직접 사상 캐쉬의 캐쉬 실패율을 감소시키기 위한 성김도 정책

(Determination of a Grain Size for Reducing Cache Miss Rate of Direct-Mapped Caches)

정 인 범 † 공 기 석 †† 이 준 원 †††

(In-Bum Jung)(Ki-sok Kong)(Joon-Won Lee)

요 약 저 전력회로의 설계를 위해서, 전체 회로의 면적을 줄임으로써 용량성 부하(capacitance)값을 줄이는 방법으로 적절한 트랜지스터를 선택하여 사이징하는 방법을 이용할 수 있는데, 이 때 트랜지스터 사이징을 수행하면서 적당한 위치에 버퍼를 삽입하여주면 더 좋은 결과를 가져올 수 있다. 본 논문은 TILOS 알고리즘을 이용하여 트랜지스터 사이징(sizing)을 수행하는 동시에 버퍼의 삽입을 수행하는 알고리즘 두 가지를 소개하고 이 두 방법을 비교한다. 그 첫 번째 방법은 Template Window를 이용하여 직접 시뮬레이션하는 방법이고 다른 하나는 보외법(Extrapolation)을 이용하는 방법이다. 이와 같이 버퍼를 삽입하면서 트랜지스터 사이징을 수행한 결과, 버퍼를 삽입하지 않을 때 보다 10-20%의 면적감소를 얻었을 수 있었으며 보외법을 이용한 방법 보다 Template Window를 이용했을 때 더 좋은 결과를 얻을 수 있었다.

Abstract In data parallel programs incurring high cache locality, the choice of grain sizes affects cache performance. Though the grain sizes chosen provide fair load balance among processors, the grain sizes that ignore underlying caching effect result in address interferences between grains allocated to a processor. These address interferences appear to have a negative impact on the cache locality, since they result in cache conflict misses. To address this problem, we propose a best grain size driven from a cache size and the number of processors based on direct mapped cache's characteristic. Since the proposed method does not map the grains to the same location in the cache, cache conflict misses are reduced. Simulation results show that the proposed best grain size substantially improves the performance of tested data parallel programs through the reduction of cache misses on direct-mapped caches.

1. 서 론

데이타 병렬성을 나타내는 병렬 프로그램은 연속적 주소 공간을 가지는 커다란 데이타 배열들에 대한 동일한 처리를 수행하는 특성을 가지고 있다. 이러한 프로그램에서 병렬 처리를 수행하기 위해서는 데이타 배열을 그레인 크기에 따라서 프로세서들에게 배분하므로 이루어진다. 이때 사용되는 그레인 크기들은 프로세서들 사이에 공정한 부하 배분을 하기 위해서 데이타 배열을 프로세서 개수의 배수로 나누어 얻어진다. 따라서 사용되는 그레인 크기는 배수의 크기에 의해서 최대 성긴 그레인(coarse grain)부터 최소 조밀 그레인(fine grain)까지 분류될 수 있다. 데이타 배열의 일부분을 차지하는 이러한 그레인들은 메모리 주소상에서 연속적인 지역에 위치하게 되므로 그레인 내부의 항목들을 참조시 공간적 구역성(spatial locality)을 발생하게 된다. 또한 각각의 프로세서에게 할당된 그레인들은 병렬 루프에서 반

· 본 논문은 정보통신부 대학소프트웨어연구센터 지원사업(리눅스 기반 실시간 운영체제 및 통신)과학기술부 국가지정연구실 지원사업(고성능 클러스터 시스템 개발)의 지원을 받았습니다.

† 학생회원 : 한국과학기술원 전자전산학과
jib@calab.kaist.ac.kr
†† 정 회 원 : .한국전자통신연구원 내장형S/W연구팀 연구원
kskong@etri.re.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
joon@cs.kaist.ac.kr

논문접수 : 1999년 12월 6일
심사완료 : 2000년 5월 30일

복적으로 사용되므로 시간 구역성(temporal locality)을 발생한다.

발생된 구역성은 캐쉬 메모리에서 이용되므로 프로그램의 성능 향상에 캐쉬 메모리는 커다란 영향을 미치고 있다. 그러나 사용되는 그레인 크기가 프로세서들 사이에 균등한 부하 균형을 제공하더라도 캐쉬 메모리의 특성을 고려하지 않을 경우 캐쉬 공간에서 그레인들 사이에 주소 간섭(address interference) 현상을 일으킬 수 있다. 이런 주소 간섭은 캐쉬 충돌 실패(cache conflict miss)를 발생하므로 프로그램에서 발생하는 메모리 참조 구역성에 손상을 입히게 된다. 특히 캐쉬 메모리 중 직접사상(direct mapped) 캐쉬는 어소시어티브(associative) 캐쉬들보다 더 많은 캐쉬 충돌 실패를 발생시키므로 메모리 참조 지연시간을 더욱 증가시킨다. 그러나 간단한 하드웨어 설계를 기반으로 한 직접 사상 캐쉬는 빠른 동작을 제공하며, 또한 단순한 동작 특성은 주소 간섭을 줄일 수 있는 그레인 크기를 계산 할 수 있게 한다.

본 논문에서는 전형적인 데이타 병렬 프로그램들을 선택하고 이들 프로그램들에서 부하 균형만을 고려한 다양한 그레인 크기들이 이용될 때 각각의 프로세서에 할당된 그레인들 사이의 주소 간섭과 이에 의한 캐쉬 충돌 실패들을 측정한다. 이러한 관찰을 바탕으로 직접 사상 캐쉬상에서 캐쉬의 크기와 병렬처리에 참가한 프로세서들의 개수를 이용하므로 그레인들 사이의 주소 간섭을 줄일 수 있는 최적 그레인 크기를 제안한다. 제안된 최적 그레인 크기는 각각의 프로세서들에게 할당된 그레인들이 캐쉬안에서 같은 주소 공간을 차지하지 않게 하므로 주소 간섭 현상을 현저하게 줄인다. 따라서 캐쉬 충돌 실패에 의하여 프로그램의 메모리 참조 구역성이 영향 받지 않으므로 프로그램 성능 향상에 기여하게 된다. 모의 시험에서 제안된 최적 그레인 크기는 시험된 병렬 프로그램들에서 부하 균형만을 위하여 선택된 모든 다른 그레인 크기들보다 개선된 성능을 나타낸다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 사용된 데이타 병렬 프로그램들 및 이들을 병렬 처리 할 때 사용되는 그레인 크기 정책에 대하여 설명한다. 3장에서는 직접 사상 캐쉬 특성을 고려한 최적 그레인 크기를 제시한다. 4장에는 본 논문에서 사용된 모의 시험 환경에 대하여 설명한다. 5장에서는 시험된 병렬 프로그램에서 최적 그레인 크기의 성능 평가를 수행한다. 6장에서는 본 논문과 관련된 연구에 대하여 설명하며 7장에서 본 논문의 결론을 제시하고자 한다.

2. 데이타 병렬 프로그램에서 부하 균형 배분을 위한 그레인 크기

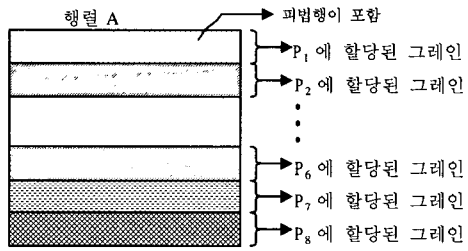
본 논문에서 사용되는 데이타 병렬 프로그램들로는 과학 계산용 프로그램으로 널리 알려진 LU(LU-Decomposition), FFT(Fast Fourier Transform), BMM(Blocked Matrix Multiplication) 3가지를 선택하였다[8]. 이들 프로그램들은 모든 데이타들에 대하여 동일한 처리를 수행하므로 병렬 처리를 위해서는 데이타 항목들을 프로세서들에게 균형 있게 배분하면 되는 데이타 병렬성을 보인다. 특히 선택된 프로그램들은 수행도중 동적으로 데이타 항목들이 증가되지 않으며 선택된 그레인 크기는 메모리의 연속된 공간을 차지하므로 그레인 크기에 의한 주소 간섭 현상을 분석하기에 용이한 특징을 가지고있다. 본 연구에서는 그레인들에 대한 스케줄링 정책으로는 정적 스케줄링(static scheduling)을 사용하여 캐쉬 효과를 증가하고자 했다. 정적 스케줄링은 각 프로세스가 수행할 그레인들이 프로그램 시작하기 전에 지정되는 방식으로 데이타 병렬 프로그램과 같이 데이타에 대한 참조 형태가 고정적인 프로그램들에서 캐쉬 성능을 향상시키므로 효과적이다[5,9,10].

다음은 선택된 프로그램들의 특징과 이들 프로그램들에서 사용하는 그레인 크기들에 대한 설명이다. 그레인 크기는 프로세서에 할당된 데이타 양에 따라서 가장 성긴 그레인(coarse grain) 크기와 이보다 작은 그레인 크기들로 분류된다. 여기서 선택된 그레인 크기들은 가능한 모든 프로세서들에게 동등한 부하 균형이 유지되도록 하였다.

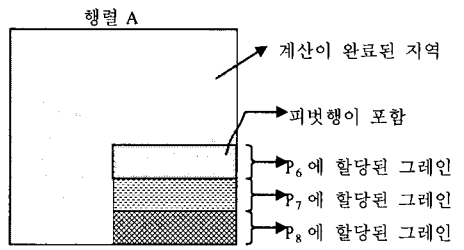
- **LU 프로그램**

LU는 행렬 A를 하삼각 행렬 L과 상삼각 행렬 U의 곱으로 분해하는 알고리즘이다. 주요 데이타 구조는 분해되는 행렬이다. LU 프로그램의 계산 과정은 가장 밖에 있는 루프에서 피벗(pivot)행이 계산된 후 피벗행 밑에 있는 행(rows)들이 피벗행의 값을 가지고 계산된다. 8개의 프로세서를 가지고 행렬 크기 256×256를 병렬 처리를 할 경우 최대 성긴 그레인 크기는 32 rows 이고 작은 그레인 크기들로는 16 rows, 8 rows, 4 rows, 2 rows, 1 row 가 사용될 수 있다. 제시된 그레인 크기들은 프로그램이 수행 시작 전 에는 각각의 프로세서에게 동일한 부하(workload)를 분배하게 한다. 그러나, LU 프로그램은 계산이 진행됨에 따라 계산량이 점점 적어지는 특성을 가지고있다. 즉 순환 구문의 단계

가 반복될 때 마다 해당 피벗행이 행렬의 아래로 진행되고 따라서 계산량이 줄어들게 된다. 따라서 일찍 계산 완료되는 그레인들을 갖는 프로세서들은 더 이상 계산에 참여하지 못하므로 프로세서 활용성(utilization)이 떨어진다. 예로서, 그림 1은 성긴 그레인 크기가 사용될 때 프로그램의 시작 시점 및 행렬의 절반 이상이 수행 완료된 시점에 행렬에 남아 있는 그레인들과 이들을 수행하는 프로세서들을 보여준다. 그림 1(나)에 나타나듯이 전체 계산량의 절반 이상이 완료된 경우 절반 이상의 프로세서들이 계산에 참여하지 못함을 보여준다(P_i는 프로세서 번호i 를 표시한다).



(가) 계산 시작전 상태



(나) 절반 이상이 완료된 상태

그림 1 병렬 처리되는 LU 프로그램

• BMM 프로그램

BMM 프로그램은 시간 구역성을 개선하기 위한 최적화 기법인 블록화 알고리즘을 적용한 행렬 곱셈이다[1,3]. 주요 데이타 구조는 행렬 곱셈을 위한 3개의 2차원 배열들이다. 두개의 행렬 X와 Y를 곱하여 결과 행렬 Z를 계산하기 위하여 행렬 Y 내부의 서브 행렬들인 B×B 블록들이 집중적으로 반복 사용된다. 이 프로그램에서는 블록 크기 B가 병렬 처리를 위한 그레인 크기로 사용된다. 256×256 행렬들의 곱셈을 16개의 프로세서를 사용하여 병렬처리 할 경우 최대의 성긴 그레인 크기는 16×16 블록이고 작은 그레인 크기들은 8×8 블록, 4×4 블록, 2×2 블록, 1×1 블록 이다. 이들 그레인 크기들은 16개의 프로세서들 사이에 공정한 부하 배분을 위하여 행렬 Y의 행 크기를 프로세서 개수의 배수로 나누어 얻었다.

그림 2은 BMM 프로그램에서 행렬들이 16개의 프로세서에 의해 성긴 그레인 크기로 나누어진 후 각각의 프로세서에서 참조되는 지역들을 보여준다. 그림에 나타나듯이 각각의 프로세서는 행렬 Z와 Y에서는 고정된 부분을 할당 받으며 이들 지역들은 프로그램 수행 중 반복적으로 재사용되므로 캐쉬 성능을 향상 시킨다. 반면에 행렬 X는 프로그램 진행 중 모든 프로세서에 의해 한번만 참조되므로 캐쉬 성능에 미치는 영향은 적다.

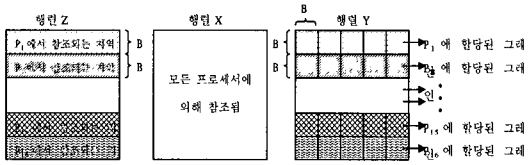


그림 2 병렬 처리되는 BMM 프로그램

• FFT 프로그램

시험에 사용한 FFT 프로그램은 반복적인(iterative) Cooley-Tukey의 알고리즘(N-point, 1차원, 비순서, radix-2 FFT)을 사용했다. 이 알고리즘은 분할(divider)과 결정(conquer)의 특성을 가지고 있다. 주요 데이타 구조로는 소스 포인트 배열과 결과 포인트 배열로 구성되는 2개의 1차원 배열들이다. 입력 포인트, 즉 배열의 항목수가 65536 points일 때 병렬 처리를 위하여 16개의 프로세서를 사용할 경우 최대의 성긴 그레인 크기는 4096 points 이고 작은 그레인 크기들은 2048 points, 1024 points, 512 points, 128 points, 64 points, 32 points, 16 points, 8 points, 4 points, 2 points, 1

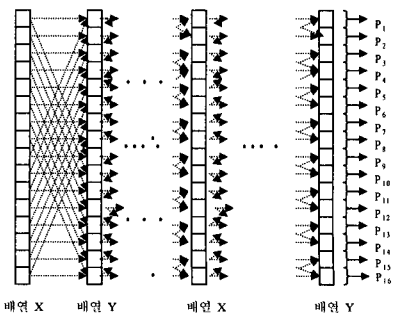


그림 3 병렬 처리되는 FFT 프로그램

point들을 사용할 수 있다. 이들 그레인 크기들은 계산에 참여한 프로세서들 사이에 공정한 부하 배분을 위하여 소스 포인트 개수를 프로세서 개수의 배수로 나누어 얻었다. 그림 3은 FFT 프로그램의 데이타가 16개의 프로세서에 의해 성긴 그레인 크기로 나누어진 후 수행되는 중간 단계들을 보여준다(P_i는 각 프로세서 i 에 할당된 지역을 보여준다).

3. 최적 그레인 크기

병렬처리에 참가한 프로세서들의 캐쉬 메모리 합이 프로그램의 데이타보다 클 경우에도 캐쉬 실패는 발생된다. 이러한 원인은 각각의 프로세서에 할당된 그레인들을 수행하기 위하여 참조되는 주소공간이 캐쉬 영역 안에서 서로간에 주소 간섭을 일으키기 때문이다. 이러한 문제점은 그레인 크기를 결정할 때 캐쉬 메모리의 특성을 고려하지 않으므로 발생한다. 5장의 모의 실험에서는 2 장에서 제시한 부하균형만을 유지하기 위한 그레인 크기들이 서로간의 주소간섭으로 인하여 cache conflict miss가 발생되는 것을 측정한다.

본 논문에서는 이러한 문제점을 해결하기 위하여 직접 사상 캐쉬의 동작 특성을 이용하여 병렬처리에 참여한 프로세서의 개수와 사용된 캐쉬 크기로부터 주소 간섭 현상을 줄이는 최적 그레인 크기를 제안 하고자 한다. 최적 그레인 크기를 도출하기 위해서는 다음과 같은 가정이 필요하다. 첫째, 모든 프로세서들이 같은 캐쉬 크기를 가지고 있어야 한다. 둘째, 프로그램에서 사용되는 데이타 배열들의 주소는 연속적인 주소 공간을 갖는 메모리 지역에 할당되어야 한다. 이를 위하여 운영체제는 page coloring 기법[11]처럼 페이지의 가상 주소와 실 주소가 같은 캐쉬 위치에 사상 되도록 하거나 아니면 하드웨어적으로 가상주소를 사용하여 캐쉬 색인을 하여야 한다. 셋째, 프로그램의 데이타 크기는 사용하는 프로세서들의 캐쉬 총합보다 작아야 한다. 즉 데이타의 크기가 캐쉬들의 크기보다 클 경우 실질적으로 캐쉬 실패를 피하기는 어렵기 때문이다. 넷째, 그레인들의 스케줄링은 정적 스케줄링을 사용한다. 정적 스케줄링은 2장에서 설명한 것처럼 캐쉬 효과를 높일 뿐만 아니라 각각의 프로세서에 할당되는 그레인들을 예측할 수 있다.

최적 그레인 크기를 얻기 전에 다음의 두 가지 용어를 정의한다. 첫째, 최소 그레인 크기(Minimal Grain Size)는 해당된 데이타 병렬 프로그램의 특성상 수행 가능한 가장 작은 그레인 크기를 나타낸다. 둘째, 청크 크기(chunk size) 는 그레인 크기와 프로세서 개수의 곱이며 여러 프로세서가 동시에 수행해야 할 그레인 집합이다. 즉, 하나의 청크에는 각 프로세서에게 할당되는 그레인이 한 개씩 포함된다. 따라서 프로그램의 데이타 배열은 청크 단위로 나누어지며, 모든 프로세서들은 하나의 청크에서 자신에게 할당된 그레인을 수행한 후 다음 번 청크에서 자신의 그레인을 수행하는 과정을 반복하게 된다. 특히, 위에서의 가정들 중 청크내의 그레인들은 정적 스케줄링을 하므로 각각의 프로세서에 할당되는 그레인들은 각각의 청크내에서 고정적 위치를 차지한다

이러한 가정 및 용어를 바탕으로 직접 사상 캐쉬의 주소 사상 (address mapping)은 캐쉬 크기만큼의 주기성을 가지고 반복된다는 특성, 그리고 그레인 크기에 따른 메모리 참조의 주기성을 보이는 데이타 병렬 프로그램의 특성으로부터 최적 그레인 크기는 다음 같이 유추할 수 있다.

$$G = \alpha \times M \quad (1)$$

$$G \times N = C + G \quad (2)$$

G : 최적 그레인 크기
M : 최소 그레인 크기
α: 임의의 상수 값
N : 프로세서 개수
C : 하나의 캐쉬 크기
$G \times N$: 청크 크기

식(1)은 최적 그레인은 최소 그레인 크기의 상수배로 나타남을 표시한다. 최소 그레인 크기는 프로그램이 수행될 수 있는 최소의 단위이므로 이 식은 당연한 귀결이다. 식(2)는 하나의 청크 크기(G×N)가 캐쉬와 최적 그레인 크기의 합(C + G)과 같음을 보인다. 이 식의 유추된 의도는 다음과 같다. 직접 사상 캐쉬에서 캐쉬 크기가 C 이고 참조되는 메모리의 주소가 A 일때 캐쉬 주소공간에 사상되는 위치는 A mod C의 결과, 즉 주소 A를 캐쉬 크기 C로 나눈 나머지에 해당되는 캐쉬 주소이다. 따라서 식(2)처럼 하나의 청크 크기가 캐쉬 크기(C) 와 최적 그레인 크기(G)의 합으로 지정할 경우 캐쉬 공간에서 다음 번 청크의 시작 주소는 G 만큼 지난 주소를 차지하게 된다. 또한 위에서 설명된 것처럼 청크내의 그레인들에 대하여 정적 스케줄링을 사용하므로 각각의 프로세서는 청크들에서 일정한 위치에 있는 그레인을 할당받는다. 따라서 G 만큼 지난 주소에서 시작되는 다음 청크내의 그레인은 직접 사상 캐쉬 주소 공간에서 현재 청크의 그레인이 차지한 지역 다음에 뒤이어서 위치하게된다. 따라서 그레인들 사이의 주소간섭은

발생하지 않게 된다.

그림4는 한개의 데이타 배열 X[]를 사용하는 병렬 프로그램에서 4개의 프로세서가 최적 그레인을 사용하여 할당된 그레인들을 자신의 캐쉬 공간에 위치시키는 과정을 보여준다. 예로써 프로세서 P_1에는 두개의 그레인이 할당된다. 캐쉬 주소 공간에서 첫번째 그레인의 시작 위치는 &X[i] mod C 주소에 해당되며, 반면에 두번째 그레인의 시작 위치는 &X[i + C + G] mod C 주소에 해당된다. 따라서 그림 4에서 나타나듯이 첫번째 그레인이 프로세서 P_1의 캐쉬상에 자신의 크기 G 만큼을 차지한 후 두번째 그레인이 뒤이어서 또한 G 만큼의 캐쉬 공간을 차지하게 된다.

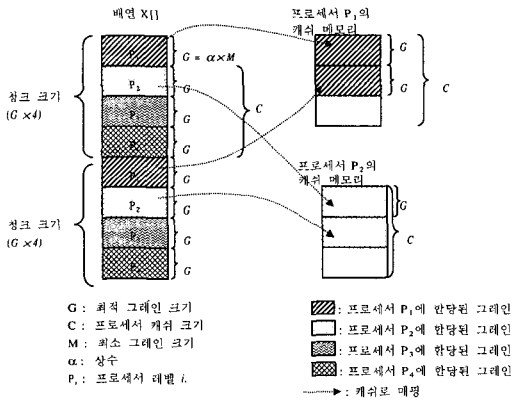


그림 4 단일 배열에서 최적화 그레인을 사용한 그레인들의 배치

$$G \times N - G = C, \qquad G = \alpha \times M = \frac{C}{(N-1)}$$

식(1)과 (2)로 부터 다음의 식들이 유추된다.
상기 식들로부터 식(3)이 다음처럼 유도된다.

$$\alpha = \frac{\lceil C \rceil}{(N-1)M} \tag{3}$$

최적 그레인 크기는 식(1)에서 최소 그레인 크기의 상수배로 정의했고 식(3)은 그 상수값을 나타낸다. 이 상수값이 프로그램이 실제 사용하는 그레인 크기를 나타낸다.

최적 그레인 크기는 다중 배열을 사용하는 경우에도 적용될 수 있다. 그러나 각 배열의 크기가 청크 크기의 배수가 아닐 경우 마지막 청크로부터 자신의 그레인을 할당받지 못하는 프로세서들이 발생한다. 이러한 데이타 배열들에 최적 그레인을 적용할 경우 주소 간섭이 배제된 데이타들이 프로세서들에게 할당되지 않을 수 있다. 따라서 다중 데이타 배열들을 사용할 경우 배열과 배열 사이에 계산에서는 사용되지 않는 패딩 구조를 삽입하여 마지막 청크를 채워준다. 즉, 모든 배열의 시작 주소가 청크의 시작주소가 되도록 한다. 따라서 모든 프로세서는 최적 그레인 크기에 따라 주소간섭이 발생하지 않는 예정된 주소에 위치한 자신의 그레인들을 각각의 청크로부터 할당받을 수 있게된다. 패딩 작업은 프로그램에서 데이타 구조 할당을 위한 메모리 할당 부분에서 쉽게 추가 될 수 있다.

그림 5는 두개의 배열 X[], Y[]와 5개의 프로세서를 사용할 때 최적 그레인이 사용되는 것을 보여준다. 프로그램에서 배열 X[]와 Y[]는 연속적으로 메모리를 차지하며 각 배열들의 크기는 청크의 배수가 아님을 가정한다. 따라서 배열 X[]의 마지막 청크인 두번째 청크에는 패딩 구조가 추가되었다. 한 예로써 프로세서 P_1에는 네개의 그레인이 할당된다. P_1 의 캐쉬 주소 공간에서 첫번째 그레인의 시작 위치는 *&X[i] mod C* 주소이며, 두번째 그레인의 시작 위치는 &X[i + C + G] mod C, 세번째 그레인은 &X[i + 2C + 2G] ≡ &Y[i] mod C, 네번째 그레인은 Y[i + C + G] mod C 에 해당되는 주소를 차지한다. 따라서 그림 5에 나타나듯이 프로세서 P_1 의 캐쉬 메모리에 이들 네개의 그레인들은 주소 충돌 없이 연속적으로 위치하게 된다.

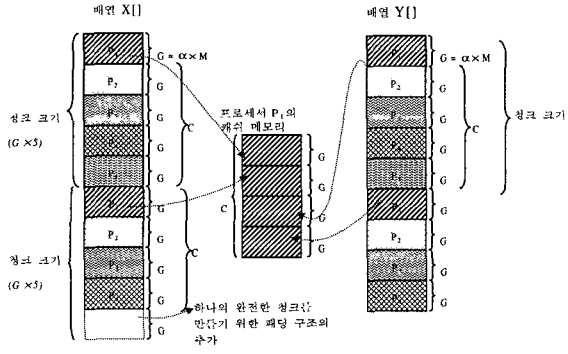


그림 5 다중 배열에서 최적화 그레인을 사용한 그레인들의 배치

4. 모의 시험 환경

모의시험 환경은 병렬 프로그램들을 수행하는 기능 시험기(functional simulator)와 공유 메모리 다중 프로세서를 모의 시험하는 구조 시험기(architectural simulator)로 구성된다. 기능 시험기로는 MINT(Mips INTerpreter)[6]를 사용하였고 구조 시험기로는 버스 기반의 다중 프로세서를 구성하였다. 프로세서들은 메모

리 참조를 제외하고는 하나의 사이클(cycle)당 하나의 명령어를 수행하는 동일한 캐쉬 메모리 크기를 갖는 리스크(RISC) 프로세서로 가정하였다.

모의시험에서 사용한 캐쉬는 크기가 128 Kbytes인 직접 사상(direct-mapped) 캐쉬로 캐쉬 라인은 16 bytes를 사용하였다. 모의 시험된 캐쉬 일관성 프로토콜은 Illinois[7]에서 제안된 4가지 상태를 갖는 쓰기 무효화 프로토콜(write invalidation protocol)을 사용했다. 현재 마이크로 프로세서 기술(예 : MIPS R10000, UltraSparc-II)에 근거하여 클록율(clock rate)은 250 Mhz로 메모리 지연시간은 약 80 ns(nano second), 시스템 버스폭은 128 bits로 가정하였다. 표1은 이러한 가정을 기반으로 하여 1 싸이클의 버스 전송 시간과 1 싸이클의 주소 해석 시간을 포함한 캐쉬 일관성 프로토콜의 시간 변수들의 값을 나타내고 있다.

표 1 캐쉬 일관성 프로토콜을 위한 시간 변수값

사건	제어 동작	소요시간
공유 캐쉬 라인에 대한 쓰기	무효화 신호 처리	3 cycles
캐쉬 실패	다른 프로세서의 캐쉬로부터 데이타 전송	7 cycles
캐쉬 실패	메모리로부터 데이타 전송	22 cycles

5. 성 능

최적 그레인 크기에서의 프로그램 성능과 2장에서 제시한 단순히 부하의 균형을 맞추기 위하여 제시된 다양한 그레인 크기들에서의 프로그램 성능을 비교한다. 표 2는 각 프로그램들의 데이타 크기, 사용된 프로세서 개수, 최소 그레인 크기 그리고 4장의 식(3)에서 유도된 최적 그레인 크기를 보여준다. 사용된 프로그램의 데이타 크기와 기타 그레인 크기들은 2장에서 제시된 것을 사용한다. 또한 캐쉬 크기가 충분하여도 캐쉬 실패가 발생하는 현상을 관찰하기 위하여 각 프로그램의 데이타 크기는 계산에 참여한 프로세서들의 캐쉬 크기의 총합보다 작게 하였다.

표 2 프로그램들과 최적 그레인 크기

프로그램	데이타 크기	프로세서 개수	최소 그레인 크기	최적 그레인 크기
LU	하나의 256×256 행렬(512 Kbytes)	8	1 row (2 Kbytes)	10 rows
FFT	두 개의 65536 points배열 (1 Mbytes)	16	1 point (8 bytes)	1093 points
BMM	세 개의 256×256 행렬(1.5 Mbytes)	16	1×1 블록들로 구성된row (2 Kbytes)	5×5 블록들로 구성된5 rows

시험에서 측정할 척도는 프로그램의 수행 시간과 캐쉬 실패를 측정하였다. 캐쉬 실패는 4가지 경우로 분류하여 측정하였다. 즉 캐쉬 실패의 3C 모델[2]에 의한 Compulsory miss, Conflict miss, Capacity miss와 공유 캐쉬 블록에 대한 캐쉬 일관성 프로토콜 유지를 위한 protocol miss이다. 이러한 분류된 캐쉬 실패의 종류들은 캐쉬 크기가 충분함에도 불구하고 그레인들 사이의 주소 간섭에 의하여 발생되는 cache conflict miss가 전체 캐쉬 실패에 미치는 정도를 평가할 수 있게 해준다.

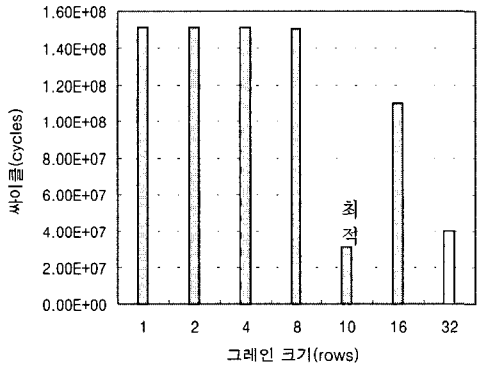
- **LU 프로그램**

그림 6은 LU 프로그램이 최적 그레인 크기와 2장에서 제시된 다른 그레인 크기들에서 수행될 때 수행 시간과 캐쉬 실패를 나타낸다. 최적 그레인 크기(10 rows)에서 가장 좋은 성능을 보인다. 그러나 그림 6(나)에서 최적 그레인을 제외한 나머지 그레인 크기들에서는 성긴 그레인 크기(32rows)에서 가장 적은 캐쉬 실패를 발생하였다. 이러한 이유는 성긴 그레인 크기로 데이타들을 분할 할 경우 각각의 프로세서에 한 개의 그레인만 할당되고, 프로그램의 데이타 크기가 프로세서 캐쉬들의 합보다 작기 때문에 하나의 그레인 크기는 하나의 프로세서 캐쉬 메모리에 모두 적재될 수 있기 때문이다. 따라서 성긴 그레인 크기를 사용할 때 발생되는 캐쉬 실패는 할당된 그레인과 계산을 위해 필요한 피봇행 사이의 주소간섭에 의한 cache conflict miss에 의해서만 발생된다. 반면에 성긴 그레인 크기보다 작은 그레인 크기를 사용하는 경우 각각의 프로세서에는 여러 개의 그레인들이 할당되고 이들 그레인들은 캐쉬 공간에서 주소간섭을 발생하므로 캐쉬 실패가 증가되었다.

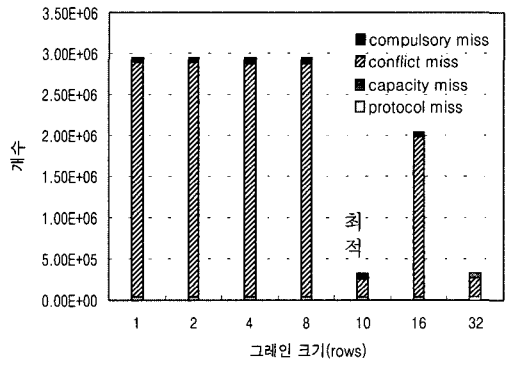
최적 그레인 크기도 성긴 그레인 크기보다 작은 크기를 갖는다(즉, 10 rows). 그러나 3장에서 설명한 것처럼 최적 그레인 크기를 사용할 경우 각각의 프로세서에 할당된 그레인들은 캐쉬 메모리에서 서로간 주소간섭을 발생하지 않는다. 따라서 성긴 그레인 크기와 마찬가지로 그레인들과 피봇행 사이의 주소간섭만이 캐쉬 실패의 원인이 되었다. 그러므로 최적 그레인 크기와 성긴 그레인 크기는 대등한 캐쉬 실패를 발생함을 보인다.

이렇게 동등한 캐쉬 실패 개수를 갖더라도 그림6(가)에서 나타나듯이 최적 그레인은 성긴 그레인 보다 좋은 성능을 보인다. 이러한 이유는 2장에서 설명했듯이 성긴 그레인을 사용할 경우 LU 프로그램의 특성상 계산이

진행됨에 따라 계산에 참여하지 못하는 프로세서들이 생기기 때문이다. 즉, 성긴 그레인 크기를 사용할 때 각각의 프로세서에는 한 개의 그레인이 할당되고 이들 그레인들 중 계산이 일찍 종료되는 그레인을 할당받은 프로세서는 계산 과정에서 일찍 제외되기 때문이다. 반면에 최적 그레인 크기는 각각의 프로세서에게 여러 개의 작은 그레인들을 할당된다. 이들 그레인들은 행렬 내부에 서로 떨어져 흩어져 위치하고 있으므로 모든 프로세서들은 LU 프로그램의 특성상 처리할 데이타량이 점차 줄어들지라도 프로그램 종료시 까지 모두 계산에 참여하게 된다. 즉, 최적 그레인 크기는 작은 그레인들 사이의 주소간섭을 제거하므로 cache conflict miss를 감소했을 뿐만 아니라 프로세서 활용성을 높이므로 성긴 그레인 크기보다 약 22.6%의 성능 향상을 가져왔다.



(가) 수행 시간



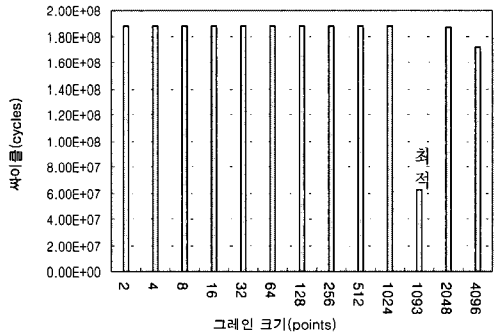
(나) 캐쉬 실패

그림 6 LU 프로그램: 최적화 그레인 크기와 다른 그레인 크기들과의 비교

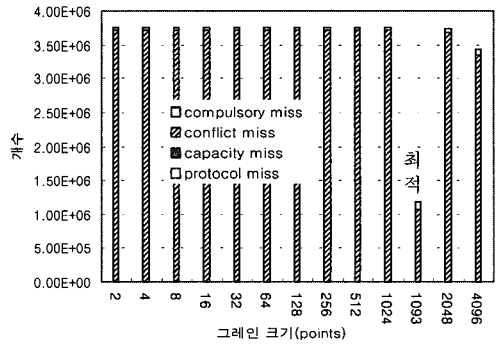
- **FFT 프로그램**

그림 7은 최적 그레인 크기와 1 point그레인 크기를 제외한 다른 그레인들에서 FFT 프로그램의 수행 시간들과 캐쉬 실패를 나타낸다. 1 point의 그레인 크기는 성능이 너무 저조해 그림에서 제외하였다. 그림 7(가)는 최적 그레인 크기(1093 points)가 가장 좋은 성능을 나타냄을 보인다. FFT 프로그램은 2개의 배열들을 사용하며 각각의 배열에 최적 그레인 크기를 적용할 수 있다. 그러나 배열들의 크기가 청크 크기의 배수로 구성되지 않으므로(즉, 청크 크기 : 16 processors×1093 points, 데이타 크기 : 65536 points) 3장에서 설명된 것처럼 마지막 청크에는 4416 points 만큼의 패딩 구조를 추가하였다.

그림 7(나)에 측정된 것처럼 최적 그레인 크기를 제외한 모든 그레인 크기들에서는 cache conflict miss가 가장 큰 캐쉬 실패의 원인임을 알수있다. 이런 cache conflict miss는 하나의 프로세서에 할당된 그레인들 사이의 주소 간섭으로 발생된다. 반면에 최적 그레인 크기에서는 그레인들 사이의 주소간섭이 제거되므로 cache



(가) 수행시간



(나) 캐쉬 실패

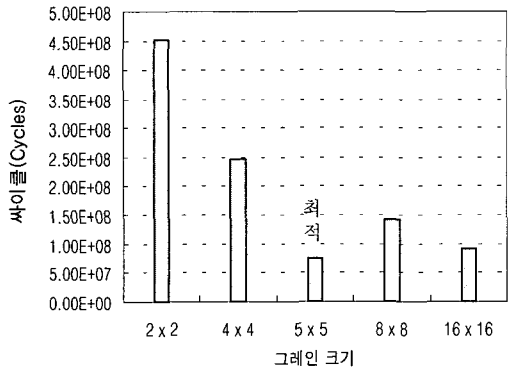
그림 7 FFT 프로그램: 최적화 그레인 크기와 다른 그레인 크기들과의 비교

conflict miss가 다른 그레인 크기들 보다 현저하게 줄어들었다. 이러한 캐쉬 실패의 감소는 그림 7(가)에 나타난 것 처럼 최적 그레인 크기는 성긴 그레인 크기(4096 points)보다 약 63.9%의 성능 향상을 제공하였다.

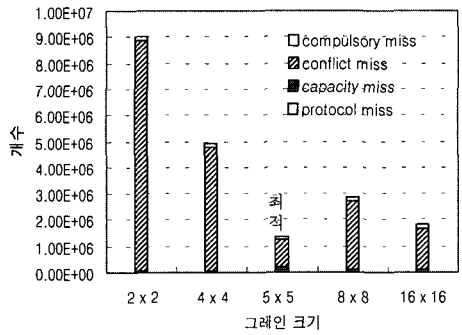
- **BMM 프로그램**

BMM 프로그램은 3개의 배열들을 사용한다. 2장에서 설명된 것 처럼 행렬 X와 행렬 Y를 곱하여 Z 행렬을 생성하는 경우 행렬 Y와 행렬 Z는 그레인 크기에 따라서 행렬내의 고정된 부분들이 각각의 프로세서에 할당된다. 따라서 이들 행렬에는 최적 그레인 크기를 적용할 수 있다. 반면에 행렬 X에 대해서는 각각의 프로세서들이 자신이 참조하는 고정된 범위를 갖는 것이 아니라 계산이 수행되는 도중에 행렬 전체 항목을 한번씩 참조하게 되므로 최적 그레인 크기의 적용이 의미가 없다. 따라서 BMM 프로그램에서 최적 그레인 크기를 행렬 Y, Z에 적용하였다. 또한 최적 그레인을 사용할 때 행렬의 크기가 청크 크기의 배수가 아니므로 (즉, 청크 크기 : 16 processors×5 rows, 데이타 크기 : 256 rows) 3장에서 설명된 것처럼 행렬 Y와 Z의 마지막 청크에 64 rows 만큼의 패딩 구조가 추가되었다.

그림 8은 BMM 프로그램이 최적 그레인 크기와 다른 그레인들에서 수행될 때 수행 시간들과 캐쉬 실패를 나타낸다. 1×1 그레인 크기는 성능이 너무 저조해서 그래프에서 제외했다. 그림 8(나)에 측정된 것처럼 최적 그레인 크기는 가장 적은 캐쉬 실패를 발생하였고 따라서 가장 좋은 성능을 나타내었다. 이러한 이유는 다른 그레인 크기들이 사용될 때 하나의 프로세서에는 행렬 Y 와 Z로부터 할당된 그레인들이 캐쉬 공간에서 주소간섭을 일으켰기 때문이다. 반면에 최적 그레인 크기에서는 이러한 주소 간섭이 줄어들기 때문에 다른 그레인 크기들보다 적은 cache conflict miss를 나타내었다. 이러한 캐쉬 실패의 감소는 그림 8(가)에서 나타난 것처럼 최적 그레인 크기가 성긴 그레인 크기(16×16)보다 수행 시간에서 약 17.2%의 성능 향상을 가져오게 했다.



(가) 수행 시간



(나) 캐쉬 실태

그림 8 BMM 프로그램: 최적화 그레인 크기와 다른 그레인 크기들과의 비교

6. 관련연구

캐쉬 성능은 참조의 구역성에 종속된다. 프로그램에서 참조하는 주소들의 연속성이 캐쉬 내에 모두 존재할 경우 캐쉬 성능은 향상된다. 이러한 구역성이 나타나는 프로그램들을 만들기 위하여 많은 최적화 기법들이 소스 코드들의 변경을 통하여 제안되었다[1,2,3]. Lebeck[1]는 프로그램에서 발생되는 캐쉬 실패들의 종류를 구별하고 많은 캐쉬 실패들이 발생하는 데이타 구조 및 소스 라인들을 시각적으로 프로그래머에게 알려줄 수 있는 도구를 제안하였다. 이 연구에서는 프로그램에서 판명된 캐쉬 실패의 원인들을 해결하기 위하여 배열 병합, 순환 구문 병합 및 분리, 데이타구조의 패딩(padding)및 패킹(packing)등의 방법들을 사용하여 향상된 캐쉬 성능을 시험하였다. Lam[3]은 다양한 캐쉬 구조에서 블록 알고리즘을 사용한 행렬 곱셈을 데이타 크기를 변화시키면서 프로그램의 성능 변화를 측정하였다. 이 실험을 근간으로 블록 자체의 주소 간섭을 방지하기 위한 계산된 블록 크기를 제안하였다.

캐쉬 성능은 프로세서 스케줄링 방법에 의해서도 영향을 받는다[4,5]. Tucker[4]은 프로세서 활용도를 높이기 위하여 프로세스 제어 접근방식(process control approach)을 제안하였다. 이 방식은 프로세서들을 여러 그룹으로 분류하고 하나의 프로그램에서 수행할 수 있는 프로세스들의 개수와 실제 할당된 프로세서 개수를 일치시키므로 동기화 지연시간 및 캐쉬 효과를 향상하고자 했다. 병렬 프로그램을 다양한 그레인 크기들에서

수행하여 발생되는 캐쉬 실패 및 프로그램 수행에 필요한 동기화 비용을 측정하는 연구가 있었다[5]. 특히 이 연구에서는 다양한 크기로 쪼개어진 그레인들을 정적 스케줄링과 동적 스케줄링으로 각각 수행하였고 시험된 각각의 병렬 프로그램들의 최고의 성능을 나타낸 스케줄링 방법과 그레인 크기를 제시하였다. 그러나 사용된 그레인 크기들은 단지 데이타를 프로세서 개수의 배수로 나누어 결정하였으므로 시스템 하드웨어 환경을 그레인 선정시 고려하지 않았다.

이전 연구들에서는 병렬 프로그램의 그레인 크기를 선택할 때 캐쉬 메모리의 특성을 고려하지 않았다. 병렬 프로그램에서 선택되는 그레인 크기가 프로세서들 사이에 부하 균형만을 고려하는 경우 프로세서들이 충분한 캐쉬 메모리을 가지고 있어도 하나의 프로세서에 할당된 그레인들 사이의 주소 간섭은 캐쉬 실패를 증가시킨다. 이런 캐쉬 실패는 프로그램 성능에 커다란 영향을 미친다. 따라서 캐쉬 크기와 병렬처리에 참여한 프로세서 개수를 이용한 주소간섭을 배제시킬 수 있는 최적의 그레인 크기에 대한 연구가 필요했다.

7. 결 론

본 논문에서는 직접사상 캐쉬를 사용하는 다중 프로세서 시스템에서 데이타 병렬 프로그램 수행할 때 일어나는 cache conflict miss를 병렬 프로그램의 그레인 크기 관점에서 연구하였다. 데이타 병렬 프로그램에서는 프로세서들의 캐쉬 총합이 프로그램의 데이타 크기보다 클지라도 캐쉬 실패가 발생되며 이들 캐쉬실패에 의하여 소모되는 시간은 프로그램 수행시간의 큰 비중을 차지한다. 이러한 이유는 프로세서들 사이의 부하 균형만을 고려하여 병렬 프로그램의 그레인 크기를 정하기 때문이다. 즉 데이타 배열들이 정해진 그레인 크기에 의해 프로세서들에게 할당되는 경우 각각의 프로세서에 할당된 그레인 들은 캐쉬 주소 공간에서 서로간의 주소 간섭을 발생하여 cache conflict miss를 증가시키기 때문이다. 특히 프로세서 활용성을 증가시키기 위해 작은 그레인 크기를 사용할수록 더욱 많은 작은 그레인들이 각각의 프로세서에 할당되므로 주소 간섭 현상은 더욱 증가하게 된다.

이러한 문제를 해결하기 위하여 본 논문에서는 직접사상 캐쉬 메모리의 특성과 사용되는 프로세서 개수를 바탕으로 그레인 크기를 제안했다. 제안된 최적 그레인 크기는 작은 그레인 크기일지라도 하나의 프로세서에 할당되는 그레인들이 같은 캐쉬 주소 공간에 배치되지 않게 하므로 주소 간섭 현상을 발생하지 않게 하였다. 모의 시험에 의하면 제안된 최적 그레인은 작은 그레인 크기임에도 불구하고 시험된 데이타 병렬 프로그램들에서 cache conflict miss를 현저하게 감소시켰다. 또한 부하가 감소되는 특징을 갖고있는 프로그램에서는 작은 그레인 크기의 본질적인 장점인 프로세서 활용성을 향상시켜 프로그램의 성능을 향상 시켰다.

최적 그레인 크기를 사용하기 위해서는 프로그래머는 캐쉬의 종류, 크기 그리고 사용되는 프로세서의 개수를 파악해야 한다. 그러나 본 논문에서 사용된 데이타 병렬 프로그램들처럼 그레인들이 공간적 및 시간 구역성을 보이는 경우, 최적 그레인 크기는 그레인들 사이의 주소 간섭을 방지하기 때문에 프로그램의 구역성이 손상되지 않고 이용될 수 있게 보장해준다.

참 고 문 헌

[1] Alvin R. Lebeck and David A. Wood, "Cache Profiling and the SPEC Benchmarks: A Case Study," IEEE Computer, Vol.27, No.10, pp 15-26, Oct. 1994.

[2] David A. Patterson, John L. Hennessy, Computer A Quantitative Approach, 2nd Ed., p.405, MORGAN KAUFMANN PUBLISHERS, Inc., 1996.

[3] Monica S. Lam, Edward E. Rothberg and Michael E. Wolf, "The Cache Performance and Optimizations of Blocked Algorithms, "In Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, pp 63-74, April 1991.

[4] A. Tucker and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Multiprocessors," In Proceedings of the 12th ACM Symposium on Operating System Principles, pp 159-166, Dec. 1989.

[5] 정인범, 이준원,"병렬 프로그램에서 성김도와 스케줄링 정책들의 효과,"한국정보과학회 논문지, 제 25권, 제11호, pp. 1214-1224, 1998

[6] J.E. Veenstra and R.J. Fowler, "MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. "In Proceeding of 2nd International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)," pp 201-207, Jan. 1994.

[7] J. Archibald and J-L. Baer, "Cache Coherence Protocols : Evaluation Using a Multiprocessors Simulation Model," ACM Transactions on Computer Systems, Vol 4, No 4, pp 273-298, Nov. 1986.

[8] Vipin Kumar, Ananth Grama, Anshul Gupta and George Karypis, "Introduction to Parallel Comput-

ing(Design and Analysis of Algorithms)." The Benjamin/Cummings Publishing Company, Inc., pp.169, pp.179, 1994.
[9] K. C. Sevcik, "Application Scheduling and Processor Allocation in Multiprogrammed Parallel Processing Systems," Technical Report CSRI-282, Computer Systems Research Institute, University of Toronto, March 1993.
[10] S. H. Chiang, R. K. Mansharamani and M. K. Vernon, "Use of Application Characteristics and Limited Preemption for Run-to-Completion Parallel Processor Scheduling Policies," In Proceeding of the ACM SIGMETRICS Conference, pp 33-44, May 1994
[11] R.E. Kessler and Mark D. Hill, "Page Placement Algorithms for Large Real-Index Caches," ACM Transaction on Computer Systems, Vol 10, No 4, pp 338-359, Nov. 1992

정 인 범
정보과학회논문지 : 시스템 및 이론
제 27 권 제 1 호 참조

공 기 석
1984년 서울대학교 제어계측공학과 학사. 1986년 서울대학교 대학원 컴퓨터공학과 석사. 1999년 한국과학기술원 전산학과 박사. 1986년 ~ 1989년 삼성전자. 1989년 ~ 1992년 삼보컴퓨터(SOLVIT). 1992년 ~ 1994년 EOS Technologies, Inc. 1999년 ~ 현재 한국전자통신연구원 내장형 S/W 연구팀 근무. 관심분야는 운영체계, 실시간 시스템, 컴퓨터구조, 컴퓨터 시스템 성능평가

이 준 원
정보과학회논문지 : 시스템 및 이론
제 27 권 제 1 호 참조