쓰레드를 이용한 루프 캐리 종속성을 가진 루프의 스케쥴링

김현철[U] 이종국 유기영
경 북 대 학 교 컴 퓨 터 공 학 과
{hckim, kooki}@purple.knu.ac.kr

Scheduling of loop with carried dependence using thread

Hyun-Chul Kim[U] Jong-Kook Lee Kee-Young Yoo
Dept. of Computer Engineering, Kyungpook University

요 약

루프를 병렬 처리하기 위해 공유 메모리 다중처리기에 루프를 할당하는 네 가지 기법들을 루프 캐리 종속성(loop-carried dependence)을 가진 루프의 할당에 적용하기 위해 하여 변형 후 그들의 성능을 비교 분석한다. 구현은 자바 쓰레드 환경에서 하였다. 또한, 반복들 간에 종속 관계가 생기는 루프의 효율적 수행을 위해 CDSS(Carried-Dependence Self-Scheduling) 할당 기법을 제안한다. 종속 거리, 쓰레드 수, 반복 수등을 다양하게 하여 시뮬레이션 해 본 결과 제안한 CDSS는 양호한 부하 균형을 유지하였으며 다른 기법들에 비해 루프 수행 시간을 줄여 효율적임을 알 수 있었다.

1. 서론

루프의 병렬 수행을 위해 반복들을 실행 시간에 다중 프로세서에 할당하는 루프의 동적 스케쥴링 기법은 사용된 작업 큐에 따라 중앙과 지역 작업 큐 혹은, 분산 큐 기반의 할당 기법으로 나누어지며, 전용 스케쥴러의 존재 여부에 따라 중앙 집중식과 분산 스케쥴러 혹은, 셀프 스케쥴러 기법으로 구분된다[1,2,3]. 셀프 스케쥴링은 병렬 컴파일러에 의해 생성되는 드라이브 코드(drive code)가 삽입됨으로 이루어진다. 프로세서 각각의 수행 코드 앞, 뒤 부분에 스케쥴링을 위해 추가되어진 코드 부분도 수행함으로써, 스케쥴러가 분산된 개념을 가지며 별도의 고정된 전역 스케쥴러 프로세서가 필요하지 않다. 즉, 프로세서가 코드 실행과 스케쥴링 기능을 병행하는 것이다[1,2,3].

본 논문에서는 여러 가지 셀프 스케쥴링 기법을 루프 캐리 종속성(loop- carried dependence)을 가진 루프의 할당에 적용하기 위해 그들을 변형하여 자바 쓰레드 환경에서 구현한 후 성능을 비교 분석한다. 또한 종속성이 존재하는 루프를 효율적으로 수행하기 위한 새로운 할당 기법을 제안한다.

2. 루프 할당 기법들

본 논문에서는 여러 할당 기법들 중에 Factoring, SS(Self-Scheduling), GSS(Guided Self-Scheduling), CSS(Chunk Self-Scheduling) 기법을 캐리 종속성을 가진 루프 할당에 적용한다.

SS 알고리즘은 병렬 루프의 모든 반복이 수행되어질 때까지 휴면 프로세서들이 하나의 반복만을 가져와 수행한다. 거의 완벽한 작업 균등을 유지하는 반면에, 각 프로세서가 임계 구역인 중앙 작업 큐에 반복 개수 만큼인 n번의 접근을 하기에 많은 스케쥴링 오버헤드를 발생시킨다[4]. CSS는 한 번에 k개의 반복을 가져옴으로 스케쥴링 오버헤드를 줄이지만, 부하 균형은 SS보다 좋지 못하다. 그리고, 적당한 k 값을 선택하기가 힘들다. 일반적으로, k 는 $\lceil n/p \rceil$로 결정된다[4,5,6]. GSS 할당 기법은 루프의 시작 부분에서는 큰 덩어리를 할당하여 스케쥴링 오버헤드(횟수)를 감소시키며, 루프의 끝 부분에서는 덩어리의 크기를 작게 하여 부하 균형을 좋게 한다. GSS에서의 덩어리 크기 결정은 다음과 같이 이루어진다[7].

$$R_0 = n,\ R_{i+1} = R_i - G_i,\ G_i = \lceil R_i/p \rceil$$

여기서, R_i는 i 번째 스케쥴링 단계의 남은 반복의 수, G_i는 i 번째 스케쥴링에 할당되는 덩어리의 크기이다. Factoring 기법은 반복들의 수행 시간이 다양하게 변할 수 있는 경우에 부하 불균형의 위험을 줄일 수 있다[8]. 루프 할당은 각 배치별로 이루어지며, GSS 할당 기법과 같이 남은 반복의 전부를 고려하는 것이 아니라, 그 중 절반의 부분 집합만을 프로세서의 수로 나누어서 할당한다. 각 배치는 동일한 크기의 덩어리를 p번

알고리즘 1. 제안한 CDSS 할당 기법

```
Entry Block :
    /* get_routine */
 lock(Queue)
     Temp = Queue
  unlock(Queue)
  if (Temp[head] == 1) then
     i = get_from_queue(1) /*루프의 시작 부분 할당단계 */
     Execute Block(1)
  else  if (Rest > d-1)
    start_chunk=get_from_queue(d) /* 중간 부분 할당단계*/
  else
    start_chunk = get_from_queue(Rest) /*마지막부분할당*/
  end if
```

아래와 같이 할당한다.

$$R_o = n,\ R_{i+1} = R_i - pF_i,\ \ F_i = \lceil R_i/x_i p \rceil,\ \ x_i = 2$$

여기서, F_i 는 i 번째 배치의 스케쥴링에 사용된 덩어리의 크기이다. Factoring 기법으로 스케쥴링 하기 위해 x_i를 최적의 값[8]인 2로 하여 스케쥴링한다. 이러한 많은 루프 스케쥴링 알고리즘의 성능을 좌우하는 것은 다음과 같다. 첫째는, 스케쥴링 횟수와 관계되는 루프 스케쥴링 오버헤드이며, 둘째로 부하 불균형이다. 최적의 스케쥴링 알고리즘은 위의 요소들을 모두 최소화하는 것이지만, 서로 상충 관계(trade- offs)가 있기에 두 가지 모두를 만족시키기는 불가능하다.

3. 제안한 할당 기법과 여러 기법들의 변형

3.1 제안한 CDSS 할당 기법

어떤 반복에서 사용된(read/write) 값들이 다른 반복에서 사용되어지는 반복들간에 종속 관계가 존재하는 루프의 수행 시에는 프로세서가 수행을 위해 루프 반복들을 가져와도 그 반복의 종속 관계의 만족 여부에 따라,수행 또는 대기 상태가 된다. 본 논문에서는 이러한 동기화가 필요한 루프의 효율적 수행을 새로운 할당 기법을 제안한다. 제안한 할당 기법 CDSS(Carried-Dependence Self Scheduling)는 알고리즘 1과 같다. 종속 거리를 고려하여 루프를 크게 세 단계별로 스케쥴링한다. 첫 번째 한 번의 할당이 한 개의 반복을 가지는 초기 단계와 휴면 프로세서가 종속 거리 d 개만큼의 덩어리를 가져오는 중간 단계 그리고, 남은 것이 d 보다 작을 때, 전부를 가져오는 마지막 단계로 나눌 수 있다.

3.2 기존 할당 기법의 변형

루프 캐리 종속성이 있는 루프를 스케쥴링 하기 위해서는 기존 알고리즘들의 변형이 필요하다. 각 프로세서가 수행을 위해 가져온 반복들의 종속성이 만족되어 현재 수행 가능한지 체크하는 루틴이 추가 되어야한다. 이러한 연산 결과에 따라 가져온 반복이 수행 또는 대기 상태가 된다. 그리고 만약, 수행 가능하다면 수행 후 그것에 종속적인 후행 반복을 찾아 종속성이 만족되어 현재 수행 가능함을 알리는 루틴 또한 필요하다. 지금까지 설명한 알고리즘의 변형이 필요한 부분은 제안한 CDSS 할당 기법에도 동일하게 적용된다. 알고리즘 2는 기존 알고리즘의 변형을 위해 추가되어지는 부분이다.

알고리즘 2. 할당 기법의 변형 부분

```
   /* exec_test_routine */
  for k=0, d-1
     i = start_chunk + k
     lock(CrossDep)
        Temp = CrossDep
     unlock(CrossDep)
Exec_test:
      if (Temp[i] == 1) then
        Execute Block(i)
      else
        waiting for system_defined interval
        reload Temp from CrossDep
        Exec_test
        end if
   end for

Execute Block :
    executable code

Exit Block :          /* fetch_set_routine */
   j = detect(i)      /* j = i+d */
   lock(CrossDep)
      fetch_set(j, 1)
   unlock(CrossDep)
```

각 반복들의 종속성 만족 여부에 관한 정보를 표현하는 자료구조 *CrossDep* 는 n(반복 수) 비트의 배열로, 각 원소에 저장된 값은 첨자에 해당하는 반복의 자료 종속성에 관계되는 정보이며, 루프의 i 번째 반복 L_i의 종속적 후행 반복인 L_j 의 수행 가능 여부를 결정한다. *CrossDep* 의 j 번째 비트 값이 1이면, L_j의 선행 노드인 L_i의 수행이 끝나서 L_j가 현재 수행 가능함을 의미한다. 초기 값으로, 종속 그래프에서 들어오는 종속 아크가 없는 반복들은 1로 셋(set) 되어진다. 즉, 이러한 반복들은 독립적이기에 바로 수행 할 수 있으며, 나머지는 모두 0으로 채워진다. 임계 영역에 해당하는 공유 변수 *CrossDep*와 중앙 작업 큐는 잠금(lock)으로 직렬화(serialization)를 유지한다.

5. 루프 할당 기법의 비교 분석

루프 캐리 종속성을 가진 루프를 병렬 처리하기 위해서 본 논문에서 제안한 CDSS와 여러 루프 할당 정책 중에서 중앙 작업 큐 기반인 SS, CSS, GSS, Factoring의 4개의 변형된 알고리즘들을 자바 쓰레드로 구현하여 성능을 비교 분석하였다.

CDSS기법을 제외한 나머지 알고리즘들은 다양한 종속 거리, 쓰레드 수, 루프의 반복 수에 따라 서로간의 성능의 우수성이 바뀌는 것을 알 수 있었다. 하지만, 본 논문에서 제안한 CDSS 할당 기법은 이러한 실험 환경의 변

표 1. 스케쥴링 기법별 수행 시간

d	t	N	Schedule Style				
			GSS	Fact	SS	CSS	CDSS
2	2	60	0.08	0.04	0.06	0.05	0.03
		120	0.07	0.07	0.068	0.07	0.05
		180	0.06	0.06	0.07	0.07	0.048
	3	60	0.06	0.07	0.062	0.061	0.04
		120	0.05	0.071	0.061	0.07	0.05
		180	0.07	0.07	0.074	0.068	0.05
	4	60	0.07	0.065	0.06	0.06	0.054
		120	0.06	0.067	0.07	0.076	0.04
		180	0.073	0.077	0.08	0.074	0.069
3	2	60	0.16	0.143	0.14	0.147	0.136
		120	0.17	0.24	0.238	0.249	0.178
		180	0.523	0.334	0.571	0.33	0.311
	3	60	0.104	0.116	0.121	0.11	0.107
		120	0.223	0.169	0.18	0.175	0.172
		180	0.230	0.248	0.24	0.25	0.221
	4	60	0.098	0.11	0.106	0.09	0.082
		120	0.159	0.147	0.16	0.153	0.133
		180	0.18	0.19	0.202	0.215	0.181
4	2	60	0.168	0.178	0.147	0.151	0.144
		120	0.177	0.23	0.238	0.241	0.17
		180	0.410	0.331	0.34	0.321	0.299
	3	60	0.110	0.1	0.14	0.117	0.09
		120	0.220	0.18	0.19	0.177	0.17
		180	0.238	0.24	0.253	0.25	0.22
	4	60	0.097	0.11	0.101	0.1	0.081
		120	0.17	0.161	0.155	0.145	0.14
		180	0.199	0.198	0.21	0.201	0.188

화에 상관없이 다른 기법들에 대체적으로 최소의 수행 시간을 유지하여 효율적임을 알 수 있었다. 표 1은 각 스케쥴링 기법별 수행 시간의 실험 결과이다. 실험은 SUN UltraSparc 333MHz Single CPU 시스템에서 수행하였다. 표1의 시간 단위는 초(Second)이며, 각 값은 10번 수행한 시간의 평균값이다.

병렬 처리를 위해 사용된 쓰레드 수와 수행 시간의 관계를 알아보면 다음과 같다. 병렬 처리를 위해 많은 수의 쓰레드를 사용할수록 루프 수행 시간을 줄여 효율적임을 알 수 있었다. 하지만, 그렇지 않은 경우도 있었다(SS기법).여러 할당 기법 중 SS는 다양한 실험 환경의 변화에 따라 성능의 우수성 순위가 자주 바뀌었다. 또한, 종속 거리가 클수록 본 논문에서 제안한 할당 기법이 다른 알고리즘에 비해 매우 우수하였다.

6. 결론

본 논문에서는 중앙 큐 기반의 네 가지 루프 할당 기법들을 종속성을 가진 루프의 스케쥴링에 적용하기 위해 변형 후 그들의 성능을 비교 분석하였다. 또한 종속 거리를 고려하여 스케쥴링하는 CDSS 할당 기법을 제안하였다. 종속 거리, 쓰레드 수, 반복 수등을 다양하게 하여 실험 한 결과, 제안한 할당 기법은 양호한 부하 균형을 유지하며, 전체 지연 시간을 최소화하여 다른 할당 기법에 비해 대체적으로 최소의 수행 시간을 유지하여 효율적임을 보였다.

참 고 문 헌

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

[2]M.J.Quinn, *Parallel Computing -Theory and Practice*, McGraw-Hill, 1994

[3]H.Zima and B.Chapman, *Super Compiler for Parallel and Vector Computers*, Addison- Wesley, 1991

[4]Kruskal,C.P.,and Weiss,A. "Allocating independent subtasks on parallel processors", *IEEE Trans. Software Eng.* vol.11, pp.1001-1016, 1985

[5]P.Tang and P.C.Yew, "Processor Self-Scheduling for multiple nested parallel loops," *Proc. 1986 Int. Conf. Parallel Processing*, pp.528-535, 1986

[6]Z.Fang, P.Tang, P.C.Yew, and C.Q.Zhu,"Dynamic Processor Self-Scheduling for General Parallel Nested Loops," *IEEE Trans. on Computers*, vol. 39, no.7, pp.919-929, 1990

[7]C.D.Polychronopoulos and D.Kuck, "Guided Self-Scheduling: A Practical Scheme for Parallel Supercomputers,"*IEEE Trans. on Computers*, vol.36, no.12, pp.1425-1439, 1987

[8]S.E. Hummel, E. Schonberg, and L.E. Flynn,"Factoring : A Method for Scheduling Parallel Loops,"*Comm.ACM*, vol.35,no.8, pp.90-101, 1992