트 벡터 알고리즘 모두 단일처리기 시스템에서는 스레드가 동일한 개수의 태스크를 수행하였으나, 먼저 수행을 끝낸 스레드의 종료 시각과 마지막 태스크를 수행한 스레드의 종료시각간의 차. 이는 매번 실행 때마다 크게 생겨 부하 균형이 둘 다 좋지 못했다. 하지만, 다중처리기 환경에서는 두 알고리즘 모두 좋은 부하 균형을 보였다.

5. 결론

본 논문에서는 공유 메모리 다중처리기 환경에서 프로그램을 계층적으로 표현한 HTG의 함수 병렬성 추출을 위한 새로운 태스크 스케쥴링 기법을 제안하였다. 고성능의 병렬 시스템이 아닌 단일처리기의 멀티스레드를 이용하기 위해 자바 스레드로 구현하였다.

기존의 비트 벡터 알고리즘과 비교 분석한 결과, 비트 벡트 할당 기법에 비해 수행 시간을 줄여 효율적임을 보였다. 다중처리기 환경에서 좋은 부하 균형을 보였으며, 성능 향상을 위한 최적의 스레드 개수는 특정 응용과 시스템에 종속적임을 알 수 있었다.

참 고 문 헌

[1]M.Wolfe, *High Performance Compilers for Parallel Computing*, Addison-Wesley, 1996

[2]K.Kennedy,"Compiler Technology for Machine -Independent Parallel Programming," *Inter. Journal of Parallel Programming*, vol.22,no.1, pp.79-97, 1994.

[3]M.Schlansker, T.M.Conte, J.Dehnert, K.Ebcioglu, J.Z.Fang and C.L.Thompson,"Compiler for instruction -Level Parallelism," *IEEE Computer*, vol.30, no.12, pp.63-69,1997.

[4]Y.Yan,C.Jin and X. Zhang,"Adaptively Scheduling Parallel Loops in Distributed Shared-Memory Systems,"*IEEE Trans.on Parallel and Distributed Systems*, vol.8, no.1,pp.70-81,1997.

[5]E.P. Markatos and T.J. LeBlanc, "Using Processor Affinity in Loop Scheduling onShared-Memory Multiprocessors,"*IEEE Trans. on Parallel and Distributed Systems*,vol.5, no.4, pp.379-400, 1994.

[6]M.Girkar and C. D. Polychronopoulos, "Automatic Extraction of Functional Parallelism from Ordinary Programs," *IEEE Trans. on Parallel and Distributed Systems*, vol.3, no.2, pp.166-178,1992.

[7]M.Girkar and C.D.Polychronopoulos,"Extracting Task-Level parallelism," *ACM Trans. on Programming Languages and Systems*,vol.17,no.4,pp.600-634,1995.

[8]J.E.Moreira and C.D.Polychronopoulos, "Auto -scheduling in a Shared Memory Multiprocessor," Technical Report 1337, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1994.

[9]M.Campione,*The Java Tutorial*,Addison-Wesley,1999.