미세동시다중쓰레딩을 위한 버디 할당방식의 레지스터 파일

조 석 환[†] · 이 재 도[††] · 윤 영 우[†††]

요 약

본 논문에서는 버디 크기의 레지스터 블록을 한 클럭주기만에 할당·반환하는 버디레지스터 파일의 구성을 제시한다. 버디레지스터 파일에서는 버디블럭 번호와 문맥상대 레지스터 번호를 OR연산하여 레지스터 번호가 구해지므로 가변크기 레지스터 파일처럼 레지스터 접근시간이 길어지지 않는다. 버디블럭을 할당·반환기의 구성과 논리식을 제시하고, 미세동시다중쓰레딩 프로세서에서 고정크기 레지스터 파일, 재배치 레지스터 파일 및 버디 레지스터 파일 각 방식이 성능에 미치는 영향을 분석·비교 하였다. 모의실험 결과, 버디 할당기법은 모든 실험 매개변수에 대하여 다른 방식들보다 우수한 성능을 나타내었다.

A Buddy Register File for the Fine Grain Simultaneous Multithreading

Seuk-Hwan Cho[†] · Jae-Do Lee[††] · Young-Woo Yoon[†††]

ABSTRACT

This paper presents buddy register file that can allocate and release a variable sized register block in one processor cycle. Buddy register file calculates the effective register number by ORing buddy block number with context relative register number. As a result, the access time is not so much delayed as that of variable sized register file. An organization and logical expression of the buddy register file are presented. Performance degradations due to the limits of register resources were analyzed and compared with traditional fixed sized regiser file and relocatable register file. Simulation results show that for every simulation parameter, buddy register file out performs all other register files.

1. 서 론

한 개의 프로세서 안에서 여러 개의 쓰레드와 빠른 문맥절환을 지원하는 다중쓰레딩(multithreading)은 다중프로세서의 원격메모리 접근, 동기 등과 같은 긴 잠복시간의 연산과 다른 쓰레드 안의 연산을 중첩실행함으로써 프로세서 이용율을 향상 시킨다[1][2]. 다중쓰레딩의 문맥절환단위로서 쓰레드가 절환될 때 문맥이 절환되는 coarse grain 다중쓰레딩과 명령마다 문맥이 절환되는 미세(fine grain)다중쓰레딩이 있다. 후자는 부동소수점 연산, 분기예측오류 등과 같은 비교적 짧은 잠복시간에도 다중쓰레딩의 효과가 있으므로 전자보다 성능이 우수하다. 한 개의 쓰레드 안에 있는 여러 개의 명령을 동시에 인출하여 실행하는 수퍼스칼라에 미세다중쓰레딩을 결합하면 명령간 종속문제가 대폭 완화되어 높은 성능 향상을 나타낸다. 이러

† 정 회 원:가톨릭상지전문대학 전산정보처리과 부교수
†† 정 회 원:대구보건전문대학 사무자동화과 조교수
††† 종신회원:영남대학교 공과대학 전산공학과 교수

논문접수:1997년 11월 24일, 심사완료:1998년 2월 4일

한 방식을 동시다중쓰레딩(simultaneous multithreading)이라 한다[3]. 현재의 수퍼스칼라 프로세서는 단일 쓰레드로부터 네 개까지의 명령을 동시에 인출하여 실행할 수 있으나 종속문제 등으로 인하여 최대효율이 스칼라 프로세서의 2.5배를 넘지 못한다. 집적회로의 집적도가 급속히 높아져서 수퍼스칼라 프로세서가 더욱 많은 명령을 동시에 인출·실행할 수 있어도 이러한 종속 때문에 더 이상의 성능 개선을 기대하기 힘들다. 이런 맥락에서 미세동시다중 쓰레딩이 차세대 마이크로 프로세서에 채택될 것으로 기대된다.

효율적인 다중쓰레딩을 위해서는 문맥절환이 빨라야 하며 빠른 문맥절환을 하기 위해서 문맥절환시에 목적하는 문맥이 레지스터파일 안에 있어야 한다. 여러 개의 문맥이 동시에 레지스터파일 안에 있어야 하므로 매우 큰 레지스터파일이 필요하다. 동시다중쓰레딩에서는 여러 개의 쓰레드의 흐름이 병렬로 프로세서 안에서 수행하므로 더욱 큰 레지스터파일이 필요하다.

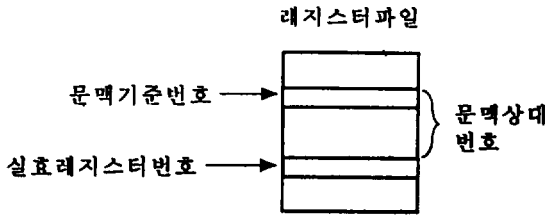
그러나 레지스터의 규모가 커지면 칩 면적을 많이 요구하는 외에 접근시간이 길어져 프로세서 클럭 주기를 제한하는 문제가 생긴다. 다른 방법으로서 문맥마다 일정한 크기의 레지스터 집합을 할당하지 않고 필요한 크기 만큼 할당하여 레지스터 파일의 이용율을 높임으로써 문제를 완화할 수도 있다[4][5][6]. 그러나 이 경우에는 프로세서가 참조하는 레지스터 번호를 계산하기 위하여 두 정수를 덧셈하여야 하는데 실행 명령마다 이러한 연산을 하여야 하므로 결국 클럭주기를 길게하는 부작용이 생긴다. 또 다른 방법으로서 버디(buddy) 메모리 할당기법을 레지스터 파일에 적용하여 2의 누승 크기(이하 버디라고 부르기로 함) 단위로 레지스터 집합을 할당하는 재배치(relocatable) 레지스터가 제안되어 있다[7][8][9][10]. 재배치 방식에서는 두 정수를 OR연산하여 레지스터 번호를 계산하므로 클럭주기에 거의 영향을 미치지 않는다. 그러나 절환하려는 문맥이 레지스터에 있는 경우에는 절환 비용이 거의 들지 않으나 그렇지 않는 경우에는 레지스터 할당, 대치 등의 작업이 필요하다. 또한 컴파일 시에 레지스터 할당·반환 루틴을 삽입하여 수행하고 할당에 실패할 경우, 성공할 때까지 순환할당방식으로 할당프로그램 수행을 반복하는데 이러한 작업이 모두 임계영역에서 이루어져야 하므로 평균 문맥절환 시간이 커지게 된다. 이런 이유로 재배치방식은 coarse grain 다중쓰레딩에 적용하는 것으로 가정하고 있다. 그러나 미세동시다중쓰레딩에서는 문맥절환 비용이 무시될 수 있을 정도로 작아야 하므로 이 방식의 적용은 곤란하다.

본 논문에서는 미세동시다중쓰레딩의 레지스터로 인한 성능저하 문제를 완화하기 위하여 한 클럭주기만에 버디를 할당·반환하는 버디 레지스터 파일을 구성하고, 고정크기 레지스터 파일 및 재배치 레지스터 파일과 시스템에 미치는 성능을 비교 분석한다. 2장에서는 버디 레지스터 파일의 원리를 소개하고 할당반환기구를 구성하며 3장에서는 모의실험 및 결과를 분석한다.

2. 레지스터 파일의 구성과 기능

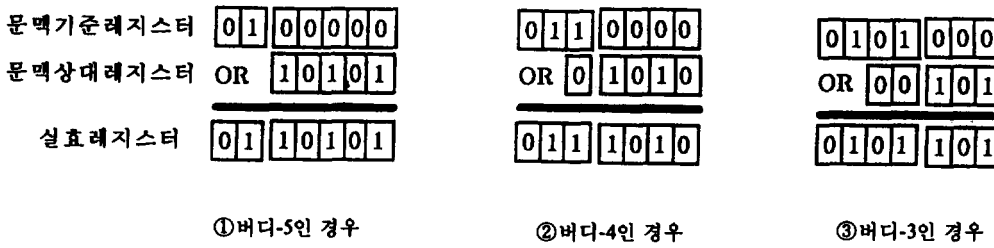
2.1 실효 레지스터

가변크기 레지스터에서 실효 레지스터는 (그림 1)과 같이 문맥의 기준 레지스터 번호와 명령의 레지스터 번호인 문맥상대 레지스터를 연산하여 결정된다.



(그림 1) 실효레지스터 번호
(Fig. 1) Number of effective register

레지스터 블럭의 크기를 할당할 문맥과 같게 하면 레지스터의 이용율은 높아지지만 문맥기준번호와 문맥상대 번호를 덧셈연산하여야 한다. 레지스터 접근시간이 프로세서 클럭주기의 임계경로인 경우 이로 인하여 클럭주기가 길어져 역효과가 생긴다. 레지스터 블럭의 크기를 2의 누승 단위로 하여 문맥을 할당하면 문맥기준 번호와 문맥상대 번호를 OR연산하면 되므로 레지스터 접근시간에 거의 영향을 미치지 않는다. 이것은 버디 메모리 할당방식을 레지스터 할당에 적용한 것이다. (그림 2)는 128개 레지스터로 구성된 레지스터 파일의 예이다. 2^k 크기의 레지스터 블럭



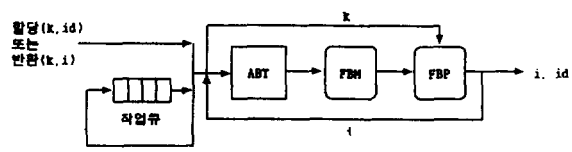
(그림 2) 버디할당 방식의 실효레지스터 번호 계산
(Fig. 2) Calculation of effective register numbers in buddy allocating

을 버디-k라 하고 문맥상대 번호는 5비트로 구성된다고 가정하자. 버디-5인 경우, 문맥기준번호의 하위 5비트가 '0'이 되므로 실효 레지스터의 상위 2비트와 하위 5비트는 각각 문맥기준번호의 상위 2비트와 문맥상대 번호의 5비트로써 구성된다. 버디-4인 경우, 문맥기준번호의 하위 4비트가 '0'이 되므로 실효 레지스터의 상위 3비트와 하위 4비트는 각각 문맥기준번호의 상위 1비트와 문맥상대 번호의 4비트로써 구성된다. 버디-3인 경우에는 문맥기준번호의 하위 3비트가 '0'이 되므로 실효 레지스터의 상위 4비트와 하위 3비트는 각각 문맥기준번호의 상위 4비트와 문맥상대 번호의 3비트로써 구성된다. 따라서 실효 레지스터는 문맥기준번호와 문맥상대 번호를 OR연산하여 결정된다.

2.2 블럭 할당·반환

함수가 인용될 때 문맥이 할당되고 복귀될 때 반환된다. 함수의 복귀 순서는 호출의 역순이므로 여러 개의 문맥을 저장하는 레지스터는 스택으로 구성된다. 다중쓰레드 컴퓨터가 쓰레드를 생성할 때에도 문맥이 할당되어 쓰레드가 소멸될 때 반환된다. 쓰레드가 소멸되는 순서는 생성과 직접적인 관계가 없으므로 여러 개의 쓰레드 문맥을 저장한 레지스터는 무순서적으로 할당·반환될 수 있어야 한다. 프로세서가 쓰레드를 수행하던 중 어떤 사건을 기다리게 되고 이때 다른 쓰레드가 레지스터 할당을 기다리고 있는 상황에서는 레지스터를 양도하면 이용율을 높힐 수 있다. 서로 종속관계가 없는 쓰레드들을 병렬로 수행하므로 레지스터 양도는 무순서로 발생하고 따라서 레지스터 블럭도 무순서로 할당·반환하게 된다.

문맥의 블럭 할당기구는 (그림 3)과 같이 구성된다. ABT(active block table)는 각 버디별로 할당한 블럭을 저장하는 레지스터이며, FBM(free block map)은 ABT로부터 할당가능한 블럭을 탐색하는 조합회로이고, FBP(free block pointer)는 그중 하나의 블럭을 선택하는 조합회로이다. 새로 들어온 할당·반환요구는 작업큐에 들어 있는 작업보다 항상 우선적으로 선택된다. 버디-k의 할당을 요구하는 작업이 성공하면 FBP로부터 인덱스-i와 쓰레드-id가 출력되고 ABT의 해당 위치에 할당되었음을 기록한다. 할당에 실패한 경우에는 작업큐에 다시 들어가서 다음 차례를 기다린다. (버디-k, 인덱스-i)의 반환을 하는 작업인 경우에는 ABT의 해당 위치에 반환되었음을 기록한다. 반환작업은 항상 성공하므로 작업큐에는 할당요구만 기록된다.

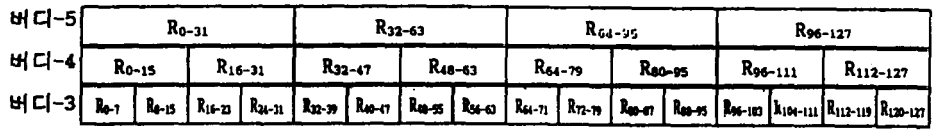


단, ABT(active block table) k : 버디 레벨
FBM(free block map) i : 인덱스
FBP(free block pointer) id : 쓰레드 번호

(그림 3) 블럭할당 기구의 구성
(Fig. 3) Organization of block allocation mechanism

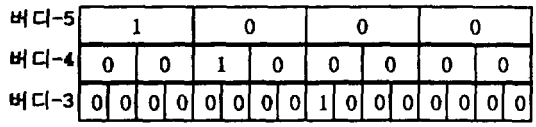
2.2.1 ABT

ABT는 각 버디별로 한개씩의 레지스터로 구성된다. 예를 들면, 버디-5는 4비트, 버디-4는 8비트, 버디-3



(그림 4) ABT의 각 비트가 나타내는 레지스터 집합
(Fig. 4) Bit representation of register set in ABT

은 16비트로 구성되고 각 비트는 (그림 4)와 같이 배정된 레지스터 집합이 할당되어 있는지를 나타낸다. 레지스터 $R_{0\text{-}31}$에 버디-5의 문맥을 할당하고, $R_{32\text{-}47}$에 버디-4를, $R_{64\text{-}71}$에 버디-3을 할당한 경우 ABT의 상태는 (그림 5)와 같다. 할당된 집합은 논리 '1'로 표시된다.



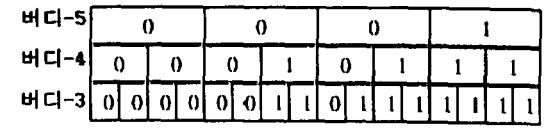
(그림 5) ABT의 할당 예
(Fig. 5) An example of allocation in ABT

본 논문에서는 레지스터의 총수를 128개로 한다. 64개인 경우, 다중 쓰레딩의 효과가 떨어지고, 256인 경우에는 레지스터 접근시간이 프로세서 주기보다 길어질 가능성이 생긴다.

2.2.2 FBM

FBM에서는 ABT의 값으로부터 각 버디의 블럭마다 할당가능 여부를 연산한다. 할당가능(free)한 블럭을 논리 '1'로 표시한다. 어떤 버디의 어떤 블럭이 할당되면 그 母블럭과 子블럭은 할당할 수 없게 된다. 위의 예에서 FBM의 출력은 (그림 6)과 같다. ABT(5, 0)이 '1'이므로 FBM(5, 0) 자신과 그 자블럭인 FBM(4, 0), FBM(4, 1), FBM(3, 0)~FBM(3, 3)이 모두 '0'이 된다. ABT(4, 2)가 '1'이므로 FBM(4, 2) 자신과 모블럭인 FBM(5, 1), 자블럭인 FBM(3, 4), FBM(3, 5)가 모두 '0'이 된다. ABT(3, 8)이 '1'이므로 FBM(3, 8) 자신과 그 모블럭인 FBM(4, 4), FBM(5, 2)가 모두 '0'이 된다.

ABT의 버디-k, i행에 기억된 논리값을 T(k, i)라 하고, FBM의 해당 위치의 논리출력을 M(k, i)라 하자. M(k, i)은 버디-k의 i번째 블록이 할당가능함을 나타내는데 그 논리식은 식(1)과 같다. 식(1)에서 첫째 항은



(그림 6) FBM의 예
(Fig. 6) An example of FBM

ABT의 k행중에 이미 할당된 블록이 있는 경우를 나타내고, 둘째 항은 ABT에서 k보다 큰 버디의 블록중에 할당이 된 경우를 나타내며, 셋째 항은 ABT에서 k보다 작은 버디의 블록중에 할당된 것이 있는 경우를 나타낸다. ⊥(bottom)은 버디중 가장 낮은 레벨을 가리키고 ⊤(top)은 버디중 가장 높은 레벨을 가리킨다.

$$\neg M(k,i) = T(k,i) + \sum_{j=1}^{\top-k} T\left(k+j, \frac{i}{2^j}\right) + \sum_{j=1}^{k-\perp} \sum_{h=0}^{2^j-1} T(k-j, i\times 2^j + h) \quad (1)$$

단, $k = \perp \sim \top$, $i = 0 \sim 2^{7-k}-1$

2.2.3 FBP

FBP는 FBM의 출력으로부터 해당 버디의 할당 가능한 블럭중 하나를 선택한다. 위의 예에서 만일 버디-4의 버디를 하나 요구하면 인덱스 '3'을 출력한다. 버디-4 안에 '1'인 블럭이 하나도 없으면 'fail' 신호를 출력한다. 메모리 버디할당 기법에서 할당을 요구한 버디 레벨에 두개 이상의 블럭이 할당가능한 경우, 블럭 선택에는 여러가지 전략이 있다. 본 연구는 레지스터 블럭 할당이 프로세서 주기 단위로 이루어 져야 하므로 하드웨어 복잡도(시간 및 면적)의 간결성에 비중을 두어 할당가능한 버디블럭중 번호가 가장 높은 블럭을 선택한다. FBP의 버디-k에서 선택된 블럭 i의 이진코드를 $I_k^6 I_k^5 I_k^4 I_k^3 I_k^2 I_k^1 I_k^0$라 하자. 비트 I_k^j의 값을 결정하는 논리식은 식(2)와 같다. B_k^i는 어떤 수

h의 이진 코드인 $B_k^6B_k^5B_k^4B_k^3B_k^2B_k^1B_k^0$의 j번째 비트의 값을 나타내며, $k=\perp\sim\top$이다.

$$I_k^j=\sum_{h=0}^{2^{7-k}-1} B_h^j\cdot M(k,j), \qquad j=k\sim 6$$
$$I_k^j=0, \qquad j=0\sim k-1 \tag{2}$$

버디-k에서 할당가능한 블럭이 한개도 없는 경우, $I_k^6=I_k^5=I_k^4=I_k^3=I_k^2=I_k^1=I_k^0=0$이 되어 블럭 0의 할당과 같은 결과로 나타난다. 따라서 식(3)과 같이 블럭 0이 할당가능하지 않은데도 불구하고 $I_k^6=I_k^5=I_k^4=I_k^3=I_k^2=I_k^1=I_k^0=0$이 될 경우 할당에 실패한 것으로 판단하고 논리값 F_k(fail)를 '1'로 한다.

$$\neg F_k=M(k,0)+\sum_{j=0}^{6} I_k^j \tag{3}$$

버디-k 인 블럭의 할당요구가 있으면, FBP는 F_k와 $I_k^6\sim I_k^0$ 값을 출력한다. $F_k=1$이면 할당에 실패한 경우이므로 k값이 대기행렬에 들어간다.

블럭을 반환하는 경우에는 ABT의 해당 비트를 논리 '0'으로 한다. FBM과 FBP는 조합회로이므로 ABT에 기록된 값이 갱신되면 따라서 변화하게 된다.

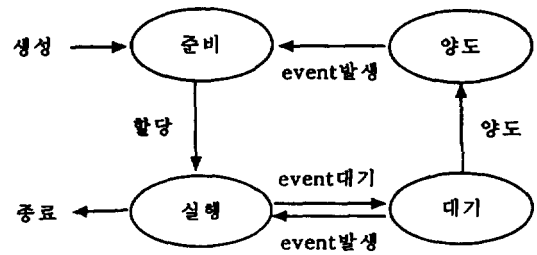
2.2.4 할당·해제 논리기의 수행시간

블록이 새로 할당되거나 해제되면 ABT에 기록되고, ABT의 출력이 FBM과 FBP를 전파하여 선택된 블록 i의 값을 출력한다. 논리 게이트의 신호전파지연시간을 t_g, 레지스터의 래치시간을 t_r이라고 하자. 식(1)에서 FBM은 NOR 논리만으로 구성되므로 FBM내의 신호전파지연시간은 t_g가 된다. 식(2)에서 $B_k^j\cdot M(k,j)$ 논리곱은 설계할 때에 결정되므로 I_k^j을 결정하는데 걸리는 시간은 t_g가 된다. 식(3)의 F_k도 NOR 논리만으로 구성되므로 논리값 결정시간이 t_g가 되는데, I_k^j와 F_k는 서로 나란히 수행되므로 두 신호 모두를 결정하는데 걸리는 시간은 t_g이다. 여러 개의 I_k^j 중에서 요구하는 버디의 인덱스를 출력하는 멀티플렉싱 논리는 $2t_g$가 걸리므로 FBP논리는 총 $3t_g$가 걸린다. 따라서, 블록 할당요구로부터 블록선택까지 걸리는 시간은 총 t_r+4t_g이다. 해제의 경우에도 같은 논리이므로 같은 시간이 걸린다.

2.3 쓰레드의 상태변환

함수가 인용되거나 쓰레드가 fork될 때 새로운 레지스터 블럭이 할당된다. 쓰레드는 fork(쓰레드-id, 버디-k)명령을 실행하여 사용가능 레지스터 블록을 할당한 다음 새로운 쓰레드를 생성시키고 함수를 인용할 때에는 call(함수-id, 버디-k)를 실행한다. 번역할 때 문맥크기에 따라 버디레벨 k를 결정하여 명령에 삽입한다. 함수를 번역할 때 인용되는 함수의 문맥크기를 미리 알 수 없는 경우에는 링크할 때 버디레벨 k를 call명령에 삽입한다.

미세동시다중쓰레딩에서는 각 쓰레드로부터 명령이 하나씩 인출·해독되어 여러 개의 명령 흐름이 동시에 명령 파이프라인을 흐른다. 명령이 파이프라인을 흐르기 위해서는 먼저 해당 쓰레드의 문맥저장을 위한 레지스터가 할당되어야 하고 명령 파이프라인의 각 단계에서 인출기, 해독기, 연산장치, 캐쉬기억, 주기억장치 등 필요한 자원을 할당받아야 한다. 이러한 자원을 할당받을 때까지 쓰레드의 흐름은 일시적으로 중단된다. 그외에도 데이타 종속과 제어종속에 의하여 흐름이 중단되기도 하는데, 이 경우 종속된 명령의 연산시간 만큼 지연된다. 시스템 내에 쓰레드가 충분히 있으면 어떤 쓰레드의 흐름이 중단되어도 다른 쓰레드가 대체되어 명령 파이프라인을 흐른다. 미세다중쓰레딩에서는 쓰레드의 대체로 인한 시간 손실이 없으므로 프로세서 효율을 일정수준으로 유지할 수 있다. 레지스터 블럭할당에 따른 쓰레드의 상태변화는 (그림 7)과 같다. 생성된 쓰레드는 준비상태에서 레지스터 블럭을 할당받기 위해서 기다리고 쓰레드들간의 레지스터 할당 경쟁은 순환할당 규칙을 갖는 하드웨어 대기행렬에 의해서 제어된다. 레지스



(그림 7) 쓰레드 상태 변환
(Fig. 7) State transition of a thread

터를 할당받은 쓰레드는 실행상태로 들어간다. 실행이 종료되지 않은 상태에서 사건을 기다릴 일이 생기면 대기상태로 들어간다. 쓰레드가 실행중에 기다리는 사건은 위에서 언급한 여러가지 종속관계의 해소이다.

대기상태에서 사건 해결 즉, 종속관계가 해소되면 다시 실행상태로 들어간다. 쓰레드 서로간에도 여러가지 종속관계가 있을 수 있는데 그 결과 deadlock이 일어날 수 있다. 이때 레지스터를 할당받은 쓰레드가 모두 대기상태에 있어서 실행상태의 쓰레드가 하나도 없게 된다. 이 경우에는 준비상태에 있는 쓰레드 하나를 선택하여 레지스터 블럭을 양도하고 해제상태로 들어가게 함으로써 deadlock을 해소한다. 해제상태에서 사건이 해결되면 준비상태에 들어가서 다시 레지스터 블럭할당을 요구한다.

3. 모의실험 및 분석

네가지 레지스터 할당기법을 실험한다. 첫번째는 무한 크기의 레지스터 파일을 가진 프로세서이고, 두번째는 128개의 레지스터 파일을 32개의 레지스터를 갖는 4개의 블록으로 할당하는 고정 레지스터 파일할당 기법이고, 세번째는 128개의 레지스터 파일을 버디 2, 3, 4, 5로 구분하여 할당하는 버디 레지스터 할당기법이다. 그리고 네번째는 위의 버디 레지스터 할당을 소프트웨어적으로 수행하는 재배치 레지스터 할당 기법이다. 첫 번째 무한크기의 레지스터 파일의 실험결과는 다른 실험결과들에 대한 비교 기준으로서 사용된다.

레지스터를 제외한 프로세서의 자원은 무한하며 쓰레드 안의 명령 사이에는 항상 종속관계가 있다고 가정한다. 그리고 실행시간이 한 클럭주기일 경우에는 지연없이 파이프라인을 흐를 수 있으나 두 클럭주기 이상일 경우에는 쓰레드의 흐름이 그 시간만큼 멈춘다고 가정한다. 따라서 쓰레드가 레지스터를 할당받고 종속된 명령의 지연이 없는 한 실행이 계속될 수 있다. 이렇게 다른 요인을 배제함으로써 레지스터 파일 자원의 제한이 성능저하에 미치는 영향을 알 수 있다.

실제의 프로그램은 한 개의 쓰레드로부터 실행을 시작하여 수행과정중에 증가, 감소를 반복하다가 하나로 되어 사라진다. 그러나 프로그램을 쓰레드로 분할하는 컴파일러는 아직 개발되어 있지 않으므로 본 논문에서는 쓰레드 개수에 따른 성능변화를 조사한다. 쓰레드는 모의실험이 시작할 때 발생되어 수명이 다하면 새로운 쓰레드를 생성하고 자신은 사라진다. 새로 생성된 쓰레드는 분포곡선에 따라 새로운 매개변수의 값을 할당 받기 때문에 사라진 쓰레드와는 다른 동작특성을 갖는다.

대기상태에 있는 쓰레드가 레지스터 블럭을 양도하는 조건은 사용가능한 레지스터 영역이 없고 실행중인 쓰레드의 개수가 하나도 없어서 deadlock이 발생한 경우로 한정한다.

3.1 입력매개변수

전체 레지스터 파일은 128개의 레지스터로 구성되며 모의실험 수행시간은 10만 클럭주기로 한다. 10만 클럭주기 이상 수행해도 결과에는 변화가 없었다. 매개변수로서 쓰레드 개수, 문맥의 크기, 잠복시간, 쓰레드의 수명 및 실행시간간격이 레지스터의 이용에 미치는 영향을 조사한다. 총 모의실험 시간 중 쓰레드가 실행상태에 있는 시간비를 성능으로 정의한다. 레지스터 파일 부족으로 인하여 성능이 저하되는 정도를 측정하기 위하여, 레지스터가 무한히 있는 시스템의 성능에 대한 세가지 기법의 성능비를 산출한다. 명령이 수행되는 과정에서 〈표 1〉은 세가지 할당기법에 공통적으로 사용된 시뮬레이션 입력 매개 변수들이며 〈표 2〉는 세가지 할당방식 각각의 시간적 비용을 나타낸다. 〈표 1〉에서 실행시간간격은 단일 쓰레드에서 종속명령에 의해 파이프라인이 스톨되는 사건들 사이의 시간간격을 뜻한다. 잠복시간은 대기상태의 쓰레드가 종속문제가 해소되기 까지의 시간을 뜻한다.

〈표 1〉 시뮬레이션 매개 변수
〈Table 1〉 Simulation parameters

입력매개변수	값	비 고
쓰레드 개수	1~12 개	
평균 문맥 크기	1~24 레지스터	一樣 분포
평균 쓰레드 수명	8~256 주기	정규 분포
평균 실행시간 간격	4~64 주기	지수 분포
평균 잠복시간	2~32 주기	지수 분포

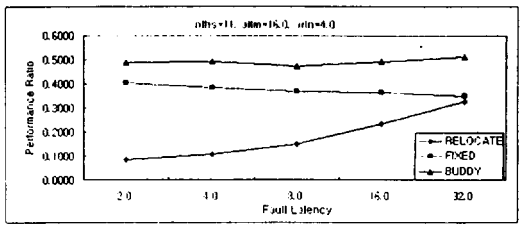
〈표 2〉 시간 비용
〈Table 2〉 Cost assumptions

	고정크기 할당	재배치 할당	버디 할당
할당(성공)	1	25	1
할당(실패)	1	15	1
해제	1	5	1
문맥 load/store	문맥의 크기	문맥의 크기	문맥의 크기
쓰레드 대기행렬 삽입/제거	1	10	1

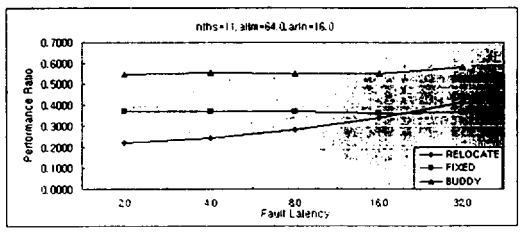
버디할당은 하드웨어 할당기를 사용함으로써 쓰레드가 1 클럭주기 내에 레지스터 블럭을 할당받을 수 있는 것으로 가정한다. 레지스터 파일은 여러 개의 read port와 write port가 있고, 크기도 ABT보다 크므로 블록 할당·해제기의 수행시간 $t_r + 4t_g$은 레지스터 파일의 접근시간과 비교하여 크지 않을 것으로 예상된다.

3.2 쓰레드 수명, 잠복시간, 실행시간 간격 및 쓰레드 개수의 변화

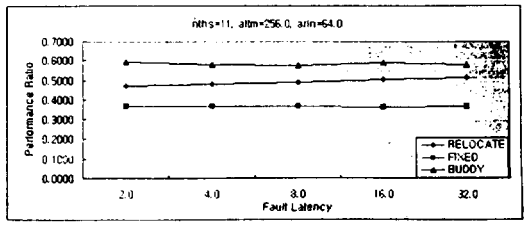
(그림 8), (그림 9) 및 (그림 10)은 발생되는 쓰레드의 수를 11개로 고정하고, 잠복시간(latency)과 실행시간 간격(run length) 및 수명(life time)의 변화에 따른 성능비이다. 성능비는 무한크기의 레지스터 파일에 대한 각 방식의 성능비이다. 잠복시간과 수명 및 실행시간의 정도에 관계 없이 버디 할당기법이 다른 방식보다 우수한 성능을 보인다. 재배치 할당방식은 잠복시간과 수명이 짧을 경우에는 고정크기 할당방식에 미치지 못하나, 그러한 인수들이 길어짐에 따라 점차 우수한 성능을 보인다. 이것은 쓰레드 수행시간에 대한 할당과 해제 시간부담의 비중이 상대적으로 작아진 결과로 풀이된다. 고정크기 할당방식은 잠복시간과 수명이 길어질수록 성능비가 약간씩 저하되는 경향을 보인다. 그러나 버디할당방식의 경우는 거의 변화를 보이지 않는다. Nths는 쓰레드의 개수(number of threads), altm은 평균수명(average life time), arln은 평균실행시간간격(average run length), aflt은 평균잠복시간(average fault latency)를 각각 뜻한다. (그림 10)은 쓰레드의 개수에 따른 성능비 변화를 나타낸다. 세가지 방식 모두 쓰레드 개수가 많아질수록 성능비가 낮아지는데 이것은 쓰레드 개수가 많아질수록 레지스터 자원의 한계가 성능향상을 제한하는 것으로 풀이된다.



(a) 쓰레드개수 = 11, 평균수명 = 16, 평균실행시간 = 4인 경우
(a) nths = 11, altm = 16, arln = 4

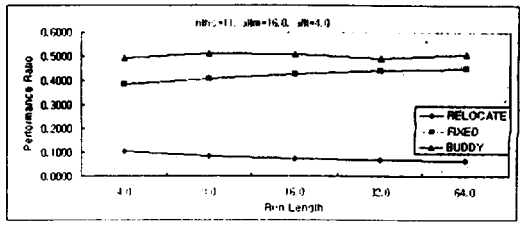


(b) 쓰레드개수 = 11, 평균수명 = 64, 평균실행시간 = 16인 경우
(b) nths = 11, altm = 64, arln = 16

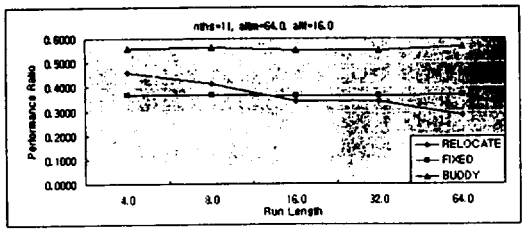


(c) 쓰레드개수 = 11, 평균수명 = 256, 평균실행시간 = 64인 경우
(c) nths = 11, altm = 256, arln = 64

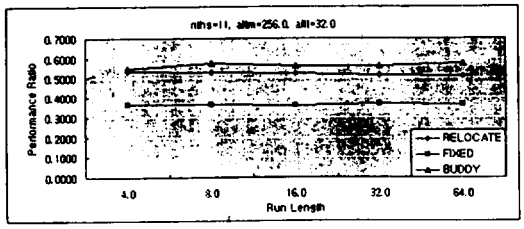
(그림 8) 잠복시간에 따른 성능의 변화
(Fig. 8) Performance ratio vs. fault latency



(a) 쓰레드개수 = 11, 평균수명 = 16, 평균잠복시간 = 4인 경우
(a) nths = 11, altm = 16, aflt = 4

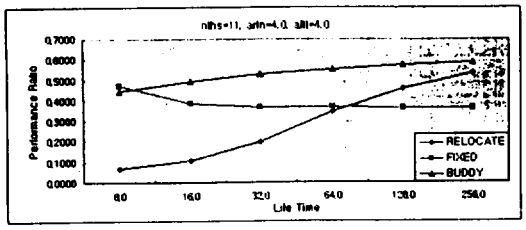


(b) 쓰레드개수=11, 평균수명=64, 평균잠복시간=16인 경우
(b) nths = 11, altm = 64, aflt = 16

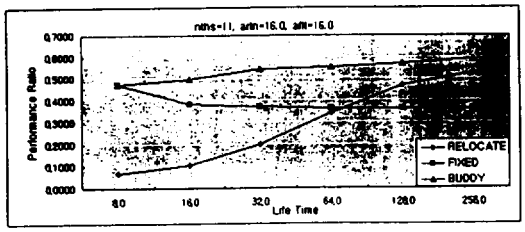


(c) 쓰레드개수=11, 평균수명=256, 평균잠복시간=32인 경우
(c) nths = 11, altm = 256, aflt = 32

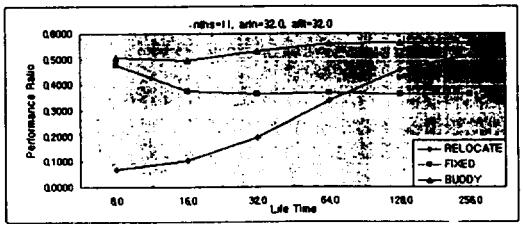
(그림 9) 실행시간에 따른 성능의 변화
(Fig. 9) Performance ratio vs. run length



(a) 쓰레드개수=11, 평균실행시간=4, 평균잠복시간=4인 경우
(a) nths = 11, arln = 4, aflt = 4

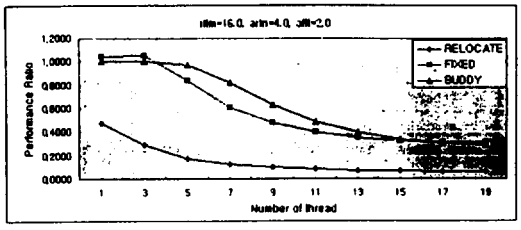


(b) 쓰레드개수=11, 평균실행시간=16, 평균잠복시간=16인 경우
(b) nths = 11, arln = 16, aflt = 16

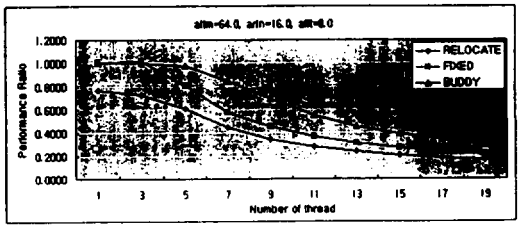


(c) 쓰레드개수=11, 평균실행시간=32, 평균잠복시간=32인 경우
(c) nths = 11, arln = 32, aflt = 32

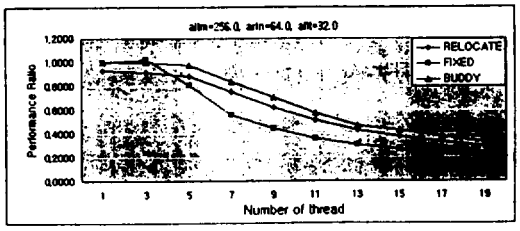
(그림 10) 수명에 따른 성능의 변화
(Fig. 10) Performance ratio vs. life time



(a) 평균수명=16, 평균실행시간=4, 평균잠복시간=2인 경우
(a) nths = 11, altm = 16, aflt = 4



(b) 평균수명=64, 평균실행시간=16, 평균잠복시간=8인 경우
(b) altm = 64, arln = 16, aflt = 8



(c) 평균수명=256, 평균실행시간=64, 평균잠복시간=32인 경우
(c) altm = 256, arln = 64, aflt = 32

(그림 11) 쓰레드의 개수에 따른 성능의 변화
(Fig. 11) Performance ratio vs. number of threads

4. 결론 및 연구과제

미세동시다중쓰레딩의 레지스터로 인한 성능저하 문제를 완화하기 위하여 한 클럭주기만에 버디를 할당·반환하는 버디 레지스터 파일의 구성을 제시하고, 고정크기 레지스터 파일 및 재배치 레지스터 파일과 시스템에 미치는 성능을 비교 분석하였다.

모의실험 결과, 버디 할당기법은 잠복시간과 쓰레드수명, 실행시간간격의 정도에 관계 없이 다른 방식보다 우수한 성능을 나타내었다. 재배치 할당방식은 잠복시간과 수명이 짧을 경우에는 고정크기 할당방식에 미치지 못하나, 그러한 인수들이 길어짐에 따라 쓰레드 수행시간에 대한 할당과 해제 시간부담의 비중이 상대적으로 작아지므로 점차 우수한 성능을 보였다. 고정크기 할당방식은 잠복시간과 수명이 길어질수록 성능비가 약간씩 저하되는 경향을 보였는데, 버디할당방식의 경우는 거의 변화를 보이지 않았다. 또 실행시간간격은 세가지 경우 모두 성능비에 뚜렷한 영향을 미치지 않았다. 쓰레드 개수가 많아질수록 레지스터 자원의 한계가 성능향상을 제한하여 세가지 방식 모두 쓰레드 개수가 많아질수록 성능비가 낮아졌다.

모의실험에서는 버디할당으로 인하여 프로세서의 주기가 변하지 않고, 한 프로세서 주기만에 레지스터 할당과 해제가 가능한 것으로 가정하였다. 앞으로의 과제는 버디할당방식의 레지스터 파일을 설계하여 실제의 레지스터 접근시간과 할당·해제 시간을 확인하고 이 방식의 채택으로 인하여 회로의 면적이 얼마나 커지는지를 확인하는 것이다.

참 고 문 헌

[1] B. Lee, A. R. Hurson, "Dataflow Architectures and Multithreading," IEEE Computer, pp. 27-39, Aug., 1994.

[2] G. T. Byrd, M. A. Holliday, "Multithreaded Processor Architectures," IEEE Spectrum, pp. 38-46, Aug., 1995.

[3] D. M. Tullsen, S. J. Eggers, H. M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," Proc. 22nd Inter. Symp. Comp. Arch., pp. 392-403, 1995.

[4] D. A. Patterson, C. H. Sequin, "A VLSI RISC," IEEE Computer, vol. 15. no. 9, pp. 8-21, 1982.

[5] G. Russel, P. Shaw, "Shifting Register Windows," IEEE Micro, pp. 28-35, Oct. 1993.

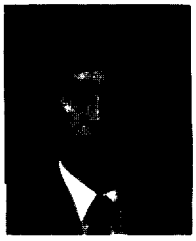
[6] B. K. Bray, "Specialized Caches to Improve Data Access Performance," Pd.D thesis, Stanford Univ. 1993.

[7] C. A. Waldspurger, W. E. Weihl, "Register Relocation: Flexible Contexts for Multithreading," Proceedings the 20th International Symposium on Computer Architectures, pp. 120-130, 1993.

[8] M. Gulati, "Multithreading on a Superscalar Microprocessor," MS thesis, Univ. of California Irvine, 1995.

[9] Knuth, D. E., The Art of Computer Programming I : Fundamental Algorithm Reading, Addison-Wisley, 1972.

[10] Chang J. M, Grhringer E. F, "A High-Performance Memory Allocator for Object-Oriented Systems," IEEE Trans. on Computers, vol. 45, No. 3, pp. 357-366, Mar. 1996.



조 석 환

1980년 영남대학교 전자공학과 졸업(학사)

1985년 영남대학교 대학원 전자공학과 졸업(공학석사)

1995년 영남대학교 대학원 컴퓨터공학과 박사과정 수료

1986년~현재 가톨릭상지전문대학 전산정보처리과 부교수

이 재 도

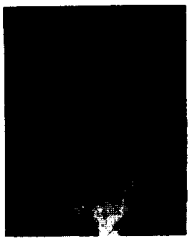
1984년 영남대학교 전자공학과 (공학사)

1989년 영남대학교 전자공학과 (공학석사)

1998년 영남대학교 전산공학과 (박사과정)

1991년~현재 대구보건전문대학 사무자동화과 조교수

관심분야: 컴퓨터시스템, 멀티미디어



윤 영 우

1972년 영남대학교 공과대학 전자공학과(공학사)
1984년 영남대학교 대학원 전자공학과(공학석사)
1988년 영남대학교 대학원 전자공학과(공학박사)
1988년~현재 영남대학교 공과대학 전산공학과 부교수
관심분야: 컴퓨터 구조, 프로그래밍 언어, 병렬처리