다중스레드 모델에서 최단 프레임 우선 스레드 스케줄링 알고리즘

(Shortest-Frame-First Scheduling Algorithm of Threads On Multithreaded Models)

심 우 호 † 유 원 희 †† 양 창 모 †††
(Woo Ho Sim) (Weon Hee Yoo) (Chang Mo Yang)

요 약 기존 다중스레드 모델에서의 주로 사용되는 선입선출 스케줄링 알고리즘은 실행의 지역성을 고려하지 않았기 때문에 높은 문맥전환 비용과 상대적으로 수행 시간이 짧은 프레임의 지연이 야기되어 일부 환경에서는 실행의 효율성을 떨어뜨리는 요인이 된다. 선입선출 스케줄링 알고리즘의 문제를 개선한 TAM의 퀀텀 단위 스케줄링 방법은 퀀텀 단위의 우선권을 너무 강조하므로 프로그램 실행의 병렬성을 제한시켜 프로세서의 활용도가 저하될 수 있고, 프레임 내에 있는 스레드들 간의 동기화로 인한 지연이 발생될 경우 대기 시간이 길어질 수 있다는 문제점을 가지고 있다.

위의 문제점들을 해결하기 위해 본 논문에서는 컴파일러에 의해 생성된 스레드의 크기와 동기화 정보를 이용하여 상대적으로 가장 짧은 프레임의 실행 시간을 예상하여 이를 우선적으로 처리하는 최단 프레임 우선(shortest-frame-first) 스케줄링 알고리즘을 제안한다. 다중스레드 모델은 실행의 일부분 특히 동기화 처리를 컴파일러에 의존하는 방식을 취함으로써 작업 시간에 대한 정확한 예상과 일관성을 쉽게 이용할 수 있다.

제안한 최단 프레임 우선 스케줄링 알고리즘을 선입선출 스케줄링 알고리즘과 비교한 실험 결과, 실행 시간의 평가에서는 평균 15% 정도 실행 시간을 단축시켰고 프로세서 활용도의 평가에서는 5% 정도의 성능 향상을 얻었다. 그리고 대기 시간의 평가에서는 평균 24% 정도의 대기 시간을 줄였다.

Abstract Because FIFO thread scheduling used in the existing multithreaded models does not consider locality in programs, it may result in the decrease of the performance of execution, caused by the frequent context switching overhead and delay of execution of relatively short frames. Quantum unit scheduling enhances the performance a little, but it still has the problems such as the decrease in the processor utilization and the longer delay due to its heavy dependency on the priority of the quantum units.

In this paper, we propose shortest-frame-first(SFF) thread scheduling algorithm. Our algorithm selects and schedules the frame that is expected to take the shortest execution time using thread size and synchronization information analyzed at compile-time. We can estimate the relative execution time of each frame at compile-time. Using SFF thread scheduling algorithm on the multithreaded models, we can expect the faster execution, better utilization of the processor, increased throughput and short waiting time compared to FIFO scheduling.

† 비 회 원 : 마인드웨어(주) 연구원
xobject@hyuksung.com
†† 종신회원 : 인하대학교 전자계산학과 교수
whyoo@dragon.inha.ac.kr
††† 비 회 원 : 청주교대 전산교육과 교수
cmyang@zeus.chongju-e.ac.kr
논문접수 : 1999년 3월 2일
심사완료 : 2000년 2월 23일

1. 서 론

데이타플로우 모델의 단점을 해결하기 위한 방안으로 프로그램에 존재하는 병렬성을 최대한 활용하는 데이타플로우 모델과 지역성을 이용하여 순차코드를 효율적으로 수행하는 폰 노이만 모델의 혼합형 계산 모델인 다중스레드 모델(multithreaded model)에 관한 여러 연구가 있었다[1-7].

기존 대부분의 다중스레드 모델은 선입선출(first-in, first-out: FIFO) 정책에 기반을 둔 스케줄링 알고리즘을 사용한다. 이는 데이타플로우 모델의 점화 규칙에 기인한 기본적인 구조일 뿐 아니라 스레드 실행에 대한 스케줄링 제어에 특별한 알고리즘이 필요가 없으므로 구현 비용이 낮은 이유도 있지만[8] 실제 다중스레드 모델에서 스케줄링 관한 연구는 지금까지 별로 이루어지고 있지 못한 상황이다.

선입선출 알고리즘은 간단하고 스레드 스케줄러의 구현 비용이 상대적으로 적으며 각 스레드에 대한 스케줄링의 공정성이 좋다는 장점이 있는 반면 스케줄러가 실행의 지역성을 고려하지 않아 잦은 문맥전환과 상대적으로 수행 시간이 짧은 프레임의 지연이 야기되어 일부 환경에서는 실행의 효율성을 떨어뜨리는 최악의 상황(worst case)을 초래할 수 있다. 이러한 선입선출 알고리즘의 문제점을 해결하기 위해 TAM에서 제안한 퀀텀 단위의 스케줄링 알고리즘이 있다[1, 9]. 그러나 퀀텀 단위의 스케줄링 알고리즘은 과도하게 지역성을 극대화 시키려했기 때문에 프레임 및 스레드에 대한 병렬성이 떨어질 수 있고 한 개의 활성 프레임 내에 있는 스레드들 간의 동기화로 인한 지연이 발생될 경우 대기 시간이 길어질 수 있다는 문제점을 가지고 있다.

본 논문에서는 기존 다중스레드 모델의 스케줄링 알고리즘의 문제점을 해결하기 위해 최단 프레임 우선(shortest-frame-first:SFF) 정책을 이용한 스레드 스케줄링 알고리즘을 제안한다. 최단 프레임 우선 알고리즘을 이용한 스레드 스케줄링은 컴파일 시간에 얻어지는 부가정보를 이용하여 프레임의 표면적인 수행 시간을 예상하여 실행 시간에 상대적으로 가장 짧은 프레임을 우선적으로 선택하여 실행시키는 방법을 취한다.

제안한 최단 프레임 우선 스케줄링 알고리즘의 성능 평가를 위해 선입선출 스케줄링 알고리즘과 비교한 실험에서 스레드 수행의 평균 대기 시간은 현저히 줄었고 따라서 전체적인 프로그램 실행 시간을 단축 시켰다.

2. 다중스레드 모델의 스레드 스케줄링

다중스레드 모델에서의 스레드 스케줄링에 관한 연구는 지금까지 별로 이루어지고 있지 못한 상황이다. 이는 스케줄링이 지역적으로 이루어지는 반면에 전체적인 프로그램 실행에 직접적인 영향을 끼치기 때문에 상황에 따른 적당한 스케줄링 정책을 결정하기 어렵기 때문이다. 이러한 이유로 프로그램에 대한 전체적인 이해와 실행할 때의 상황을 고려한 스케줄링이 이루어져야 한다.

최적화된 스케줄링 알고리즘은 NP-complete 문제이다. 따라서 스케줄링 알고리즘은 운영체제의 구현 연구에서 주로 경험에 근거한 스케줄링 알고리즘으로 많이 제안되었다. 다중스레드 모델에서 이런 경험적인 스케줄링 방법으로는 스레드 우선 기반 방법을 사용하는 TAM의 퀀텀 단위의 스케줄링[1, 9]과 레지스터와 캐쉬 메모리와 같은 하드웨어의 지원을 받는 EARTH[10]가 있다.

표 1 기존 다중스레드 모델의 동기화 및 스케줄링 방식분류

모델	스레드 실행 방식	동기화 방식	스케줄링 방식
Monsoon	비중단형	묵시적	선입선출
IHA	중단형	묵시적	선입선출
TAM	비중단형	명시적	퀀텀단위
P-RISC	비중단형	명시적	선입선출
J-Machine	중단형	묵시적	선입선출
*T	비중단형	명시적	선입선출
EM-X(4,5)	비중단형	묵시적	선입선출

표 1과 같이 다중스레드 모델에서의 스케줄링에 관한 연구는 지금까지 별로 이루어지고 있지 못한 상황이다. 대부분의 다중스레드 모델은 선입선출 스케줄링 알고리즘을 이용하는 이유는 데이타플로우 모델의 점화 규칙에 기인한 기본적인 구조이며 스레드 실행에 대한 스케줄링 제어가 거의 없어 구현 비용이 적기 때문이다[8]. 그러나 스케줄링에 대한 제어력이 약하기 때문에 긴급을 요하는 작업이거나 사용자와의 상호작용을 필요로 하는 작업에서 문맥 교환의 과도한 발생이 문제가 될 수 있다.

이를 개선한 방법으로 TAM에서의 퀀텀 단위 스케줄링이 있다. 퀀텀 단위의 스케줄링 알고리즘은 데이타 및 실행의 지역성을 높이는 것을 목적으로 제안된 우선순위 기반 알고리즘이다. 퀀텀이란 활성화된 프레임에서 처음 실행되는 스레드(enter thread)에서 마지막 스레드(leave thread)의 실행이 완료되는 시간을 의미한다. 즉 새로운 프레임을 활성화시켜 그 프레임 내에 있는 모든 스레드가 수행을 마칠 때까지 계속해서 스레드를 실행시키는 방법이다. 컴파일러가 POST, FORK 등의 스레드 제어 명령과 스레드의 entry count 등을 생성하여 스레드에 삽입함으로써 최소의 퀀텀을 구성한다. 따라서 컴파일러가 만들어내는 스레드의 질에 따라 스케줄링의 효율성이 좌우된다. 이 방식의 장점으로는 실행에 대한

지역성을 높였고 이로 인한 스레드 스위칭 비용을 감소하고 캐쉬의 일관성을 향상시켰다. 그러나 하나의 활성 프레임에서 생기는 동기화로 인하여 실제로 지역성이 떨어질 수 있고 그와는 반대로 너무 지역성을 고려한 나머지 먼저 실행을 요하는 스레드의 긴 지연을 피하기 힘들다.

3. 최단 프레임 우선 스케줄링 알고리즘

본 절에서는 기존 다중스레드 모델의 스케줄링 문제점을 보완한 최단 프레임 우선 스케줄링 알고리즘을 설명하고 제안한 스케줄러의 성능 평가를 위해 다중스레드 모델에서의 스케줄링 평가 기준을 재정의한다.

3.1 최단 프레임 우선 스케줄링 알고리즘의 배경

본 논문에서 제안하는 최단 프레임 우선(이후 SFF라 칭함) 스케줄링 알고리즘은 운영체제의 SJF(shortest-job-first) 스케줄링 알고리즘의 기본 개념을 이용한다. SJF 스케줄링 알고리즘은 실행 시스템에서 작업의 남은 수행 시간을 예상하여 가장 짧게 남은 시간의 작업을 선택하여 먼저 수행하는 것을 기본 개념으로 한다[11].

스케줄링 할 때 작업에 대한 정확한 수행 시간을 예상할 수 있다면 SJF는 주어진 작업에 대하여 평균 대기 시간을 줄여주는 최적화된 스케줄링 알고리즘이다. 그래서 SJF 방식의 스케줄러가 효과적으로 작동하기 위해서는 작업 시간에 대한 정확한 예상과 일관성이 필요한데 이로 인하여 실제 구현이 어렵거나 심지어 불가능할 수도 있다.

그러나 다중스레드 모델의 실행 특성 중 컴파일러가 생성하는 프레임 정보를 실행 시간에 활용하여 시스템의 실행 부담을 감소시키는 방법을 사용하므로 다중스레드 모델에서는 작업 시간에 대한 어느 정도의 예상이 가능하다. 실행 시간에 중요하게 사용되는 프레임 정보에는 스레드의 개수와 크기, 동기화 계수, 스레드의 인자 및 지역 변수의 개수를 들 수 있다.

```
fun fact(int n) returns integer
= if (n <= 1) then 1 else n * fact(n-1)
```

그림 1 fact 함수의 정의

그림 1은 팩토리얼을 구하는 함수이며 그림 2는 팩토리얼을 구하는 함수의 분할된 데이타플로우 그래프이다. 이 데이타플로우 그래프는 그림 3과 같은 스레드 코드로 변환된다.

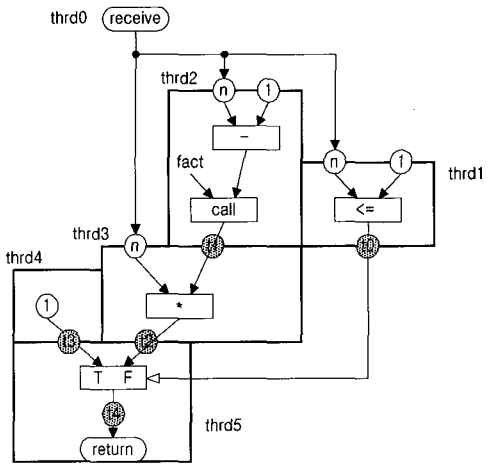

그림 2 함수 fact의 분할된 데이타플로우 그래프

그림 3에서 fact 함수는 호출에 의해 활성 프레임으로 할당되며 먼저 thrd0가 수행된다. thrd0는 fact 함수에 대한 인자를 받아들이는 역할을 하는데 인자 n값이 결정되면 이 값을 필요로 하는 thrd1, thrd2, thrd3가 실행 준비에 들어간다. thrd1에서는 n값과 1을 비교한 결과에 따라 thrd2에서 fact 함수에 대한 재귀 호출을 한다. 마지막 스레드인 thrd5는 팩토리얼을 구한 결과를 send 명령어를 통해 호출 프로그램에게 결과를 전달한다.

```
thrd0:  receive  n                          /* 인수전달
        stop
thrd1:  load     n       R0
        ltei     R0      1       R1
        store    R1      T0
        stop
thrd2:  load     n       R2
        sub      R2      1       R3
        call     fact    cfp
        send     R3      fact
        stop
thrd3:  load     n       R4
        load     T1      R5
        mul      R4      R5      R6
        store    R6      T2
        stop
thrd4:  load     1       R7
        store    R7      T3
        stop
thrd5:  load     T0      R8
        jmpt     R8      label1         /* if R8 = true jump label1
        load     T2      R9
        jmp      label2
label1: load     T3      R9
label2: store    R9      T4
        send     T4      pfp
        stop
```

그림 3 fact 함수에 대한 스레드 코드

전체 프로그램의 수행 시간은 모든 스레드의 수행이 완료되는 시간이지만 스케줄러가 사용하는 시간은 각 스레드의 첫 번째 명령어의 실행 시작부터 마지막 명령어의 실행 시작까지의 시간이다. 이러한 스레드의 실행 시간은 컴파일 시간에 예상될 수 있으므로 컴파일러가 활성 프레임의 표면적인 실행 시간을 정할 수 있다. 표면적인 실행 시간이라 한 것은 실제 프레임의 실행 시간이 아닌 컴파일러에 의해 정적으로 결정되는 프레임의 크기를 기준으로 예상되는 실행시간을 사용하기 때문이다.

3.2 SFF 스케줄링 알고리즘

SFF의 기본 알고리즘은 동기화 처리가 끝난 스레드에 대하여 컴파일러에 의해 생성된 전체 스레드의 표면 예상 실행 시간을 이용하여 스케줄링 큐에서 상대적으로 가장 짧은 프레임을 선택하여 우선적으로 실행시킨다. SFF 스케줄링 알고리즘의 수행을 위한 가정은 다음과 같다. 첫째, 각 활성 프레임 내에 있는 스레드의 동기화 처리가 완료되어야 한다. 즉 각 스레드의 동기화 계수가 0이어야 한다. 둘째, 최단 프레임으로 될 수 있는 활성 프레임은 다른 활성 프레임과의 동기화가 없어야 한다.

```
func C/B0 ( arg0 )
{
        ...
        call C/B1
        call C/B2
        ...
}

func C/B1 ( arg0, arg1 )
{
        ...
        call C/B3
        ...
        call C/B4
        ...
}

func C/B2 ( arg0 )
{
        ...
}

func C/B3 ( arg0, arg1 )
{
        ...
}

func C/B4( arg0, arg1 )
{
        ...
}
```

그림 4 프로그램 예

그림 4는 5개의 코드블록으로 이루어진 프로그램이며 각 코드블록의 관계는 그림 5와 같이 함수 호출 관계 그래프로 표현된다. 그리고 각 코드블록의 크기는 그림 6과 같다고 가정한다. 그림 4의 예제 프로그램이 실행되면, 먼저 C/B_0가 활성 프레임이 되어 실행된다. 그림 5의 함수 호출 그래프에서 보듯이 C/B_0에 의해 C/B_1와 C/B_2가 함수호출에 의해 활성 프레임이 되고 다시 C/B_1에 의해 C/B_3과 C/B_4도 마찬가지로 활성 프레임이 되어 실행된다.

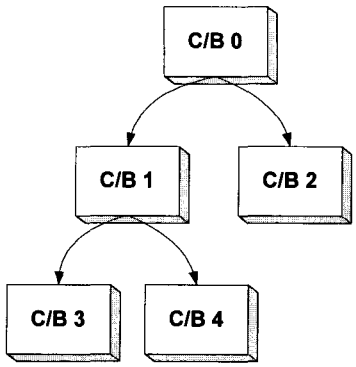

그림 5 함수 호출 그래프

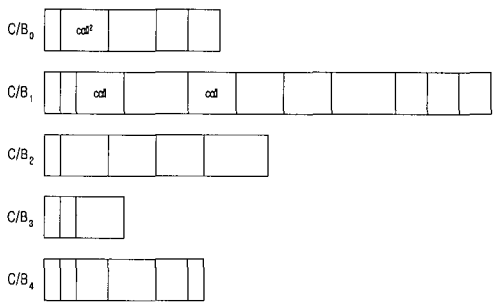

그림 6 코드블록의 크기

그림 4를 통해 제안한 SFF 스케줄링 알고리즘을 적용하면 C/B_0가 먼저 수행되어 초기 최단 프레임(AF_{sf})이 된다. 그리고 함수 호출 관계에 의해 C/B_1부터 스레드의 동기화 처리가 끝난 뒤 각 스레드의 시작주소와 활성 프레임 주소인 토큰 단위로 묶여 스케줄링 큐에 삽입된다. 이때 컴파일러에 의해 생성된 코드블록의 크기 정보를 이용하여 상대적으로 가장 짧은 활성 프레임의 토큰일 경우 큐 마지막에 삽입시키지 않고 우선 순위를 두어 큐에 삽입시킨다. 결과적으로 SFF 스케줄링 알고리즘에 의해 최단 프레임의 스레드들이 먼저 수행하게 된다.

[알고리즘 1]이 SFF 스케줄링 알고리즘이다. [알고리즘 1]에서 사용되는 기호와 의미는 다음과 같다.

- AF_m m 번째 활성 프레임

- AF_{sf} 최단 활성 프레임
- T_n n 번째 스레드
- TOK_{mn} 활성 프레임과 스레드의 주소를 갖는 토큰 즉, addr(AF_m, T_n)
- SQ 스케줄링 큐
- endp 스케줄링 큐 내의 마지막 위치
- sqp 스케줄링 큐 내의 위치
- size(AF) 활성 프레임의 크기
- pos(TOK) 스케줄링 큐 내에서의 위치

[알고리즘 1] SFF 스케줄링 알고리즘

```
입력: 초기 SQ
출력: 변화된 상태의 SQ
방법:
1 while (SQ ≠ null) do
2    TOK_mn ← get_token(SQ)
3    if (TOK_mn is first_token) then
4       new AF_sf ← AF_m            /* 최단 프레임의 초기화
5    else if (AF_sf ≠ AF_m) then
6            if (size(AF_sf) > size(AF_m)) then
7             new AF_sf ← AF_m   /* 최단 프레임의 재설정
8              new sqp ← pos(TOK_mn)  /* SQ 내의 위치
                                          재설정
9         else
10           new sqp ← endp
11        end-if
12        append TOK_mn to SQ[sqp]
13    end-if
14 end-while
```

[알고리즘 1]에서 동기화 처리가 끝난 스레드가 토큰의 형태로 스케줄링 큐에 삽입된다. 이때 스케줄링 큐에 있는 토큰의 프레임을 식별한 뒤 컴파일러에 의해 생성된 프레임의 크기 정보를 이용하여 최단 프레임을 결정한다. 만약 최단 프레임의 토큰이 아니라면, 선입선출 알고리즘에 따라 스케줄링 큐에 삽입된다.

3.3 스케줄링의 성능 평가

스케줄러의 기본적인 역할은 시스템 자원의 효율적인 할당과 교착상태(deadlock)와 같은 오류를 피하는 것이다. 시스템 자원에는 정적 자원으로 메모리, I/O 장치, 보조기억 장치가 있고 동적 자원으로 프로세서가 있다. 특히 SFF 알고리즘은 동적 자원에 속하는 프로세서의 효율적인 사용을 도모하는데 이는 활성 프레임의 평균 대기 시간을 줄이고 처리량을 높이는 데 효과적이다.

스케줄러의 성능 평가를 위하여 시뮬레이션 모델[11]을 사용하였다. 시뮬레이션은 시간에 따라 실제 구현하고자 하는 시스템의 특성을 모형화하는 일종의 프로그램으로 실제 시스템 자료 및 인위적인 자료를 입력으로 결과를 분석하고 보여준다. 일반적인 스케줄러의 성능 기준은 프로세서 활용도, 처리량, 회수 시간(turnaround time), 대기 시간 등이 있다.

다중스레드 모델에 적용된 프로세서 활용도의 의미는 하나의 프로세서에서 여러 개의 코드블록이 수행 중일 때, 전체 실행 시간에 대한 평균 계산 시간의 비율이다. 즉 단일 프로세서 환경일 때 전체 수행 시간은 다음과 같이 계산된다. 다음 식에서 T_{wait}는 대기 시간이고 $T_{computing}$은 계산 시간을 의미한다.

$$T_{exec} = T_{wait} + T_{computing}$$

그리므로 프로세서 활용도 P_{util}는 다음과 같이 계산한다.

$$P_{util} = T_{computing} / T_{exec}$$

그리고 다중 프로세서일 때는 단일 프로세서의 계산을 이용하여 다음과 같이 구할 수 있다.

$$P_{util} = \Sigma T_{computing} / \Sigma T_{exec}$$

본 논문의 다중스레드 모델에 적용된 처리량은 단위 시간에 수행이 완료된 코드블록의 개수를 말한다. 좀더 자세히 설명한다면 한 시점에서 여러 개의 활성 프레임을 하나의 집합 단위로 보아 수행이 모두 완료되는 시점에서의 코드블록의 개수를 뜻한다. 일반적인 의미에서의 처리량과 마찬가지로 다중스레드 모델에서도 이는 곧 프로그램의 실행 시간과 직접적인 관계를 가지고 있다.

본 논문의 다중스레드의 모델에 적용된 대기 시간은 활성 프레임 상태에 있는 여러 개의 코드블록들이 수행될 때 대기했던 시간의 합을 의미한다. 코드블록의 전체 실행 시간을 ΣT_{exec}, 스레드 수행 시간을 $\Sigma T_{computing}$라 할 때 코드블록에 대한 대기 시간 W는 다음 식에 의해서 계산된다.

$$W = \Sigma T_{exec} - \Sigma T_{computing}$$

4. 실 험

본 장은 제안한 SFF 스케줄링 알고리즘의 성능 평가를 위해 기존 선입선출 스케줄링 알고리즘과의 비교 실험을 한다. 퀀텀 단위 스케줄링은 선입 선출 방식이나 본 논문에서 제안한 최단 프레임 우선 방식과는 달리 컴파일러가 만들어내는 스레드의 질에 따라 스케줄링의 효율성이 좌우되므로 본 논문에서 제안한 최단 프레임

우선 방식을 퀀텀 단위 스케줄링과 비교하려면 단순히 본 연구에서 사용한 다중스레드 모델의 스케줄링 방식만을 수정하는 것으로 불충분하며 컴파일러를 재구성하여야 한다. 퀀텀 단위 스케줄링 방식의 성능에 대해서는 Culler[12]에 제시되어 있다.

우선 실험 모델과 환경을 언급하고 예제 프로그램을 통해 각 스케줄링의 성능 평가 기준에 의해 실험 및 평가한다.

4.1 실험 모델과 방법

본 실험은 유닉스 환경에서 구현된 스케줄링 큐잉 모델에 기반한 스케줄러의 시뮬레이터를 통해 이루어진다. 그림 7은 실험을 위해 구현된 다중스레드 모델에서의 스케줄링 시뮬레이션 모델이다.

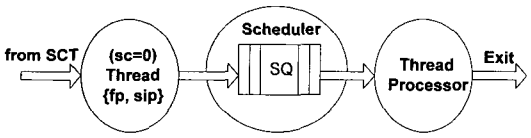

그림 7 스케줄러의 시뮬레이션 모델

표 2 예제 프로그램의 수행 특성

	s1	s2 100	fact 25	mm 20	qs 100
코드블록의 개수	5	2	1	7	8
코드블록의 크기	68	41	53	67	78
활성 프레임 개수	5	101	25	407	604
실행된 스레드 개수	20	603	125	38132	32404

표 2는 코드블록의 개수, 크기, 활성 프레임의 개수, 실행된 스레드의 개수로 각 프로그램의 특성을 나타낸다. 프로그램 s1은 두수의 합(sum)과 큰 값 구하기(max), 그리고 절대값(abs)을 구하는 함수들을 각각 호출하고, s2 100은 1부터 100까지의 합을 누적하는 프로그램이다. fact 25는 재귀 호출을 이용하여 25의 팩토리얼을 구하고, mm 20은 20 x 20의 다중 배열의 곱셈을 수행한다. 그리고 qs 100은 퀵 소트(quick sort) 알고리즘을 이용하여 100개의 난수를 정렬시키는 프로그램이다.

프로그램 실행의 실험 가정으로 첫째, 단일 프로세서에서 실행된다. 단일 프로세서 환경은 다중 프로세서의 환경과 달리 프로세간의 문맥교환 부담을 고려하지 않아 오히려 본 SFF의 성능 실험을 좀더 현실적인 환경으로 조성한다. 둘째, 레지스터의 수는 프로그램의 수행에 필요한 만큼 충분하다. 셋째, 통신 메카니즘이나 구조 메모리의 접근 비용은 고려하지 않는다.

제안한 SFF 스케줄링 알고리즘의 성능 평가 실험의 비교 대상으로 선입선출 스케줄링 알고리즘으로 한다. 실험 방법은 스케줄링의 성능 평가 요소인 실행 시간, 프로세서 활용도, 대기 시간의 관점에서 이루어진다.

4.2 실험 결과

선입선출 스케줄링 알고리즘과 SFF 스케줄링 알고리즘을 구현한 시뮬레이터의 결과는 그림 8, 그림 9, 그리고 그림 10에 있다. 각 결과 그래프는 실행 시간의 평가, 프로세서 활용도의 평가, 대기 시간의 평가를 나타낸다.

제안한 SFF 스케줄링 알고리즘을 선입선출 스케줄링 알고리즘과 비교한 실험 결과, 실행 시간의 평가에서는 평균 15% 정도 실행 시간을 단축시켰고 프로세서 활용도의 평가에서는 5% 정도의 성능 향상을 얻었다. 그리고 대기 시간의 평가에서는 평균 24% 정도의 대기 시간을 줄였다. 실험 결과에서 유추할 수 있는 사실로 SFF 스케줄링 알고리즘은 스레드 실행의 평균 대기 시간을 현저히 줄임으로써 전체적인 프로그램 실행 시간을 단축시켰다.

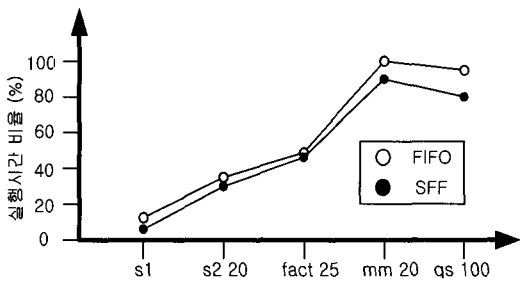

그림 8 실행 시간의 평가 결과

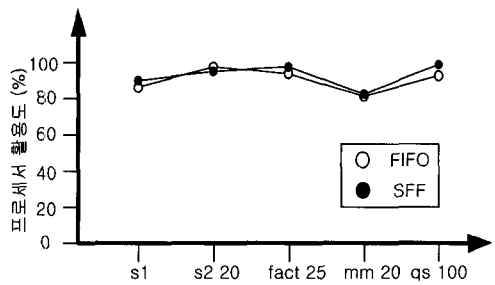

그림 9 프로세서 활용도의 평가 결과

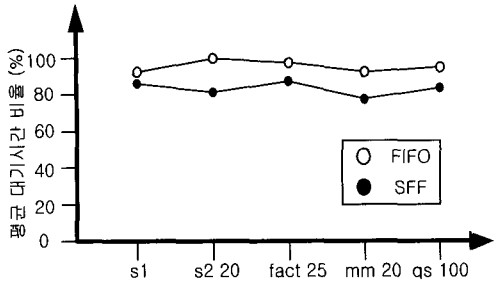

그림 10 대기시간의 평가 결과

5. 결 론

본 논문에서는 기존 다중스레드 모델에서 스케줄링 알고리즘의 문제점을 해결하기 위해 SFF 스케줄링 알고리즘을 제안하였다. SFF 알고리즘을 이용한 스레드 스케줄링은 프레임의 수행 시간을 예상하여 실행 시간에 상대적으로 가장 짧은 프레임을 우선적으로 선택하여 실행시키는 방법을 취한다. 프레임의 예상 시간은 컴파일 시간에 얻어지는 코드블록의 크기 정보를 이용하여 결정한다.

제안한 SFF 스케줄링 알고리즘의 성능 평가를 위해 선입선출 스케줄링 알고리즘과 비교한 실험에서 스레드 수행의 평균 대기 시간을 현저히 줄였고 따라서 전체적인 프로그램 실행 시간을 단축시켰다.

본 논문의 연구 결과, 다른 스케줄링 평가 요소에 비하여 향상 정도가 미약한 프로세서 활용도를 좀더 향상시키는 방안에 관한 추가 연구가 필요하다. SFF 스케줄링 방법의 우수성을 보이기 위해서는 퀀텀 단위 스케줄링과의 성능 비교가 필요하지만 이의 비교를 위해서는 본 연구에서 사용한 컴파일러의 재구성이 필요하다. 또한 다단계 조건문이나 복잡한 제어문이 있는 프로그램 환경에서 컴파일 시간에 코드블록의 크기를 정확히 예상할 수 있는 방법이 필요하다.

참 고 문 헌

[1] D. E. Culler, A. Sah, K. E. Schauser, T. Eicken and J. Wawrzynek, "Fine-grain Parallelism with Minimal Hardware Support: A Compiler-controlled Threaded Abstract Machine," *Proc. of the 4th Conf. on ASPLOS*, pp.164-175, 1990.

[2] S. H. Ha, and et al., "Design and Implementation of a Massively Parallel Multithreaded Architecture: DAVRID," *Journal of Electrical Engineering and Information Science*, Vol. 1, No. 2, pp.15-25, 1996.

[3] O. Maquelin, "The ADAM Architecture and its Simulation," *TIK-Schriftenreihe* Nr. 4, 1994.

[4] R. S. Nikhil, "Can Dataflow Subsume von Neumann Computing?," *Proc. of 16th Ann. Int'l Symp. on Computer Architecture*, pp. 262-272, 1989.

[5] R. S. Nikhil, "The Parallel Programming Language Id and it's Compilation for Parallel Machine," *LCS/CSG Memo 313*, MIT, July, 1990.

[6] R. S. Nikhil, G. M. Papadopoulos and Arvind, "*T: A multithreaded massively parallel architecture," *Proc. of the 19th Int'l Symp. on Computer Architecture*, pp. 156-167, May, 1992.

[7] G. Papadopouls and D. E. Culler, "Monsoon: an Explicit Token-Store Architecture," *Proc. of the 17th Ann. Int'l Symp. on Computer Architecture*, pp.82-91, 1990.

[8] M. D. Noakes, D. A. Wallach, and W. J. Dally, "The J-Machine Multicomputer: An architectural evaluation," *Proc. of 20th Ann. Int'l. Symp. on Computer Architecture*, 1993.

[9] D. E. Culler, S. C. Goldstein, K. E. Schauser, T. von Eicken, "TAM - A Compiler Controlled Threaded Abstract Machine," *Journal of Parallel and Distributed Computing*, Vol.18, pp.347-370, 1993.

[10] H. H. Hum and G. R. Gao, "Supporting a dynamic SPMD model in a multithread architecture," *Proc. of Compcon Spring'93*, 1993.

[11] M. Milenkovic, *Operating Systems: Concepts and Design*, pp.27-86, McGRAW-HILL Publishers, 1992.

[12] D. E. Culler, K. E. Schauser, and T. von Eicken, "Two Fundamental Limits on Dataflow Multiprocessing," Technical Report CSD-92-716, Computer Science Division, UC Berkeley, 1992.

심 우 호

1997년 2월 인하대학교 전자계산공학과 졸업(학사). 1999년 2월 인하대학교 전자계산공학과 졸업(석사). 1999년 2월 ~ 현재 마인드웨어(주) 연구원. 관심분야는 병렬처리, 병렬 객체 지향, 가상 머신, 인공지능 등임

유 원 희

1975년 서울대학교 공과대학 응용수학과 졸업. 1978년 서울대학교 대학원 계산학 전공(이학석사). 1985년 서울대학교 대학원 계산학 전공(이학박사). 1992년 ~ 1993년 University of California, Irvine 객원연구원. 1979년 ~ 현재 인하대학교 공과대학 전자계산공학과 교수. 관심분야는 프로그래밍 언어, 컴파일러, 실시간 시스템, 병렬 시스템 등임.

양 창 모

1985년 인하대학교 전자계산학과 졸업(이학사). 1988년 인하대학교 대학원 전자계산학과 졸업(이학석사). 1996년 인하대학교 대학원 전자계산공학과(공학박사). 1990년 ~ 1997년 동명전문대학 전자계산학과 부교수. 1998년 ~ 현재 청주교대 전산교육과 전임강사. 관심분야는 프로그래밍 언어(함수언어), 컴파일러, 계산모델.