다중 스레드 모델에서 스레드 우선 순위에 따른 스레드 스케쥴링 기법

이정호[U] 고훈준[U] 양창모[I] 유원희[U]
[U]인하대학교 전자계산공학과
[I]청주교육대학교 전자계산공학과
g1991274@inhavision.inha.ac.kr

The thread scheduling method based on the priority of threads on the multithread models

Jung-Ho Lee[U] Hun-Joon Kouh[U] Chang-Mo Yang[I] Weon-Hee Yoo[U]
Dept. of Computer Science, InHa University
Dept of Computer Science, ChongJu National University of Education

요 약

폰 노이만 모델의 지역성과 데이타플로우 모델의 병렬성을 결합하여 등장한 모델이 다중 스레드 모델이다. 다중스레드 모델의 목적은 통신시간과 계산 시간을 겹침으로써 프로세서의 활용도를 높이고자 하는 것이다. 기존의 대부분의 다중 스레드 모델의 스레드 스케쥴링 기법은 FIFO 혹은 FILO 방식을 사용하고 있다. 본 논문에서는 프로세서의 활용도를 높이고 프로세서의 휴지 시간을 줄이기 위해서 원격 함수 호출 혹은 원격 메모리 참조 기능의 스레드(이후로는 원격 스레드라 부름)와 계산 기능의 스레드가 동시에 활성화 되었을 때 원격 스레드들을 먼저 수행하는 것이 프로세서의 지연 시간을 줄이고 병렬성을 높이는 데 효과적임을 제안한다. 이것을 구현하기 위해서 프레임 내부의 지속 벡터(CV)를 CCV(call continuation vector)와 LCV(local continuation vector) 둘로 구분하였다. 스레드가 활성화 될 때 CCV에는 원격 스레드들을, LCV에는 계산 스레드들을 저장한 후, CCV에 저장된 스레드들을 먼저 수행하고 LCV를 나중에 수행한다.

1. 서론

폰노이만 모델에서 명령어는 메모리에 위치하고 프로세서에 의해 그 명령어가 인출되어 실행되므로 메모리와 프로세서 사이의 통신속도가 병목(bottleneck)이 되어 실행 속도의 제한을 가져온다. 이러한 문제를 극복하기 위해서 제안된 해결책들 중에는 대규모 레지스트 집합, 캐쉬 메모리, 보다 간단한 메모리 접근 명령어들이 있다. 이제는
컴퓨터의 속도가 빛의 속도에 의해 제한되므로 새로운 해결책으로 병렬 처리 기계가 대안으로 제시되었다. 첫 번째 병렬 시스템을 위한 병렬 프로세서의 설계는 폰노이만 구조에 기반하였다. 폰노이만 구조에 기반한 병렬 구조는 원격 메모리 참조시의 지연문제와 동기화 비용이 비싼 문제가 있다. 이러한 폰노이만 모델 기반의 병렬처리 시스템의 대안으로 등장한 것이 데이타플로우 모델이다. 그러나 폰노이만 모델의 병목을 제거하기 위해 등장한 데이타플로우 모델은 실행단위와 매칭단위 중 느린 쪽의 속도에 좌우된다는 점이나 토큰의 복사 명령
의 필요, 과도한 병렬성으로 인한 자원 관리의 문제 등 결국 몇 개의 자체적인 병목현상을 도입한 결과를 초래하였다. 따라서 폰노이만 모델과 데이타플로우 모델의 장점을 결합하고 단점을 보완한 모델로서 다중 스레드 모델이 등장하였다. 폰노이만 프로세서는 지역성이 컴파일 시간에 결정될 수 있고 상태가 존재하기 때문에 지역성을 이용하여 보다 빠른 수행을 할 수 있다. 데이타플로우 모델은 모든 형태의 병렬성을 추출하며 긴 지연성을 감내할 수 있다. 폰노이만 모델의 장점을 데이타플로우 모델에 확장하여 등장한 모델이 다중 스레드 모델이다. 이러한 다중 스레드 모델로는 Tera[], *T[], Monsoon[], EM-4[], TAM[] 등이 있다.

다중스레드 모델의 목적은 계산시간과 통신시간을 겹치게 함으로써 원격 메모리 참조 혹은 함수를 호출함으로 인해 발생하는 지연시간에도 프로세서를 활용하게 함으로써 프로세서 활용도를 높이고자 하는 것이다.[] 특히, 원격 메모리 참조시에는 여러 단계의 연산으로 나누는 것이 중요하다. 이것은 일반적으로 다중스레딩을 통해서 구현된다. 이것을 위해서 각 프로세서는 각각이 제어권을 가진 여러개의 스레드를 가지고 하나의 스레드에서 실행이 지연되었을 때 다른 스레드로 문맥을 전환함으로써 프로세서는 실행을 진행한다. 원격 메모리 참조가 끝나면 동기화 조건을 만족시킴으로써 다른 대기중인 스레

드를 실행 가능 상태로 만든다.

본 논문은 기존의 TAM 모델을 바탕으로 개선된 스레드 스케쥴링 기법을 적용한 다중 스레드 모델을 설계하고 구현한다.

2. TAM - 기존의 다중 스레드 모델

TAM에서 가장 큰 병렬성의 단위는 프레임에 의해 표현되며 프로세서들에 분배된다. 프레임의 보다 미세한 병렬성은 스레드에 의해 표현된다. 모든 동기화, 스케쥴링, 저장 관리는 명시적이고 컴파일러에서 통제한다. TAM은 세가지 주요 저장 구조를 필요로 한다. 코드블록, 프레임과 스트럭쳐(structure), 레지스터이다.

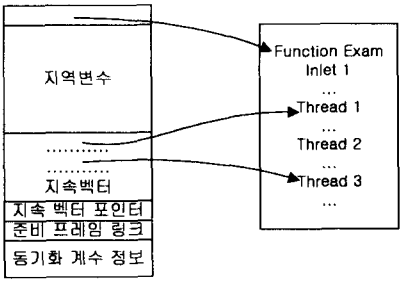
프로그램은 코드블록의 집합이며 코드블록은 함수 혹은 루프의 몸체이다. 하나의 코드블록은 스레드의 집합이며 하나의 스레드는 명령의 순차적인 집합이다. 코드블록이 호출되면 프레임이 할당되고 인자 값들은 프레임 내부의 저장위치에 기록된다. 프레임에는 연속벡터를 저장을 위한 공간이 예약되어 있으며 실행시간에 코드블록 내의 실행가능한 스레드의 시작 포인터를 저장하는데 사용된다. 스레드는 다른 스레드들을 실행 가능 상태로 만들고 메시지를 생성시킨다. 비동기적으로 메시지를 네트워크에서 받아들이면 또한 스레드를 실행 가능상태로 만든다. TAM에서 이러한 도달하는 메시지의 인터프리팅 시간을 줄이기 위해서 각 코드블록은 인렛(inlet)을 갖는다. Inlet은 코드블록의 외부적인 인터페이스이다. 인렛은 도달하는 메시지의 구성요소들을 추출하여 프레임의 슬롯에 저장하고 실행 가능해진 스레드를 CV(지속벡터)에 저장하는 일련의 명령어들이다. 프레임 사이의 통신은 메시지를 통해서 이루어진다. 메시지는 특정 코드블록의 인렛에 보내어진다. 메시지가 도달했을 때는 현재 실행중인 스레드가 인터럽트되며 인렛이 실행된다. 인렛을 실행한 후에는 인터럽트 되었던 스레드가 계속해서 실행된다. TAM에서는 인자와 결과값은 프레임 사이의 통신으로 표현된다. 호출자는 피호출자의 미리 정해진 인렛에 인자를 전달하고 피호출자는 결과값을 호출자의 미리 정해진 인렛에 전달한다.

3. 스레드 종류에 따른 우선 순위 정책

3.1. 기존의 스레드 스케쥴링 방식

TAM의 경우 스레드 제어는 FORK, SWITCH, STOP 명령어에 의해 실현된다. 동기화 스레드와 비동기화 스레드의 구분은 SYNC 라는 문장에 의해 구분되는데 이 문장은 동기화 스레드의 처음에 위치한다. SYNC 문장은 동기화 변수의 이름을 포함한다. 이 변수에는 그 스레드가 수행되기 위해서 필요한 동기화 계수의 값을 처음에 설정된다. FORK 명령이 수행될 때는 이 동기화 변수가 0인가를 체크하고 만약 아니라면 1을 감소시킨다. 비동기화 스레드에 대한 FORK 명령은 동기화 스레드에 대한 것보다 비용이 적게 든다.

SWITCH 명령은 두개의 스레드 중 하나의 스레드에 대해서 FORK를 수행하는 명령어이다. FORK 명령에 의해서 동기화가 만족되었을때 스레드는 CV에 들어간다. STOP 명령은 CV에서 다른 스레드의 IP를 뽑아내고 그 스레드로 제어를 이동시킨다. CV는 프레임 내에 포함된 작은 스택이라고 할 수 있으므로 스레드의 IP를 집어넣을 때마다 CV의 포인터를 하나씩 증가시키므로 프레임의 CV에 의한 스레드의 스케쥴링 방식은 FILO(first in last out) 방식이다.



[그림 1] 프레임의 구조

3.2. 스레드 종류에 따른 우선 순위 정책

하나의 스레드 혹은 인렛은 다른 스레드들의 동기화 계수를 감소시킴으로써 활성화 시킬 수 있다. 함수 호출이나 원격 메모리 참조 기능의 스레드와 계산 기능의 스레드가 동시에 수행 가능하게 되었을 때 원격 함수 호출이나 원격 메모리 참조 기능의 스레드가 먼저 수행되도록 함으로써 전체적인 측면에서 스레드의 활용도를 높일 수 있다. 속도 향상이 이루어지는 조건은 다음 2가지이다.

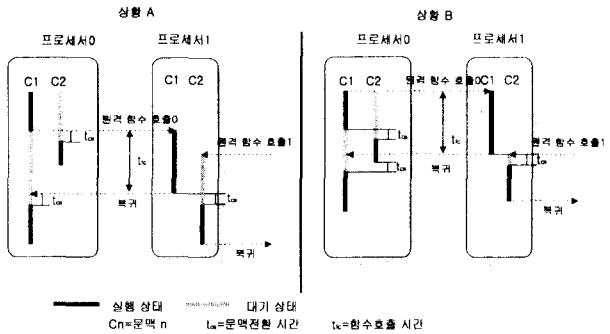
1) 원격 함수 호출 혹은 원격 메모리 참조 명령 이후 복귀 혹은 결과값 도달 시간 사이에 실행할 충분한 태스크가 없는 경우
2) 부모 프로세서로 부터 수행할 태스크가 도달하기 이전에 자식 프로세서에 실행가능한 태스크가 없었을 경우

만약 1), 2)의 경우 모두가 적용되지 않는다면 FILO 스레드 스케쥴링 방식과 동일한 수행 성능을 보인다.

본 논문에서는 위의 사항이 머신에 적용되도록 하기 위해서 CV를 CCV(call continuation vector), LCV(local continuation vector) 두 가지로 분리하였다. 원격 함수 호출이나 원격 메모리 참조 기능을 하는 스레드(이후로는 원격 스레드라 부름)의 IP(instruction pointer)를 저장

하기 CV를 CCV라 하고 그 외에 계산 기능의 스레드의 IP를 저장하기 위한 CV를 LCV라 한다. STOP 명령에 다음에 수행할 스레드를 스케쥴링 할 때는 CCV에서 먼저 스레드를 뽑아내어 IP가 가리키게 한다. 만약 CCV가 비어있는 경우에만 LCV의 스레드의 시작 주소를 뽑아내어 IP가 가리키게 만들수 있다.

이러한 스레드 스케쥴링 방식은 궁극적으로 프로세서들의 휴지 시간을 줄이는 데 효과적이다. 이러한 효과를 높이기 위해서 인렛에서 받아들인 메시지에 의해 준비 혹은 휴지 프레임의 스레드가 활성화 되었을 때 실행 프레임으로 문맥을 전환하지 않고 CCV에 저장된 원격 스레드들을 모두 수행한 후 본래의 실행 프레임으로 문맥 전환한다. 함수 호출 관련 명령어로는 FALLOC 이 있고 원격 메모리 참조 명령은 IFETCH, ITAKE 등이 있다. 스레드의 종류를 파악하기 위해서는 스레드 내에 이러한 명령어들이 있는가를 컴파일 시간에 분석하여 표기하며 실행시간에 스레드 종류를 분간하는데 사용한다.



[그림 2] 스레드 수행 순서에 따른 수행 시간 비교

프로세서의 계산시간과 대기 시간, 그리고 프로세서의 활용도의 관계를 표현하기 위해서 하나의 프로세서 내에서 $T_{compute}$를 계산시간, T_{wait}을 대기시간, T_{exec}를 수행시간, P_{util}을 병렬 시스템 전체의 프로세서의 활용도라고 한다. 그러면, $T_{exec}=T_{compute} + T_{wait}$ 이고,
$P_{util} = \Sigma T_{exec} / \Sigma T_{compute}$ 이다.

[그림 2]에서 위에서 설명한 조건 1)과 2)가 모두 만족하고 원격 스레드 1개와 계산 스레드 1개가 CV에 들어있을 경우에 어떤 스레드를 먼저 수행하느냐에 따라 수행시간이 달리지는 것을 보여 준다. 상황 A에서는 프로세서0 에서 계산 스레드를 먼저 수행하였고 상황 B에서는 원격 스레드를 먼저 수행하였다. 그랬을 때 상황 B 에서 프로세서0의 수행이 먼저 끝나는 것을 볼 수 있다. 또한 수행할 태스크를 갖고 있지 못했던 프로세서1에 보다 빨리 태스크를 던져주게 된다. 그러므로 원격 함수 호출1이 발생했을 때 보다 빨리 호출한 함수를 처리하므로 함수를 호출한 프로세서에 보다 빨리 복귀할 수 있으므로 프로세서0의 스레드 스케쥴링 순서가 제3의 프로세서에 파급효과를 미치는 것을 볼 수 있다. 이러한 파급효과가 연쇄작용을 일으키면 전체 수행 성능에 상당한 영향을 미칠 수 있다.

4. 결론 및 향후 계획

본 논문에서는 기존의 CV에 들어가는 스레드들을 FILO 방식으로 스레드 스케쥴링 하는 방법을 개선하였다. 원격 함수 호출이나 원격 메모리 참조 명령어가 포함된 스레드가 높은 우선순위를 갖고 먼저 수행되게 함으로써 프로세서의 활용도를 높이고 전체 수행 시간을 감소시킬 수 있음을 보였다. 향후 계획으로는 UNIX 환경의 실험용 시뮬레이터를 통한 구체적인 실험 데이타의 산출하고자 한다.

참고 문헌

[1] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Portfield, and B. Smith. The Tera computer system. In Int. Conf. on Supercomputing, pages 1-6. ACM Press, 1990

[2] M. J. Beckerle. An overview of the START(*T) computer system revision 2. Technical Report, Motorola Cambridge Research Center, 1992.

[3] G. M. Papadopoulos. *Implementation of a* General-Purpose Dataflow Multiprocessor. Technical Report TR-432, MIT Laboratory for Computer Science, August 1988

[4] S.Sakai. Y. Yamaguchi, K. Hiraki, Y. Kodama, and T. Yuba. An architecture of a dataflow single chip processor. In Proc. 16th Int. Symp. on Computer Architecture, pages 46-53. 1989

[5] D. E. Culler, A. Sah, K. E. Schauser, T. von Eicken, and J. Wawrzynek. Fine-grain parallelism with minimal hardware support: A compiler-controlled threaded abstract machine. In Proc. Int. Conf. on Architectural Support for Programming Languages and Operating Systems, pages 164-175, 1991