캐쉬 간섭실패의 정적분석 및 프로그램의 수행시간 예측

(Static Analysis of Cache Interference Miss and Prediction of Program Execution Time)

이 건 영 † 정 유 석 † 홍 만 표 ††
(Geon-Young Lee)(Yoo-Suk Jung)(Man-Pyo Hong)

요 약 프로그램의 실행시간은 캐쉬 메모리의 효율적 사용과 밀접한 관계가 있다. 특히 간섭 실패는 프로그램의 성능에 큰 영향을 미치지만 나타나는 형태가 불규칙적이므로 예측하기가 매우 어렵다. 본 논문에서는 직접 사상 캐쉬전략을 사용한 완전 중첩 루프 내 배열의 캐쉬 실패율(cache miss ratio)을 구하는 분석적 모델을 제시한다. 논문에서 제시한 모델은 임의의 캐쉬 위치에 각 배열이 접근한 시간을 기반으로 다음주기에서 캐쉬 실패의 발생 여부를 예측하는데, 간섭으로 발생한 캐쉬 실패 개수에 대해 기존에 제시된 모델보다 더 빠르고 정확한 예측이 가능하다. 특히, 한문장의 수행시간 예측시간은 배열의 크기와 독립적이기 때문에, 전체 프로그램의 수행시간 예측은 배열의 크기 및 문장의 반복 회수 배만큼 빠른 결과를 보여준다. 본 모델은 프로그램의 성능예측 뿐만 아니라 데이타 지역성의 최적화, 캐쉬 구성, 스케쥴링 등에서도 이용 가능하다.

Abstract Program execution time is closely related with the efficient utilization of cache memory. Moreover, the time is greatly affected by the interference miss, but the cache miss behavior of a program is highly unpredictable. In this paper we present novel analytical model of the cache miss ratio consisting mainly of array operation inside nested loops for direct-mapping caches, and using this model we predict the program execution time. Our model predicts whether or not cache miss occurs in next period based on access time of array being loaded in cache. This model is faster and more accurate than previous models. A significant advantage is that the calculation time is proportional to the number of array references in the program. Applications of this model range from performance evaluation and prediction to data locality optimizations.

1. 서 론

한 프로그램의 실행시간은 주로 캐쉬 메모리의 효율적 사용과 밀접한 관계가 있다. 그러나 프로그램의 캐쉬 실패(cache miss)는 예측하기가 매우 힘들다. 왜냐하면 입력된 변수가 조금만 변경되어도 캐쉬 실패의 개수가 매우 많이 변할 수 있기 때문이다.

· 이 논문은 두뇌한국21사업에 의하여 지원되었음.
† 비 회 원 : 아주대학교 정보통신공학과
leegy@madang.ajou.ac.kr
j8508@madang.ajou.ac.kr
†† 종신회원 : 아주대학교 정보통신공학과 교수
mphong@madang.ajou.ac.kr
논문접수 : 2000년 4월 18일
심사완료 : 2000년 9월 26일

일반적으로 캐쉬 실패는 다음과 같이 3가지로 분류되어진다. 첫 번째는 필연적 실패(compulsory miss)이다. 이는 최초 데이타를 접근할 때 필연적으로 발생한 실패로, 데이타 접근 회수로 쉽게 구할 수 있다. 두 번째는 용량 실패(capacity miss)이다. 캐쉬의 고정 엔트리(entry)로 인해, 재 사용될 모든 데이타를 저장할 수 있을 정도의 충분히 큰 캐쉬가 존재하지 않는 경우에 발생한다. 용량 실패는 이제까지 활발하게 연구되어진 분야로, 캐쉬라인을 참조로 비교적 쉽게 예측 및 측정이 가능하다[5,7]. 세 번째로 간섭(interference) 또는 충돌(conflict) 실패이다. 이는 동일한 캐쉬 위치에 여러 데이타가 사상(mapping)되므로 발생하게 된다. 대부분 간섭 실패는 완전 연관 캐쉬(fully associative cache)보다는 이보다 더 성능이 뛰어난 직접 사상 캐쉬(direct

mapping cache)에서 발생하나 아직까지 연구가 미비한 상태이다[3,9,10].

본 논문에서는 직접 사상 캐쉬에서 간섭에 의한 캐쉬 실패율을 계산한다. 간섭 실패는 데이타가 캐쉬에 적재될 때, 해당 캐쉬 위치와 참조된 시간을 알아야 하므로 캐쉬 동작을 예측하거나 측정하기가 매우 어렵다. 예를 들어, 캐쉬의 동일 위치에 사상되는 두 변수 A와 B를 각각 5번씩 참조할 경우, 참조 순서에 따라 캐쉬 실패는 0 ~ 8번까지 발생하게 된다. 참조 순서가 AAAAA BBBBB일 경우 각 A와 B의 첫 번째 참조는 필연적 실패이고, 나머지는 간섭 실패가 발생하지 않는다. 참조 순서가 ABABABABAB일 경우, 처음 A와 B는 필연적 실패이고, 나머지 8개의 참조에서는 모두 간섭 실패가 발생하게 된다.

간섭 현상을 배제한 프로그램의 수행시간은 캐쉬라인의 크기가 클수록 단축된다. 그러나 일반적으로 캐쉬 라인크기가 클수록 더 많은 간섭현상을 유발하게되어 전체 수행시간은 길어지게 된다[11]. 이러한 간섭현상을 미리 예측하여 피할 수 있다면, 캐쉬 라인의 크기를 키울 수 있을 것이며, 상대적으로 프로그램의 수행시간을 단축 할 수 있을 것이다. 우리는 변수가 캐쉬에 적재될 위치의 정보를 이용하여, 참조시간에 대한 사이클을 형성함으로써 각 변수들의 캐쉬 실패율을 알아낸다. 직접 사상 캐쉬에서 메모리의 각 변수는 캐쉬의 유일한 위치에 사상되므로 교체 정책(replacement policy)이 필요 없다. 그러므로 캐쉬에 적재될 각 변수의 위치는 메모리의 위치에 의존하며, 컴파일러나 전처리기에 의해 비교적 쉽게 계산될 수 있다.

본 논문의 구성은 다음과 같다. 다음 장에서는 관련연구에 대해 기술하고, 3장에서는 전반적으로 사용될 개념 및 다양한 주기에 대한 캐쉬 간섭실패의 정적분석에 대해 설명한다. 4장에서는 3장에서 제시한 분석적 모델을 이용하여 프로그램의 전체 수행시간을 예측하는 모델을 제시하고, 제시한 모델의 성능평가는 5장에서 이루어진다. 마지막으로 6장어서는 결론 및 향후과제를 기술한다.

2. 관련연구

Agarwal et al.은 메모리 접근 자취(memory access traces)의 캐쉬 동작에 대한 분석적 동작모델을 제안했다[1]. 이는 캐쉬 동작 예측에 대한 방향을 제시하였으며, 캐쉬 사용의 최적화를 위한 컴파일러에 사용되었다. 그러나 캐쉬 간섭현상의 불안정적 성질까지를 찾는데는 한계가 있었다. Coleman et al.은 간섭현상이 최소화 될 수 있는 블록의 크기를 선택하는 방법론을 제시한다[11]. 그러나 이 방법 또한 모든 형태의 알고리즘을 모델링 할 수 있는 일반성이나 정확성에 있어 한계를 가진다. 다른 시도로 Temam et al.은 직접사상 캐쉬에서 완전 중첩(perfectly nested) 루프내의 간섭실패(interference miss)를 계산할 수 있는 모델을 제시한다[10]. 이 모델은 비교적 정확하며 빠른 계산이 가능하지만 배열의 접근이 순차적으로 연속되어야 한다는 등의 가정이 존재한다. Harper et al.은 Temam et al.의 모델에서 몇 가지 가정들을 파괴하는 모델을 제시한다[4]. Temam et al. 과 Harper et al.은 배열의 참조 패턴이 동일하며, 캐쉬에서 항상 일정 거리만큼 떨어져 있는 변수들의 모임을 "Translation Group(TG)"이라 하고, 동일 TG내에 존재한 변수들간의 관계를 내부(Internal), 다른 TG내에 포함된 변수들과의 관계를 외부(External)라 정의한다. 이들 모델에서 간섭 실패에 의한 캐쉬 실패는 내부에서 캐쉬 실패를 구한 후, 다시 캐쉬 실패가 발생하지 않는 변수들에 대해 외부에서 구하게 된다. 그러나 이들 정의에 의하면, 동일 배열이 인덱스로 가진 루프레벨이 다르면(예: A(i,j), A(c,j)) 다른 TG에 속하게 되며, 다른 TG의 변수들은 서로에게 간섭 실패를 야기하는 변수로 취급된다. 그러나 비록 다른 TG에 속한 변수일지라도 동일 이름의 변수는 서로에게 캐쉬 적중을 일으킬 수 있다.

본 논문은 Temam et al. 및 Harper et al.에서 사용한 재사용 집합(Reuse Set)이나 간섭 실패(Interference Miss)등 기본 개념은 그대로 사용하되, 필요에 따라 재정의함으로써, 기술된 단점들을 보완할 수 있는 새로운 접근 방법을 제시한다. 이후부터는 Temam et al.과 Harper et al.이 제시한 모델을 편의상 기존모델로 명명한다.

3. 캐쉬 실패개수의 정적분석

3.1 기본개념

본 논문에서 구하고자하는 캐쉬 실패율은 직접 사상 캐쉬전략을 사용한 시스템에서 완전 중첩된 루프에 제한한다. 또한 본 논문의 관심사는 용량 실패 보단 간섭 실패에 초점을 두고 있으므로, 캐쉬의 크기는 충분히 크다고 가정한다.

• **배열** : 완전 중첩된 루프에서 j_n은 가장 안쪽(inner most) 루프를, j_1는 가장 바깥쪽(outermost) 루프를 나타내고, 안쪽으로 갈수록 순차적으로 증가하는 형태를 가진다고 할 때, 배열 v는 $v(\alpha_1 j_{r_1}+\beta_1, \alpha_2 j_{r_2}+\beta_2, \ldots\ldots, \alpha_m j_{r_m}+\beta_m)$형태를 취한다. 이때, α_i, β_i, r_i은 상수이다.

• **배열의 메모리에서 위치** : 배열 v의 메모리에서의

위치는 각 인덱스와 해당 차수 그리고 기본 주소(base address)의 연산으로 구할 수 있다. 즉, v_0가 배열 v의 기본 주소를 나타낼 때 위치는 다음 식과 같다. 이때, M_i는 i차원의 최대크기를 나타낸다.

$$\Phi(v)=(\alpha_1 j_{r_1}+\beta_1)(M_2\times..\times M_m)+\ldots+(\alpha_{m-1} j_{r_{m-1}}+\beta_{m-1})M_m +(\alpha_m j_{r_m}+\beta_m)$$

$$= v_0+\sum_{i=1}^{m-1}((\alpha_i j_{r_i}+\beta_i)\cdot\prod_{j=i+1}^{m} M_j)+(\alpha_m j_{r_m}+\beta_m)$$

• **캐쉬에 적재될 위치** : 우리는 한 변수의 $\Phi(v)$로부터 캐쉬에 적재 될 위치($P(v)$)를 알 수 있다. 직접사상 캐쉬에서, 메모리의 고정된 위치를 차지하는 변수는 항상 동일 캐쉬 위치에 사상되므로 $P(v)$는 $\Phi(v)$를 캐쉬크기로 모듈러 연산함으로써 계산할 수 있다. 즉, $P(v)=\Phi(v)\ mod\ C_S$ 다.

• **적재된 두 변수 간 거리** : 재사용 집합의 각 원소의 재사용 가능여부는 다른 변수와의 의존관계에 의해 결정된다. 참조 변수의 캐쉬 실패의 개수는 캐쉬에 적재된 다른 변수의 위치에 좌우하게 되는데, 우리는 두 변수 v와 u 의 적재된 거리를 $\delta_{v,u}=P(v)-P(u)$로 나타낸다.

• **재사용 루프레벨** : 캐쉬에 적재된 배열의 각 원소는 특정 루프에 대해 다시 쓰여질 수 있는데, 이 루프를 재사용 루프레벨(Reuse Loop Level: RLL)이라 부른다. 재사용 루프레벨은 참조변수에 사용된 루프레벨의 최소값보다 작은값 중 최대값이 선택된다. 즉, 사용된 루프레벨의 최소값보다 1작은 값이 된다. 그러므로 참조변수 v에 대한 재사용 루프 레벨(l_v)은 재사용 집합의 크기가 최대가 되는 루프가 선택된다.

• **재사용 집합** : 참조 변수 v는 재사용 루프레벨(l_v)을 기준으로 각 원소를 재 사용하게 되는데, 이때 사용된 배열 v의 원소들을 재사용 집합(RS(v))이라 하며, $RS(v)=\{\ v(\alpha_1 j_{r_1}+\beta_1, \alpha_2 j_{r_2}+\beta_2, \ldots, \alpha_m j_{r_m}+\beta_m)\mid 0\le j_i\le(N_i-1)\}$로 표현한다. 그리고 재사용 집합의 크기는 $|RS(v)|$로 표현하며, 재사용 집합의 원소 개수를 나타낸다.

```
for j1 =0 to N1 -1
  for j2 =0 to N2 -1
    for j3 =0 to N3 -1
      f ( A( j3 ), B( j2 ) )
```

프로그램 1

예를 들어, 프로그램 1에서 A(j_3)의 "재사용 루프레벨($l_{A(j_3)}$)"은 2(또는 j_2)이고, 재사용 집합은 {A(0), A(1), ..., A(N_3-1)}가 되며, $|RS(A(j_3))|$은 N_3가 된다. B(j_2)에서는 사용된 루프레벨의 최소값이 2(또는 j_2)이므로 $l_{B(j_2)}$은 1(or j_1)이 된다. 마찬가지로 재사용 집합은 {B(0), B(1), ..., B(N_2-1)}이며, $|RS(B(j_2))|$은 N_2이다.

• **자기/그룹 의존관계** : 완전 중첩된 루프 내에 사용된 변수들 중, 변수 명과 차수가 같은 경우 동일 배열을 나타낸다. 동일 배열을 나타내는 모든 변수들은 하나의 그룹(group)을 형성하며, 같은 그룹내의 변수들은 서로의 재사용 집합에 영향을 미치게 된다. 이때, 동일 그룹에 속한 변수들은 그룹 의존(Group-Dependence)관계에 있다고 말하며, 참조변수 v에 대해 $GD(v)$로 표기한다. $|GD(v)|$는 v와 그룹 의존관계에 있는 변수들의 개수를 나타낸다. 만일 하나의 그룹을 형성하는 변수가 자기 자신뿐일 경우 자기 의존(Self-Dependence)관계라 부르고, v에 대해 SD(v)로 표기한다. 예를 들어, f ($A(j_4, j_3)$, $A(j_4, j_3+2)$, $A(j_4, j_2)$, $B(j_4, j_3)$, $C(j_4)$)에서 $A(j_4, j_3)$, $A(j_4, j_3+2)$, $A(j_4, j_2)$은 하나의 GD(A)를 형성하고, $B(j_4, j_3)$와 $C(j_4)$는 각기 다른 그룹인 GD(B), GD(C)를 형성한다. 이들 그룹 중 GD(B), GD(C)는 그룹에 포함된 변수의 개수가 1개이므로 자기 의존관계만이 존재하여 SD(B), SD(C)로 표기한다.

• **간섭집합** : 재사용 집합의 각 원소가 다시 사용되어 질 때까지의 거리를 "주기"라 할 때, 일반적으로 주기가 클수록 각 원소는 다시 사용되기 전에 캐쉬에서 물러날 확률이 높게 된다. 재 사용집합은 변수들의 여러 요인으로 인해 다음 주기에서 재사용이 간섭된다. 재 사용집합을 간섭하는 모든 변수를 우리는 간섭 집합(Interference Set)이라 한다. 변수 v의 간섭 집합인 IS(v)는 용량 실패(capacity miss)로 인해 자신의 RS(v) 일부가 될 수 있으며, 다른 변수들이 될 수도 있다. 그러나 앞서 가정했듯이 캐쉬의 크기는 충분히 크므로 자신에 의한 캐쉬 실패는 배제되며, $IS(v)=\{u\mid u\notin GD(v)\}$로 표기한다. 그리고 $|IS(v)|$는 간섭 집합의 원소 개수를 나타낸다.

3.2 한 주기에 대한 캐쉬실패 분석

참조 변수 v의 RS(v)와 IS(v)에 속한 원소들이 캐쉬에 적재될 때, 두 집합 중 일부는 서로 같은 캐쉬 위치에 적재되어 캐쉬 실패가 발생할 수 있다. 즉, $\Phi(v)$에

저장된 변수 v는 캐쉬에서 $P(v)$에 적재된다. 완전 중첩 루프 내에 존재한 모든 변수가 캐쉬에 적재될 경우, 어떤 캐쉬구간은 서로 중첩이 일어나는데 이를 교차구간(intersection)이라 부른다. 교차구간에 포함된 각 변수의 원소들은 캐쉬 실패가 발생할 확률이 매우 높게 된다. 그러므로 캐쉬 실패의 개수를 구하는 문제는 교차구간의 크기를 구하는 문제와 밀접한 관계가 있다.

캐쉬 실패의 개수를 구하기 위해서는 먼저 각 구간에 대해 독립적으로 캐쉬 실패의 개수를 구한 후, 각 구간에서 구해진 개수를 더한다. 각 구간에서 캐쉬 실패의 개수는 임의의 캐쉬 위치에서 각 변수들의 접근 시간을 측정함으로써 쉽게 알 수 있다. 본 논문에서는 임의의 캐쉬 위치에 변수 v가 처음 접근한 시간을 처음 사용시간이라 부르며, 다음과 같이 계산한다. 루프 j_i 의 선행차수(leading dimension)를 $LD(j_i)=\prod_{k=i+1}^{n} N_k$ 라 할 때, 변수 v의 처음 사용시간 $T(v)=\sum_{i=r_v+1}^{n}((\alpha_k j_{r_k}+\beta_k)\cdot LD(j_i))$ *where*, $r_k=i$ 이다.

임의의 캐쉬 위치에서 각 변수의 사용시간은 해당 캐쉬 위치를 접근하는 시간을 나타내며, 그 순서는 하나의 사슬(chain)을 형성한다. 이 사슬은 본 논문에서 간섭에 의한 캐쉬 실패의 개수를 구하기 위한 기본구조로, 그림 1과 같이 형성된 사슬에서 처음에 위치한 변수는 다음 주기에서 마지막에 위치한 변수를 선행 변수로 가지므로 사이클(cycle)을 형성하게 된다. 이를 "Use Time History Cycle(UTHC)"이라 한다.

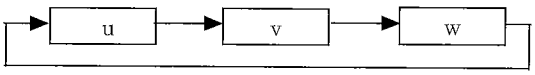

그림 1 UTHC

만일 주기가 같은 변수들이 임의의 캐쉬 위치 i 에서 UTHC가 $u \rightarrow v \rightarrow w$일 때, 변수 v 직전에 있는 u가 v의 간섭 집합이면 v는 i에서 캐쉬 실패가 되지만, u와 v가 동일 그룹에 속하면 v는 캐쉬 적중이 된다. 마찬가지로 w와 u가 동일 그룹에 속하면 u는 캐쉬 적중이 되지만 그렇지 않는 경우 캐쉬 실패가 된다. 우리는 임의의 한 구간에서 캐쉬 실패의 개수를 구하는 시간을 단축하기 위해서 구간내의 한 캐쉬 위치(일반적으로 구간의 첫 번째 캐쉬 위치)에서 각 변수들의 사용시간을 구한 후, UTHC를 형성한다. 직관적으로 알 수 있듯이 구간 내 모든 변수는 순차적이기 때문에 형성된 UTHC는 모든 구간에 동일하게 적용된다. 그러므로 한 구간 내에서 발생한 캐쉬 실패의 개수는 구간의 크기에 무관한 상수시간에 계산이 가능하다.

```
for j1 =0 to N1 -1
  for j2 =0 to N2 -1
    for j3 =0 to N3 -1
      f( A(j2, j3 ), B(j2, j3 ), C(j2, j3 ), A(j2, +2 j3 ))
```

프로그램 2

예를 들어, 그룹의존관계가 있는 프로그램 2에서 UTHC를 분석해 보자. 프로그램 2에서 $A(j_2, j_3)$와 $A(j_2+2, j_3)$는 그룹 의존관계에 있으며, B와 C는 자기 의존관계에 있다. 변수 B, C, A(j_2+2, j_3))가 A (j_2, j_3)의 시작위치로부터 거리 크기가 $\delta_{AA'} < \delta_{AB} < \delta_{AC}$이고, 캐쉬에 적재된 각 변수의 위치가 그림 2와 같다고 하자.

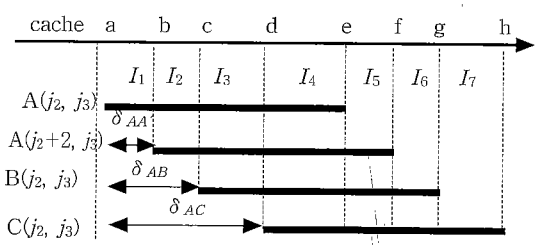

그림 2 캐쉬의 배치

이때, 각 변수의 사용시간은 t, $t-\delta_{AA'}$, $t-\delta_{AB}$, $t-\delta_{AC}$가 되며, UTHC는 $C(j_2, j_3) \rightarrow B(j_2, j_3) \rightarrow A(j_2+2, j_3) \rightarrow A(j_2, j_3)$가 된다. UTHC에서 알 수 있듯이, A(j_2, j_3)의 이전 사용 변수는 그룹의존관계에 있는 A(j_2+2, j_3)이므로, 구간 I_4에서는 A(j_2, j_3)는 모두 캐쉬 적중이 된다.

3.3 다양한 주기에 대한 캐쉬실패 분석

경우1 : 자기 의존관계(Self-Dependence)

자기의존관계에 있는 변수의 캐쉬 실패의 개수는 비교적 쉽게 구할 수 있다. 임의의 한 교차구간의 UTHC에서, 참조변수 v는 직전에 위치한 변수가 IS(v)의 원소이면 항상 캐쉬 실패가 된다. 그리고 각 변수의 RLL이 다르다면, 전체 루프에 대한 캐쉬 실패의 개수는 각 RLL을 고려한 계산이 이루어져야 한다. 일반적으로, 재사용의 주기가 클수록 캐쉬에서 물러날 확률이 더 높게 된다. 왜냐하면 재 사용되기 전에 주기가 작은 변수들이 캐쉬에 적재되어지기 때문이다. 이런 특성을 이용하여

기존모델에서는 가장 작은 주기를 가진 간섭집합을 v의 대표적인 간섭변수로 규정하고 나머지 변수들은 무시한다[4,10]. 그러나 경우에 따라서는 고려되지 않은 간섭변수는 대표로 지정된 간섭변수만큼의 캐쉬실패의 개수를 발생 할 수도 있다. 자기의존관계에서 캐쉬실패 개수를 구하는 방법은 다음 장에서 설명될 그룹의존관계와 많은 부분이 중복되기 때문에 자세한 설명은 다음 장으로 넘기고, 본 장에서는 반복회수 계산에 대한 설명만 기술한다.

사이클을 배제한 UTHC에서, v보다 앞에 위치한 모든 변수들의 집합을 Prev(v)(단, v가 가장 처음에 존재한 경우, 사이클을 고려한 UTHC가 됨)라 하자. Prev(v)에서 v의 처음사용시간과 가장 근접한 변수가 간섭 변수 u이면, 전체 루프에 대한 v의 캐쉬 실패의 개는 $\prod_{i=1}^{min(l_v, l_u)} N_i$가 된다.

예를 들어, 한 교차구간의 임의의 캐쉬 위치에 두 변수 $A(j_3, j_4, j_5)$, $B(j_4, j_5)$가 있다고 가정하면, $l_A=2$, $l_B=3$이 된다. 만일 UTHC가 A→B이면, $Prev(B(j_4, j_5))$는 $A(j_3, j_4, j_5)$이다. 그러므로 B는 해당위치에서 $\prod_{i=1}^{2=min(l_v, l_u)} N_i$번 캐쉬 실패가 발생한다.

임의의 교차구간 I_i에서, 한 캐쉬 위치에서 구한 캐쉬 실패의 개수는 모든 구간에 대해 동일하게 적용된다. 그러므로 I_i에서 발생하게 될 v의 간섭 실패의 개수는 $M_{Intf}(v) = S(I_i) \times \prod_{i=1}^{l} N_i$ 가 된다. 이때 $S(I_i)$는 I_i의 크기를 나타낸다.

경우 2 : 그룹 의존관계(Group-Dependence)

변수 v는 다음 주기에서 캐쉬에 적재된 자신 또는 그룹(GD(v))의 재사용 집합을 다시 사용하게되므로, v의 캐쉬 실패 개수는 GD(v)와 IS(v)의 주기에 따라 변하게 된다. 일반적으로 참조 변수 v의 재사용에 가장 많은 영향을 미치는 변수는 가장 작은 주기를 가진 변수이거나 사용시간이 T(v)에 가장 가까운 변수가 된다.

캐쉬 실패의 개수를 구하기 위해서는 앞서 설명했듯이 한 교차구간의 모든 캐쉬 위치에서는 동일한 형태의 UTHC를 가져야 한다. 만일 한 교차구간에서 서로 다른 형태의 UTHC를 가질 경우, 그 구간은 다시 동일한 UTHC를 가지는 여러 개의 교차구간으로 나뉘게 된다. 그러므로 한 교차구간의 모든 캐쉬 위치에서는 동일한 형태의 UTHC를 가지게 된다.

각기 다른 주기를 가진 간섭집합의 변수나 그룹의존관계의 변수들을 고려한 캐쉬 실패의 개수를 구하기 위해서는 앞서 설명된 UTHC를 새로운 방법으로 형성한다. 주기가 작은 변수는 처음 접근 시간이 늦다하더라도, 다음 주기에서는 충분히 다른 변수의 캐쉬 실패나 적중을 일으킬 수 있다. 그러므로 비록 각 변수의 처음 접근시간이 참조변수 v보다 늦다하더라도 UTHC에서는 모두 v앞에 위치시킨다. 앞으로 설명될 내용의 표기의 용이성을 위해 다음을 정의한다.

• $RT(v)_{T(i)}$: 임의의 캐쉬위치에서 v의 접근시간의 주기가 T(i)보다 큰 값 중 최소 주기시간을 나타낸다. v의 처음 접근시간 T(v), i의 처음 접근시간 T(i)에 대해, T(i)보다 큰 주기시간은 $t > \frac{T(i)-T(v)}{|RS(v)|}$ 을 만족하는 모든 정수 t이다. 그러므로 최소주기시간 $RT(v)_{T(i)} = T(v) + |RS(v)| \times \lfloor \frac{T(i)-T(v)}{|RS(v)|} \rfloor$ 가 된다.

• $LT(v)_{T(i)}$: 임의의 캐쉬위치에서 v의 접근시간의 주기가 T(i)보다 작은 값 중 최대 주기시간을 나타낸다. T(i)보다 작은 주기시간은 $t < \frac{T(i)-T(v)}{|RS(v)|}$ 을 만족하는 모든 정수 t이므로 $LT(v)_{T(i)} = T(v) + |RS(v)| \times \lfloor \frac{T(i)-T(v)}{|RS(v)|} \rfloor$ 가 된다.

• UTHC(i) : UTHC에서 변수 i의 위치를 나타내며, 앞에 위치할 수록 작은 값을 나타낸다. 그러므로 UTHC의 일부가 i -> j일 경우, UTHC(i) < UTHC(j) 가 된다.

임의의 한 교차구간에 포함된 모든 변수의 집합을 VAR_{set}라 할 때, 새로운 UTHC는 다음과 같이 형성된다. 먼저 UTHC에 참조변수 v를 넣는 후 VAR_{set}가 ∅이 될 때까지 다음 2단계를 반복한다. 첫째, VAR_{set}에 속한 각 변수에 대해, 그 사용시간이 v의 사용시간의 주기와 최소($min(RT(v)_{T(u)} - LT(u)_{(RT(v)_{T(u)})})$)인 변수 u를 선택한다. 그림 3에서 보는 것처럼, u는 v의 재사용시간에 가장 근접한 시간을 가진 변수이다.

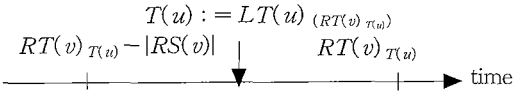

그림 3 $RT(v)_{T(u)}$와 $LT(u)_{(RT(v)_{T(u)})}$의 모습

둘째, 선택된 변수 u를 UTHC의 처음 위치에 넣은

후, u와 $\{ l_w \le l_u \ and \ RT(v)_{T(u)} = RT(v)_{T(w)} \}$을 만족하는 변수 w를 VAR_{set}에서 제거한다. 제거된 변수 w는 u보다 주기가 크거나 같으며 동일한 v의 재사용시간에 영향을 미친 변수다. 비록 w는 v의 재사용시간에 영향을 미치지만 최종적으로 영향을 미치는 변수는 u가 되므로 고려할 필요가 없는 것이다.

다음으로, 새로 형성된 UTHC를 이용하여 참조 변수 v의 캐쉬 간섭실패의 개수를 구하는 방법에 대해 살펴보자. 임의의 캐쉬위치에서 v의 재사용을 간섭하는 변수는 v의 재사용시간보다 먼저 해당캐쉬위치를 접근한 IS(v)의 변수들로, 그 사용시간이 v의 사용시간에 가장 근접한 하나의 변수가 된다. 그러므로 참조변수 v의 캐쉬실패 개수를 구하기 위해 먼저, UTHC에서 v에 가장 가까운 IS(v)의 변수(u)를 선택한다. 그리고 u와 v의 재사용 루프레벨 중 작은 루프레벨을 선택하는데, 이 루프레벨은 자기의존관계에서 설명했듯이 전체 루프에 대한 캐쉬 실패의 개수를 예측하는데 사용된다. 그러나 u와 v의 사용시간 사이에 그룹의존관계의 변수의 존재여부에 따라 캐쉬 실패의 개수는 다르다. 만일, UTHC에서 u와 v사이에 GD(v)의 변수가 존재하지 않으면, 각 변수의 주기에 상관없이 GD(v)에 속한 변수는 u와 v의 사용시간 사이에 결코 나타나지 않으므로 u는 v의 캐쉬 실패를 야기할 수 있다. 이때 u로 인해 발생하게 될 v의 캐쉬 실패 개수는 UTHC에서 u와 v사이에 존재한 IS(v)의 영향만 받게된다. 그림 4, 5에서 보는 것처럼, u와 v의 임의의 사용시간 사이에 IS(v)에 속한 변수 i의 사용시간이 존재하면, i에 의한 v의 캐쉬실패의 개수는 이미 계산된 상태가 된다. 그러므로 u에 의한 캐쉬실패의 개수에서 i에 의해 발생한 총 캐쉬 실패의 개수를 뺀 값만큼이 추가적으로 발생하게 될 캐쉬실패의 개수가 된다.

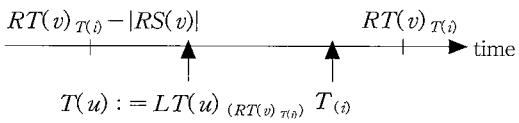

그림 4 u의 사용시간 다음에 T(i) 존재

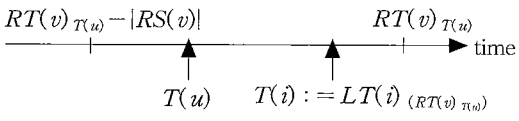

그림 5 T(u) 다음에 i의 사용시간 존재

한편, u와 참조변수 v사이에 GD(v)의 변수가 존재할 경우, UTHC(u)와 UTHC(v)사이에 존재한 GD(v)의 각 변수들에 대해, 주기가 가장 작은 변수(g)를 선택한다. 그림 6, 7에서 보는 것처럼, v의 캐쉬 실패의 개수는 u에 의한 캐쉬실패의 개수 중 g에 의한 캐쉬 적중의 개수를 제외한 나머지가 된다.

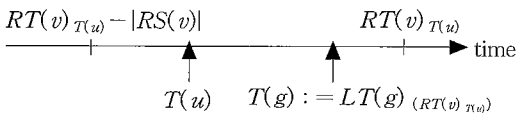

그림 6 T(u)다음에 g의 사용시간 존재

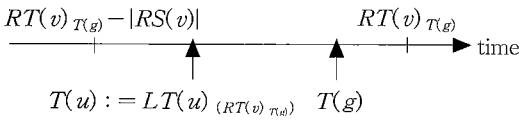

그림 7 u의 사용시간 다음에 T(g) 존재

u에 의한 v의 캐쉬 실패 개수의 계산이 끝나면, UTHC(u)보다 앞에 위치한 IS(v)의 변수들 중 불필요한 IS(v)의 변수들을 UTHC에서 제거한다. 이때 제거된 변수는 주기가 u보다 크고 u가 간섭하는 v의 사용시간을 동일하게 간섭하는 변수들이 된다. 불필요한 IS(v)의 변수들이 제거되면, 다시 UTHC에서 u 다음으로 v에 가까운 IS(v)의 변수를 선택하여 검사를 반복한다.

필연적실패의 개수는 $g \in GD(v), u \in IS(v), l_v > l_u$ 을 만족하는 모든 변수 g와 u에 대해, $T(v) < T(g)$와 $T(v) < T(u)$을 동시에 만족하면 $S(I_i)$가 된다. 왜냐하면, g보다 늦게 사용된 v는 g가 적재한 변수를 참조하기 때문에 캐쉬 적중이 되며, u보다 늦게 사용되어 발생한 v의 캐쉬 실패는 모두 간섭실패의 개수에 포함되어 계산되기 때문에 필연적 실패의 개수는 발생하지 않는다.

알고리즘 1은 임의의 한 교차구간에서 자기 및 그룹의존관계에 있는 참조변수 v의 캐쉬 실패의 개수를 구하는 알고리즘이다. 이때, M_{Intf}는 간섭실패의 개수, M_{Comp}는 필연적 실패의 개수, 그리고 M_{Total}은 전체 캐쉬 실패의 개수를 나타낸다.

Algorithm CacheMiss (Var_info, v)
Input : 각 변수의 정보(Var_info), 캐쉬실패를 구하고자하는 변수(v)
Output : v의 캐쉬 실패 개수
begin

$UTHC := NULL$
VAR_{set} := {한 교차구간에 포함된 모든 변수}
Insert(v) into UTHC

while ($VAR_{set} \neq \emptyset$)
$u := min\{i \mid RT(v)_{T(i)} - LT(i)_{(RT(v)\ T(u))}, i \in VAR_{set}\}$
Insert(u) into UTHC at the first position
Remove(u and w | $l_w \leq l_u$ *and* $RT(v)_{T(u)} = RT(v)_{T(w)}$,
$w \in VAR_{set}$)
from VAR_{set}
end while

while($\forall_i$, $i \in IS(v)$)
$u :=$ *Select the nearest* $\{i \mid i \in IS(v)$ *and* $UTHC(i) < UTHC(u)\}$ *to* v
$l := \{ RLL \mid min(l_u, l_v) \}$
if ($\nexists . \{i \mid UTHC(u) < UTHC(i) < UTHC(v), i \in GD(v)\}$)
if ($\exists . \{i \mid [(RT(v)_{T(i)} - |RS(v)|) < LT(u)_{(RT(v)\ T(u))} < T(i)]$
or $[(T(u) < LT(i)_{(RT(v)\ T(u))} < RT(v)_{T(u)}]$
$, i \in IS(v)$ *and* $UTHC(u) < UTHC(i) \}$)
$M_{Intf}(v)$ += $max(0, \prod_{i=1}^{l} N_i - \prod_{i=1}^{l} N_i)$
else
$M_{Intf}(v)$ += $\prod_{i=1}^{l} N_i$
else
$g := max\{RLL(i) \mid UTHC(u) < UTHC(i) < UTHC(v),$
$i \in GD(v)\}$
if ($\nexists . \{i \mid [T(u) < LT(g)_{(RT(v)\ T(u))} < RT(v)_{T(u)}]$
or $[LT(u)_{(RT(v)\ T(u))} < T(g) < RT(v)_{T(g)}]$, $i \in IS(v)\}$)
$M_{Intf}(v) += max(0, \prod_{i=1}^{min(l_u, l_v)} N_i - \prod_{i=1}^{min(l_u, l_i)} N_i)$
Remove($i \mid l_i \leq l_u$ *and* $T(i) < LT(u)_{(RT(v)\ T(u))}$, $i \in IS(v)$ *and*
$UTHC(i) < UTHC(u)$
from UTHC
end while
$u \in GD(v), w \in IS(v), l_v > l_w$을 만족하는 $\forall_{u,w}$ 대해, $T(v) < T(u)$ *and* $T(v) < T(w)$ 을 만족하면
$M_{comp}(v) := S(I_i)$
$M_{Total}(v) := M_{Intf}(v) \times S(I_i) + M_{Comp}(v)$
end

알고리즘 1 자기 및 그룹 의존관계에서 캐쉬 실패개수를 구하는 알고리즘

앞에서 가정했듯이 루프 내 모든 배열의 각 원소접근 시간이 증분 1로 순차적일 때, 한 구간 내 모든 캐쉬 위치는 동일한 UTHC를 갖게 된다. 만일 변수 v가 한 캐쉬위치에서 캐쉬 실패를 발생하면 전체 루프에 대해서 반복회수를 곱 한만큼이 되며, 한 교차구간에서는 구간의 크기를 다시 곱 한만큼이 된다. 그러므로 한 교차구간에서 참조변수 v의 캐쉬 실패의 개수는 상수 시간에 가능하며, 예측시간은 구간의 크기와는 무관하게 변수의 개수에 비례한다.

4. 프로그램의 수행시간 예측

4.1 한 변수의 캐쉬 실패율(Cache miss ratio)

캐쉬 실패의 개수는 이미 설명했듯이 각 변수의 교차구간을 먼저 구한 후, 각 변수의 캐쉬 접근 시간에 대한 UTHC를 이용한다. 캐쉬 실패율은 각 구간에서 발생한 캐쉬 실패 개수의 합을 총 사용회수로 나누어 구할 수 있다. 3장에서 구한, 임의의 한 구간 I_i에서 v의 캐쉬 실패 개수를 $M_{Total}(v)_i$라 하면, 전 교차구간에서 발생한 캐쉬 실패 개수($T_{miss}(v)$)는 이들의 합이 된다. v의 총 사용회수($T_{use}(v)$)는 재사용 집합의 모든 원소가 전체 루프에 대해 사용된 회수를 나타낸다. 즉, 교차구간의 개수가 k라 할 때, v의 총 사용 회수는 $T_{use}(v) = |RS(v)| \times \prod_{i=1}^{n} \{N_i \mid r_i = 0\}$가 되며, 총 캐쉬실패 개수는 $T_{miss}(v) = \sum_{i=1}^{k} M_{Total}(v)_i$ 가 된다. 그러므로 변수 v 의 캐쉬 실패율(m_v)은 $m_v = \frac{T_{miss}(v)}{T_{use}(v)}$ 이 된다. 구해진 캐쉬 실패율은 전체 프로그램의 수행시간을 예측에 사용된다.

4.2 프로그램의 수행시간 예측

프로그램의 수행시간에 영향을 미치는 요인은 크게 캐쉬의 실패/적중, 통신시간과 연산시간의 중복(overlapping), 사슬(chaining)등과 같이 다양하다. 연산시간은 프로세서와 계층적으로 구성된 메모리의 구조 및 속도, 용량 등에 의해서 결정되어진다. 기존의 알고리즘의 복잡도는 연산 유닛의 메모리 접근이 단위 시간 내에 이루어진다는 가정 하에서 이루어 졌으나 실제로 수행시간은 연산 시간뿐만 아니라 메모리 접근 시간에도 큰 영향을 받는다. 그러나 캐쉬 실패/적중이나 페이지 폴트(Page fault)의 빈도 등과 같은 메모리 접근 시간은 메모리의 용량이나 프로그램의 작성 방법 등과 같은 여러 요인으로 인하여 정확하게 예측하기는 매우 어렵다[2,8,12]. 본 논문에서는 계산의 간결화를 위해 캐쉬의 영향만을 고려한다.

캐쉬 적중으로 캐쉬에서 레지스터로 데이타 전송시간을 t_{Hcache}, 캐쉬 실패로 인한 데이타 전송시간을 t_{Mcache}, 임의의 변수 i 에 대한 캐시 실패율을 m_i , 연산 유닛에서 처리시간을 $T(ALU)$, 참조 변수의 개수

를 n_{var} 라 할 때 한 문장의 연산시간($E(S_t)$)은 다음과 같다.

$$E(S_t) = T(ALU) + \sum_{i=1}^{n_{var}} ((1 - m_i) \cdot t_{Hcache} + m_i \cdot t_{Mcache}).$$

그러므로, 한 문장 S_t의 반복 회수를 $I(S_t)$라하고, 전체 문장의 개수를 n_{stat}라 할 때, 프로그램의 총 수행시간은 $E(P) = \sum_{t=1}^{n_{stat}} (E(S_t) \times I(S_t))$가 된다.

5. 성능 평가

본 장에서는, 앞에서 기술한 알고리즘을 이용하여 각 변수의 캐쉬실패의 개수를 예측한 후, 그 결과를 시뮬레이션 및 기존논문과 비교 분석한다.

성능평가를 위해 사용된 두개의 프로그램은 각 배열이 3차 이하로 구성되어 있으며, 배열의 원소는 순차적으로 접근한다. 그리고 3장에서 기술한바와 같이 δ 는 두 변수 사이의 거리를 나타내는 것으로, 우리는 각 변수의 교차구간이 최대가 될 수 있는 임의의 값을 선택했다.

프로그램 3은 3개의 변수가 그룹 및 자기의존관계에 있고, 배열의 차수가 2, 3차로 구성된 프로그램이다. 그림 8에서 알 수 있듯이 본 논문에서 제한된 모델은 각 변수의 캐쉬 실패개수가 시뮬레이션과 동일함을 보여준다. 반면, 기존논문은 그 결과가 큰 차이를 나타내는데, 이는 비록 배열의 차수가 같다 할지라도 첨자로 사용된 루프레벨이 다르면 그룹의존관계에 있는 변수들의 정보를 간과하기 때문이다.

프로그램 4는 Perfect Club 벤치마크 프로그램인 ARC2D의 변형 프로그램이다.

```
#define N 100
for j1=0 to N-1
   for j2=0 to N-1
      for j3=0 to N-1
        f( A(j1, j2, j3 ), B(j2, j3 ), A(0, 4, j3 ))
```

프로그램 3

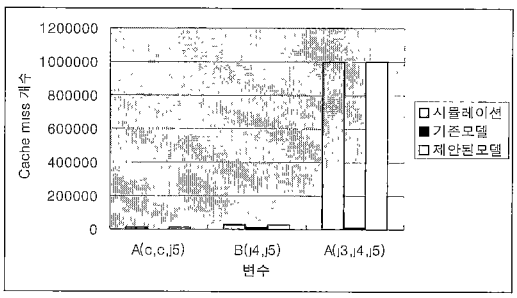

그림 8 프로그램 3의 캐쉬실패 개수

```
#define N 100
c = N-1
for j1 =0 to N-1
 for j2 =0 to N-1
   for j3 =0 to N-1
      LDD = -LDA(j3) × U(j2+2, j3 -LDB(j3)×U(j2+1,j3)
      LDA( j3 ) = LDB( j3 )
      LDB( j3 ) = LDD
      LDS( j3 ) = LDS( j3 ) + LDD × U( j2 , j3 )
      U(c, j3 ) = U(c, j3 ) + LDD × U( j2 , j3 )
      F( j2 , j3 ) = F( j2 , j3 ) - LDD × F( j2 , j3 )
```

프로그램 4

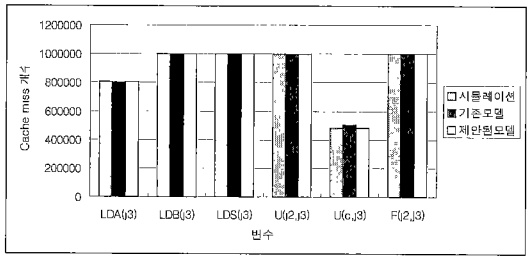

그림 9 프로그램 4의 캐쉬실패의 개수

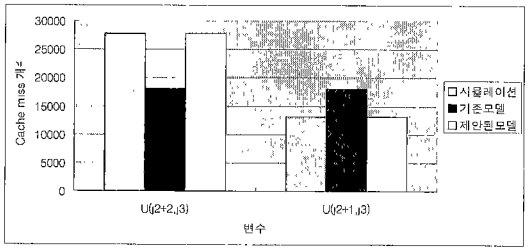

그림 10 프로그램 4의 캐쉬실패의 개수

그림 9에 포함된 변수들은 두 모델의 예측된 캐쉬 실패의 개수는 시뮬레이션 결과와 거의 일치한다. 그러나 그림 10에서 포함된 두 변수의 예측된 캐쉬실패의 개수는 두 모델간에 큰 차이가 있다. 그 이유는 앞서도 설명했듯이 기존모델은 주기가 같지 않는 변수는 그룹의존관계가 존재하지 않는 것으로 보고있다. 이로 인해 다른 변수에 미치게 될 캐쉬 실패 및 적중의 개수를 고려하지 못하고 있다. 또한 참조변수 v에 영향을 미치는 간섭변수는, 비록 여러 간섭변수들이 v의 각기 다른 사용시간에 간섭을 일으키더라도, 가장 작은 주기를 가진 간섭변수만이 고려되고 나머지 변수들은 무시되기 때문이다. 프로그램 4의 시뮬레이션 결과 전체 변수에 대한 캐

쉬 실패율은 0.66587이며, 기존모델과 제안된 모델은 각각 0.66588, 0.66725이었다. 이는 제안된 모델은 주기가 다른 모든 변수에 대해 정확한 그룹 및 자기의존관계를 규명하며, 캐쉬실패의 개수를 구하는데 반영하기 때문에 더 정확한 캐쉬 실패의 개수를 구할 수 있는 것이다.

6. 결론 및 향후과제

본 논문은 직접 사상 캐쉬구조에서 완전 중첩 루프내의 각 변수의 간섭에 의한 캐쉬 실패의 개수를 구한 후, 프로그램의 수행시간을 정적으로 예측하는 방법을 제시하였다.

제시된 모델은 먼저, 루프내 각 변수의 재사용집합을 기반으로 교차 구간을 구한 후, 적절한 IS(v)와 GD(v)를 선택한다. 다음으로 선택된 변수의 UTHC을 형성하여 변수의 캐쉬 실패율을 구하고, 구해진 캐쉬 실패의 개수를 이용하여 프로그램의 총 수행시간을 예측한다. 이는 정적 성능예측의 장점인 빠르고 정확한 예측이 가능하다. 특히, 캐쉬 실패율을 예측하는데 걸리는 시간은 배열의 크기와는 무관하고, 사용된 변수의 개수에 비례하기 때문에 배열의 크기가 큰 프로그램의 성능예측에는 훨씬 더 성능향상을 가져올 수 있을 뿐만 아니라 다양한 유형의 배열 구조에도 쉽게 확장될 수 있다. 그리고 프로그램의 수행시간에 대한 성능 예측 및 성능예측기의 모델에 활용 가능하며, 데이타 지역성의 최적화, 캐쉬 구성(cache organization), 스케쥴링(scheduling)등과 같은 많은 분야에도 이용 가능하다. 그러나 배열의 인덱스 및 각 원소의 접근이 순차적이지 않는 경우에, 나누어진 구간은 배열의 크기에 비례하여 나타나기 때문에 예측시간도 그만큼 커지게 되므로 향후에는 예측시간을 단축할 수 있는 방법에 대해 연구를 진행할 예정이다.

참 고 문 헌

[1] A. Agarwal, M. Horowitz, J. Hennessy, "An alnalutical Cache Model," ACM Transaction Compuer System, Vol.7, No.2, pp.184-215, 1989.

[2] D.S. Han, "Performance Predictor for HPF Compilers," ICPADS'97, Korea, 1997.

[3] Gabriel Rivera, Chau-Wen Tseng, "Eliminating Conflict Misses for High Performance Architectures," ICS 98, July, 1998

[4] J. S. Harper, D. J. Kerbyson and G. R. Nudd, " Predicting the Cache Miss Ratio of Loop-Nested Array References" CS-RR-336, December 5, 1997.

[5] Kathryn S. McKinley. Automatic and Interactive Parallelization. PhD thesis, Rice University, Technical Report CRPC-TR92214, April 1992.

[6] K.S. Mckinley, S. Carr, C.W. Tseng, "Improving Data Locality with Loop Transformation," ACM Transaction Program Language System, Vol.18, No.4, pp424-453, 1996

[7] Michael Wolf and Monica Lam. "A Data Locality Optimizing Algorithm," In Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, Vol.26, No.6, pp.30-44, 1991.

[8] M. Parashar, S. Hariri, "Compile-Time Performance Prediction of HPF/Fortran 90D" IEEE Parallel Distributed Technology Spring 1996.

[9] Norman P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," 25 years of the international symposia on Computer architecture, pp71-73, 1998

[10] O. Temam, C. Fricker, and W. Jalby, "Cache Interference Phenomena," In Proceedings of ACM SIGMETRICS, pp.261-271, 1994

[11] S. Coleman, K.S Mckinley, "Tile Size Selection using Cache Organisation and Data Layout," In Proceedings of the SIGPLAN '95 Conference on Programming Language Design and Implementation, Vol.30, pp.279-289, 1995.

[12] 김 동승, "MPI 환경에서의 병렬 프로그램 수행시간 모델링 및 실험", 병렬처리시스템 학술회의 논문집, Vol.9, No.3, pp.121-128, 1998

[13] 이건영 외 2명, "성능예측기를 위한 캐쉬 간섭실패의 정적분석", 병렬처리시스템 학술발표회 논문집, Vol.10, No.3, pp3-12, 1999.

이 건 영

1997년 아주대학교 컴퓨터 공학과 공학사. 1999년 아주대학교 컴퓨터 공학과 공학석사. 1999년 ~ 현재 아주대학교 정보통신공학과 박사 과정. 관심분야는 병렬처리, 컴퓨터보안, 네트웍보안, 인공지능

정 유 석

1999년 아주대학교 컴퓨터 공학과 공학사. 1999년 ~ 현재 아주대학교 정보통신공학과 석사 과정. 관심분야는 병렬처리, 컴퓨터보안, 인공지능

홍 만 표

정보과학회논문지 : 시스템 및 이론
제 27 권 제 1 호 참조