임베디드 시스템 설계에서 효율적인 메모리 접근을 고려한 변수 저장 방법

(Storage Assignment for Variables Considering Efficient Memory Access in Embedded System Design)

최 윤 서 † 김 태 환 ††
(Yoonseo Choi) (Taewhan Kim)

요 약 DRAM에 의해 지원되는 페이지(page) 접근 모드나 버스트(burst) 접근 모드를 신중하게 이용하면 DRAM의 접근 시간(access latency) 및 접근 시에 소모되는 에너지를 줄일 수 있음이 많은 설계들에서 입증되었다. 최근에는 변수들을 메모리에 적절하게 배열함으로써 페이지 접근 횟수와 버스트 접근 회수를 각각 극대화시킬 수 있음이 밝혀졌다. 그러나 이러한 최적화문제는 쉽게 최적의 해를 구할 수 없다고 알려졌기 때문에, 주로 간단한 greedy 휴리스틱을 이용해서 풀려졌다. 본 논문은 기존의 방법보다 더 좋은 결과를 얻기 위해서 0-1 선형 프로그래밍(ILP)을 근간으로 한 기법을 제안한다. 벤치마크 프로그램들을 이용한 실험 결과를 보면, 제안된 알고리즘은 각각 OFU(order of first use) 방식과, [1,2]의 방식, [3]의 방식에 비해 평균적으로 각각 32.2%, 15.1%, 3.5%만큼 페이지 접근 회수를 증가시켰으며, 또한 각각84.4%, 113.5%, 10.1% 만큼의 버스트 접근 회수를 증가시켰다.
키워드 : 임베디드 시스템, DRAM, 코드 최적화

Abstract It has been reported and verified in many design experiences that a judicious utilization of the page and burst access modes supported by DRAMs contributes a great reduction in not only the DRAM access latency but also DRAM's energy consumption. Recently, researchers showed that a careful arrangement of data variables in memory directly leads to a maximum utilization of the page and burst access modes for the variable accesses, but unfortunately, found that the problems are not tractable, consequently, resorting to simple (e.g., greedy) heuristic solutions to the problems. In this paper, to improve the quality of existing solutions, we propose 0-1 ILP-based techniques which produce optimal or near-optimal solution depending on the formulation parameters. It is shown that the proposed techniques use on average 32.2%, 15.1% and 3.5% more page accesses, and 84.8%, 113.5% and 10.1% more burst accesses compared to OFU (the order of first use) and the technique in [1,2] and the technique in [3], respectively.
Key words : Embedded system, DRAM, Code optimization

1. 서 론

임베디드 시스템에서 메모리는 성능의 병목현상과 전력소모에 주요한 요인으로 작용한다[4]. 최근의 임베디드 시스템에서 쓰이는 DRAM들은 효율적인 접근 방식들을 제공한다. 특히 페이지(page) 접근 방식과 버스트(burst) 접근 방식은 광범위하게 제공되고 있다[5,6]. 일반적으로 랜덤 접근 방식의 대기시간(latency)은 페이지 접근 방식이나 버스트 접근 방식의 그것보다 길다. 또한, 페이지/버스트 접근 방식을 이용할 때 비트 당 접근에 소모되는 전력이 랜덤 접근 방식의 그것보다 훨씬 적다고 알려져 있다. 예를 들어, IBM의 CU-11 Embedded DRAM[4] 메모리의 경우 랜덤 접근의 방식은 10ns가 걸리고 페이지 접근 방식의 경우는 5ns가 걸린다. 또한 랜덤 접근 시에는 60mA/Mb의 전류가 흐르고, 페이지 접근 시에는 13mA/Mb의 전류가 흐른다. 결론적으로, 페이지나 버스트 접근 방식을 잘 이용하는 것은 메모리 액세스 지연 시간으로 인한 성능 저하 문제를

· 이 논문은 2003년도 한국학술진흥재단의 지원에 의해서 연구되었음 (KRF-2003-003-D00336)
† 비 회 원 : 한국과학기술원 전자전산학과
yschoi@arcs.kaist.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터 공학부 교수
tkim@ssl.snu.ac.kr
논문접수 : 2003년 12월 29일
심사완료 : 2004년 11월 3일

완화시키는데 매우 필요하다. Panda외 연구자들은 [1] DRAM에서 페이지 접근 횟수를 모델링하였고 비배열(scalar) 변수들과 배열(array) 변수들을 메모리에 배열하는 알고리즘을 제안하였다. 이들이 이용한 모델링 방식은 [2]에 나온 비배열 변수들을 위한 메모리 정렬 방식과 동일하다. Khare외 연구자들은 [7], [1]의 연구를 확장시켜서 이중 메모리 아키텍처에서 버스트 접근 방식을 지원하도록 했다. 이들의 방법은 주로 두 메모리로의 접근을 번갈아하는데(interleaving) 초점을 두었다. Grun외 연구자들은[8] 메모리 접근시의 자세한 타이밍 정보를 이용하는 컴파일 기술을 제안하였다. 그들은 [9]에서 가장 많이 쓰이는 메모리 접근 패턴을 분석하고 추출하여 모으는 방식을 제안하였다. Ayukawa외 연구자들은[10] 하드웨어를 이용하여 동적으로 변수가 접근되는 순서를 조절하여 페이지 miss 부담을 줄이는 방식을 제안하였다. Hettiaratchi외 연구자들은[11] 페이지 접근 회수를 최대화되도록 배열 변수들의 주소를 정해줌으로써 전력소모를 줄이는 방법을 제안하였다. 그들은 이 문제를 다중 그래프 분할 문제로 공식화하고, Kernighan-Lin의 방식에 기초한 분할 알고리즘을 이용하여 풀었다. 마지막으로 [3]의 저자들은 페이지 또는 버스트 접근 방식의 접근 횟수를 최대화하는 문제는 NP-hard함을 밝히고, 각각을 위한 휴리스틱 알고리즘을 제안하였다. 본 논문에서는, 위에 언급된 연구들을 보충하고자, 페이지 접근 회수와 버스트 접근 회수를 최대화하는 문제에 대해 최적에 가까운 결과를 얻는 방법을 제안한다. 이를 위해, ILP에 기초한 zone_alignment 라 불리는 알고리즘을 제안한다.

2. 연구 동기

$a_1, a_2, ..., a_{i-1}, a_i, ...$ 를 프로그램 코드 상에서의 변수 접근 순서라고 하자. 변수들이 메모리 상에 정렬되어 있을 때 $v(a_i)$ 를 접근 a_i 에 의해서 실제 접근되는 변수라고 하자. $f_{page}(v(a_i))$ 과 $f_{addr}(v(a_i))$ 는 각각 변수 $v(a_i)$ 가 속해있는 메모리 상에서의 페이지와 그 페이지 내에서의 상대적인 위치라고 하자. 이 때, 우리는 페이지 접근과 버스트 접근을 다음과 같이 정의할 수 있다.

(1) 정상적인(normal) 접근 방식과 페이지(page) 접근 방식을 이용할 수 있을 때, a_i 가 페이지 접근이 된다는 의미는 $f_{page}(v(a_i)) = f_{page}(v(a_{i-1}))$ 인 조건이 성립됨을 뜻한다.

(2) 정상적인 접근 방식과 버스트 접근 방식을 이용할 수 있을 때, a_i가 버스트 접근이 된다는 의미는 $f_{page}(v(a_i)) = f_{page}(v(a_{i-1}))$이고 $f_{addr}(v(a_i)) = f_{addr}(v(a_{i-1}))+1$ 인 조건인 조건이 성립됨을 뜻한다.

페이지 접근 방식을 이용해서 접근시간과 소모전력을 감소시키는 가장 효율적인 방법 중의 하나는 페이지 접근 횟수가 최대가 되도록 메모리 상에서 변수들의 상대적인 위치를 정하는 것이다. 그림 1은 메모리 상의 변수 정렬이 메모리 접근 시간과 소모전력에 미치는 영향을 잘 보여주고 있다. 그림 1(a)에서 변수 *a, b, c*가 첫 번째 페이지에 변수 *d, e, f*가 두 번째 페이지에 있다. 이러한 변수 정렬이 주어진 상태에서, 어떻게 정상적인(normal) 접근과 페이지 접근이 일어나는 지는 그림 1의 아래 부분에 잘 요약되어 있다. 이는 6번의 페이지 접근과 (즉, 6번의 열 디코딩) 6번의 정상적인 접근(즉, 6번의 행 디코딩+열 디코딩+프리차지)로 이루어진다. 반면에, 그림 1(b)는, 또 다른 메모리 상의 변수 정렬을 보여준다. 비교 요약하면, 그림 1(b)의 메모리 상의 변수 정렬은 16.7%의 메모리 접근 시간 단축과 29.3%의 전력소모의 감소를 보여준다. 비슷한 방법으로 버스트 접근을 극대화하기 위해서 그림 1(a)와는 다른 메모리

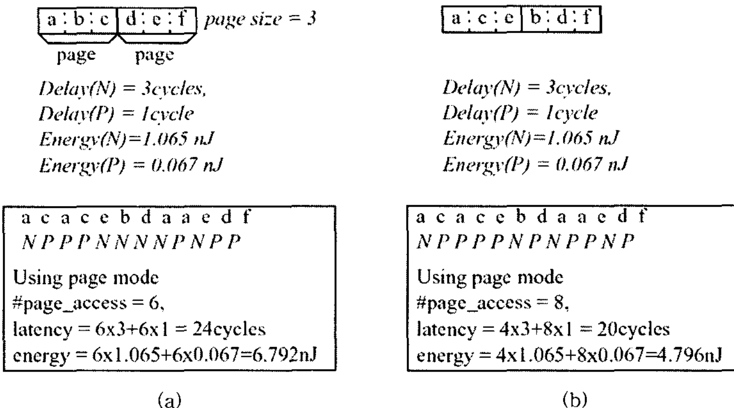


그림 1 메모리 상의 변수 정렬이 페이지 접근 횟수에 미치는 영향을 보여주는 예

상의 변수 정렬을 찾을 수 있을 것이다.

3. 페이지 접근을 최대화하기 위한 메모리 상의 변수 정렬 방식

3.1 문제 정의

우리가 풀고자하는 최적화문제는 변수 접근 순서가 주어지면, 페이지접근 회수를 최대화하는 메모리 상의 변수 정렬을 찾는 것이다. 우리는 이 문제를 *MLP 문제(Memory Layout with Page Mode)*라고 부르기로 하자. *MLP*문제의 한 객체는 (S, V, m)으로 규정된다. 여기서 S는 변수 접근 순서를, V는 접근되는 변수 집합을, m은 메모리의 페이지 크기를 나타낸다. 이때 V에 속한 변수들을 서로 겹침이 없는 그룹으로 분할해서 각각의 그룹이 m개 이하의 변수를 가지도록 한다.

정의 3.1: S의 접근 그래프는 멀티 그래프 $G(V, E)$로서, V는 S에 속한 변수들의 집합이다. 이때 V에 속한 두 노드 v_i와 v_j에 대해서 그들 사이에 n개의 에지가 존재함은 변수 접근 순서 S안에서 변수 v_i와 v_j의 접근이 정확히 n번 이웃해서 일어남을 나타낸 것이다.

그림 2(a)는 정의 3.1의 접근 그래프를 보여준다. 예를 들면, 노드 a와 c사이에는 3개의 에지가 존재하는데 이는 일련의 변수 접근 순서에서 a와 c가 정확히 3번 이웃하여 일어나기 때문이다.

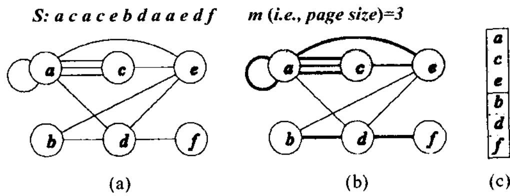


그림 2 변수 접근 순서와 (a) 접근 그래프, (b) 최적 그래프 분할, (c) (b)로부터 유도된 메모리 상의 변수 정렬

우리는 그래프 분할문제를 다음과 같이 정의한다. 멀티그래프 $G(V,E)$의 분할 $\Pi^m_{G(V,E)}$는 집합 V를 t개의 서로 겹치지 않는 부분집합들 $V_1, V_2, \ldots, V_t$로 나누어, 각각이 최대 m개의 노드들을 가지고 다음 수식의 값을 최대화하도록 한다.

$$g(\Pi^m_{G(V,E)}) = \sum_{(v_i,v_j)\in E,\, v_i,v_j \in V_l,\, l=1,\cdots,t} w(v_i, v_j), \qquad (1)$$

이때 $w(v_i,v_j)$는 v_i와 v_j 간의 에지의 개수를 나타낸다.

정리 3.1: [3] (S,V,m)의 *MLP* 문제는 접근 그래프 $G(V,E)$의 분할문제와 동일하다.

정리 3.1에 따르면 접근 그래프의 최적 분할로부터 유도된 메모리 상의 변수 정렬은 역시 최적이다. 예를 들면, $m = 3$ 일 때, 그림 2(b)는 이득이 8인(즉, 굵게 칠해진 에지의 개수) 그래프 분할을 보여준다. 그리고, 그림 2(c)는 이로부터 유도된 메모리 상의 변수 정렬을 보여주는데, 이 메모리 상의 변수 정렬을 이용하면 역시 8번의 페이지 접근이 발생한다. 따라서 우리는 해당 접근 그래프를 만들고, 그 접근 그래프의 최적의 분할을 찾는 방식으로 *MLP* 문제를 풀고자 한다.

3.2 최적 공식화

문제의 크기가 작을 때는, 그래프 분할의 정확한 0-1 ILP 공식을[12] 이용해서 문제의 최적 결과를 구한다. 주어진 변수 접근 순서 S 에 대해서 해당 접근 그래프 $G(V,E)$를 구했을 때, 우리가 풀고자 하는 문제는 수식 (1)의 값을 최대화하는 분할 $\Pi^m_{G(V,E)}$를 찾는 것이다. 0-1 ILP 변수들을 다음과 같이 정의한다.

$$x^k_{i,j} = \begin{cases} 1, & V\text{의 원소 } v_i \text{와 } v_j \text{가 같은 분할 } V_k\text{에 속하는 경우;} \\ 0, & \text{그렇지 않은 경우.} \end{cases}$$

$$y^k_i = \begin{cases} 1, & V\text{의 원소 } v_i\text{가 분할 } V_k\text{에 속하는 경우;} \\ 0, & \text{그렇지 않은 경우.} \end{cases}$$

그러면 0-1 ILP 공식은 다음과 같다.

$$\text{Maximize} \quad \sum_{k=1}^{\lceil \frac{|V|}{m} \rceil} \sum_{i=1}^{|V|-1} \sum_{j\geq i+1}^{|V|} w_{ij} \cdot x^k_{i,j} \qquad (2)$$

subject to,

$$x^k_{i,j} - x^k_{j,i} = 0, \quad i,j = 1,\cdots,|V|,\ k = 1,\cdots,\left\lceil \frac{|V|}{m} \right\rceil \qquad (3)$$

$$\sum_{k=1}^{\lceil \frac{|V|}{m} \rceil} y^k_i = 1, \qquad i = 1,\cdots,|V| \qquad (4)$$

$$\sum_{i=1}^{|V|} y^k_i \leq m, \qquad k = 1,\cdots,\left\lceil \frac{|V|}{m} \right\rceil \qquad (5)$$

$$\sum_{j=1}^{|V|} x^k_{i,j} - m \cdot y^k_i \leq 0, \quad i = 1,\cdots,|V|,\ k = 1,\cdots,\left\lceil \frac{|V|}{m} \right\rceil \qquad (6)$$

수식 (2)의 w_{ij}는 접근 그래프에서 노드 v_i와 v_j 간의 에지의 개수를 나타낸다[1]. 제한식 (3)은 $x^k_{i,j} = x^k_{j,i}$를 보장하며, 제한식 (4)는 각각의 노드가 정확히 하나의 그룹에만 속해야 함을 나타낸다. 제한식 (5)는 페이지의 크기에 대한 제한이다. 마지막으로, 제한식 (6)은 변수 v_i에 대하여 $x^k_{i,j} = 1$이면 $y^k_i = 1$임을 나타낸다.

3.3 휴리스틱 방법

크기가 큰 *MLP* 문제를 풀기 위해서 zone_alignment라는 알고리즘을 제안하며, 이는 반복적인(iterative) 방식이다. 각 반복과정에서 입력으로 주어진 접근 순서 중의 일부의 연결된 접근 순서(*access_zone*)를 추출해 내서 이 *access_zone* 에 해당하는 ILP 공식을 적용한다. *access_zone*을 추출해 내기 위해서 주어진 프로그램 상에서 변수들의 생성에서 소멸까지의 시간,

1) i=j이면 $w_{ij} = 0$으로 한다. 왜냐하면 식 (1)에 의해 자체-룹(self-loop)의 가중치는 분할 결과와 상관없이 항상 더해지기 때문이다.

즉, 라이프타임(lifetime)을 이용한다. 이러한 과정을 변수 접근 순서에 있는 모든 변수들이 메모리의 페이지에 배정될 때까지 반복한다. 이러한 zone_alignment 방식은 다음과 같은 근거들로 인해서 효과적인 결과를 보장할 수 있다: (1) 일반적으로 변수의 라이프타임은 코드 전체에 걸쳐있기 보다는 국부적인 경우가 많다. zone_alignment 방식은 변수 접근의 시간적 국부성(temporal locality)을 이용한다.; (2) zone_alignment는 각각의 *access_zone*에 대해서는 최적의 결과를 낸다. 여러 개의 *access_zone*으로 나누어졌기 때문에 일어날 수 있는 결과의 질의 저하를 최소화하기위해서 3.2절의 ILP 공식을 수정한 방식을 이용한다.

주어진 변수 접근 순서를 적당한 크기의 몇 개의 *access_zone*으로 나누었다고 하자. 적당한 크기로 나누는 문제에 대해서는 나중에 언급한다. Z를 클락 스텝 C_{start}에서부터 시작하여 클락 스텝 C_{end}에서 끝나는 하나의 *access_zone*이라고 하자. Z 와 관련된 몇 개의 용어에 대한 정의를 먼저 내린다.

- V_{in} : 라이프타임이 C_{start}와 C_{end} 사이에서, 즉, $[C_{start}, C_{end}]$ 범위 안에서 끝나고 아직 메모리의 어떤 페이지에 저장될지 정해지지 않은 변수들의 집합.
- V_{cross} : 라이프타임이 C_{end} 전에 시작되고 C_{end} 전에 끝나지만 아직 메모리의 어떤 페이지에 저장될지 정해지지 않은 변수들의 집합.
- V_{zone} : $V_{in} \cup V_{cross}$, 즉, 현 *access_zone*인 Z에서 메모리에 할당하기 위해서 고려하는 변수들의 집합.

zone_alignment에서 V_{in}에 속한 변수들은 모두 현재 *access_zone*인 Z에서 메모리 안에서 의 위치가 정해져야 되지만, V_{cross}에 속한 변수들은 선택적으로 메모리 안에서의 위치가 정해지면 된다. 즉 이번 *access_zone*에서 꼭 메모리 상에서의 위치가 정해지지 않아도 된다. 이는 V_{cross}에 속한 변수들에 대해서는 Z안에서 다른 변수들과 이웃한 접근에 대한 부분적인 정보만이 반영되어 있기 때문이다. 현재 *access_zone*인 Z에서 메모리 상의 위치가 정해지지 않은 V_{cross}에 속한 변수들은 다음 번 *access_zone* 들 중 하나에서 메모리 상의 위치를 배정받게 된다. *access_zone* Z를 위한 ILP 공식은 다음과 같다.

$$\text{Maximize} \quad \sum_{k=1}^{\lceil |V_{zone}|/m \rceil} \sum_{v_i \in V_{zone}} \sum_{v_j \in V_{zone}, j \geq i+1} w_{ij} \cdot x_{i,j}^{k}$$

subject to,

$$x_{i,j}^{k} - x_{j,i}^{k} = 0, \quad i, v_j \in V_{zone}, k = 1, \cdots, \lceil |V_{zone}|/m \rceil \quad (7)$$

$$\sum_{k=1}^{\lceil |V_{zone}|/m \rceil} y_i^k = 1, \qquad v_i \in V_{\in} \quad (8)$$

$$\sum_{k=1}^{\lceil |V_{zone}|/m \rceil} y_i^k \leq 1, \qquad v_i \in V_{cross} \quad (9)$$

$$\sum_{v_i \in V_{cross}} \sum_{k=1}^{\lceil |V_{zone}|/m \rceil} y_i^k \leq cross_bound \quad (10)$$

$$\sum_{v_i \in V_{zone}} y_i^k \leq m, \quad k = 1, \cdots, \lceil |V_{zone}|/m \rceil \quad (11)$$

$$\sum_{v_j \in V_{zone}} x_{i,j}^{k} - m \cdot y_i^k \leq 0, \; v_i \in V_{zone}, k = 1, \cdots, \lceil |V_{zone}|/m \rceil \quad (12)$$

3.2절의 제한식 (4)는 제한식 (8)과 (9)의 두개로 나뉘었다. 제한식 (8)은 V_{in}에 속한 모든 변수들은 메모리 상의 어떤 페이지에 배정되어야 함을 나타낸다. 제한식 (9)는 V_{cross}에 속한 변수들은 선택적으로 메모리 상의 어떤 페이지에 배정됨을 나타낸다. 제한식 (10)은 V_{cross}에 속한 변수들 중에서 몇 개의 변수가 Z에서 메모리 안의 위치가 결정이 될 지를 제한한다. (10)에서 *cross_bound*는 다음과 같이 정의된다.

$$cross_bound = \left| \sum_{v_i \in V_{cross}} \frac{f_z(v_i)}{f(v_i)} \right|$$

이때, $f_z(v_i)$는 *access_zone* Z에서 변수 v_i가 접근되는 횟수를 나타내고, $f(v_i)$는 전체 접근 순서에서 v_i가 접근되는 횟수를 나타낸다. $f_z(v_i)/f(v_i)$를 변수 v_i를 현재 *access_zone* Z에서 메모리에서의 위치를 정할 가능성의 척도로 쓴다. 마지막으로, (13)과 (14)는 3.2절의 (5)과 (6)에 해당한다.

현재 *access_zone* Z의 zone_alignment ILP를 풀고 난 다음에는 다음 *access_zone*을 위하여 V_{in}와 V_{cross}를 갱신한다. Z에서의 이익 값(즉, 페이지 접근 회수)은 ILP의 목적식으로부터 구할 수 있다. Z에서의 이익을 $gain(Z)$로 표현하면, 전체 접근 순서로부터 나오는 페이지 접근 회수의 총수는 다음의 식으로 계산된다.

$$\sum_{i=1}^{T} gain(Z_i) + \sum_{i=1}^{|V|} w_i^{self}$$

이때, 주어진 전체 접근 순서는 $Z_1, Z_2, \ldots, Z_T$의 T개의 *access_zone*으로 나뉘었고, w_i^{self}는 전체 접근 순서에서의 동일한 변수 v_i가 이웃해서 일어나는 회수 즉, (v_i, v_i)의 회수를 나타낸다. w_{ij} 는 $i=j$ 일 때는 0라는 것을 제외하면 접근 그래프에서 에지(v_i, v_j)의 가중치를 나타낸다.

예를 들어, 다음과 같이 *access_zone* Z_1과 Z_2로 나뉘어진 변수 접근 순서 S를 보자.

Z_1: *p1, p2, j, j, p2, p3, p1, p2, j, z, p2, j, p3, j, p1, j, j, x, p1, p2, n, z, z.*

Z_2: *pp, z, z1, p1, pp, z1, z, z, z2, i, i.*

Z_1의 ILP를 고려하면, C_{start} 는 Z_1의 첫 번째 변수가 접근되는 시점이고(즉, *p1*이 접근되는) C_{end}는 마지막으로 *z*가 접근되는 시점이다. 그러면 V_{in} = {*p2, j, p3, x, n*}이 되는데 왜냐하면, 이들은 Z_1에서만 접근되기 때문이다. 즉, 이 변수들의 라이프타임은 C_{start}와 C_{end} 사이

에 끝난다. *p1*과 *z*는 Z_1에서 처음으로 접근되고 Z_2에서도 접근되기 때문에 V_{cross} = {*p1, z*}이다. 즉, 이 변수들의 라이프타임은 C_{end} 이전에 시작되어 C_{end} 이후에 끝난다. *p1*은 전체 5번의 접근 중에 4번, *z*는 전체 6번의 접근 중에 세 번 Z_1에서 접근되기 때문에 *cross_bound* = $|4/5+3/6|$ = 2 이다. 따라서 V_{zone} = {*p1, p2, j, p3, z, x, n*} 이다.

다음으로 Z_1의 ILP 공식을 살펴보자. 쉽게 쓰기 위해서 *p1, p2, j, p3, z, x, n* 대신에 v_0부터 v_6로 대신 쓴다. 그러면 V_{in} = {v_1, v_2, v_3, v_5, v_6}, V_{cross} = {v_0, v_4}가 된다. m = 4 일때, $\left\lceil \frac{|V_{zone}|}{m} \right\rceil = \left\lceil \frac{7}{4} \right\rceil$ = 2이다. 이때 Z_1 의 ILP 공식은

$$\text{Maximize} \sum_{k=1}^{2}\sum_{i=0}^{6}\sum_{j>i+1}^{6} w_{ij} \cdot x_{i,j}^{k}$$

subject to,

$$x_{0,1}^{1}-x_{1,0}^{1}=0,\ x_{0,1}^{2}-x_{1,0}^{2}=0,$$
$$\cdots,\ x_{5,6}^{1}-x_{6,5}^{1}=0,\ x_{5,6}^{2}-x_{6,5}^{2}=0,$$
$$y_1^1+y_1^2=1,\ y_2^1+y_2^2=1,\ y_3^1+y_3^2=1,$$
$$y_5^1+y_5^2=1,\ y_6^1+y_6^2=1,$$
$$y_0^1+y_0^2\le 1,\ y_4^1+y_4^2\le 1,\ y_0^1+y_0^2+y_4^1+y_4^2\le 2,$$
$$y_0^1+\cdots+y_6^1\le 4,\ y_0^2+\cdots+y_6^2\le 4,$$
$$x_{0,1}^{1}+x_{0,2}^{1}+\cdots+x_{0,6}^{1}-4$$
$$\cdot\, y_0^1\le 0,\ x_{0,1}^{2}+x_{0,2}^{2}+\cdots+x_{0,6}^{2}-4\cdot y_0^2\le 0,$$
$$x_{1,0}^{1}+x_{1,2}^{1}+\cdots+x_{1,6}^{1}-4$$
$$\cdot\, y_1^1\le 0,\ x_{1,0}^{2}+x_{1,2}^{2}+\cdots+x_{1,6}^{2}-4\cdot y_1^2\le 0,$$
$$\cdots$$
$$x_{5,0}^{1}+\cdots+x_{5,6}^{1}-4\cdot y_5^1\le 0,\ x_{5,0}^{2}+\cdots+x_{5,6}^{2}-4\cdot y_5^2\le 0.$$

구해진 결과는 $y_0^1 = y_1^1 = y_2^1 = y_3^1 = y_4^2 = y_5^2 = y_6^2 = 1$이고 *gain*($Z_1$) = 14이다. 이는 변수 *p1, p2, j, p3* 가 하나의 페이지에 할당되고, *z, x, n*이 또 하나의 다른 페이지에 할당되는 것을 의미한다. 이 예에서 Z_1의 V_{cross}에 속한 모든 변수들이 메모리에 할당되었기 때문에 다음 *access_zone* 인 Z_2에 zone_alignment를 적용할 때는 V_{in} = {*pp, z1, z2, i*}, V_{cross} = Φ가 된다. 결과적으로 *gain*(Z_2) = 2 가 되고 모든 변수들은 하나의 페이지에 배정된다.

따라서 종합하면, 변수들은 {{*p1, p2, j, p3*}, {*z, x, n*} {*pp, z1, z2, i*}}로 분할되고 식 (15)에 의한 전체 이익은 14 + 2 + 5 (= *gain*(Z_1) + *gain*(Z_2) + $\sum w_i^{self}$) = 21이다.

access_zone [C_{start}, C_{end}] 에 zone_alignment를 적용하여 결과를 구하고 나면 다음 번 *access_zone*인 [C'_{start}, C'_{end}]를 구해야 한다. 이때 C'_{start}는 C_{end}의 바로 다음 접근 시점이다(즉, C'_{start} = C_{end} + 1). 그리고 나서 이 새로운 *access_zone*에 zone_alignment 방식을 적용한다. 모든 변수들이 모두 메모리의 페이지에 배정될 때까지 이 과정을 반복한다. 우리가 제안한 iterative 한 알고리즘의 결과의 질을 보장하기 위해서는 주어진 변수 접근 순서를 적당한 크기의 즉, ILP로 푸는데 적당한 시간이 걸리는 만큼의 여러 개의 *access_zone*으로 나누는 방식이 중요하다. 사용자로부터 주어진 *L*값을 하나의 *access_zone*안에 속할 수 있는 변수의 개수의 상한선으로 둠으로써 *access_zone* 크기를 제한한다. 특히 $L-\alpha \le |V_c| + |V_{cross}| \le L$를 만족하는 접근 시점 중에서 $|V_{cross}|$의 값이 최소인 시점을 C'_{end} 로 정한다(단, α는 작은 정수값이다). $|V_{cross}|$의 값이 최소인 시점을 C'_{end} 로 정하는 이유는 이렇게 함으로써 *access_zone*으로 나누기 때문에 발생하는 질의 저하를 줄일 수 있기 때문이다. 이러한 C'_{end}를 구하기 위해서 함수 *Get_Next_End(S)*를 이용한다. 마지막으로 그림 3은 zone_alignment 방식에 대한 요약된 흐름을 보여준다. *L*값을 조절함으로써 수행시간과 결과의 질에 대한 상충을 활용하여 원하는 결과를 얻을 수 있다.

```
zone_alignment: /* ILP based page assignment */
Inputs: Access graph G(V,E), page size m,
        access sequence S, user-defined values L and α.
Output: Page assignment for V.
• Z_rem ← S; /* Z_rem: remaining access sequence */
• c'_end ← −1;/* the access-time before the start of S */
repeat {
    • c'_start = c'_end + 1;
    • c'_end ← Get_Next_End(Z_rem);
    • Apply the ILP formulation to [c'_start, c'_end];
    • Z_rem = [c'_end + 1, end of S]; /* update Z_rem */
} until (Z_rem is empty)
```

그림 3 zone_alignment 방식의 요약

4. 버스트 접근을 최대화하기 위한 메모리 상의 변수 정렬 방식

4.1 문제 정의

이 절에서는 DRAM이 [6]의 예에서와 같이 정상적인 접근 방식과 버스트 접근 방식을 제공한다고 가정한다. 이때 풀고자하는 최적화문제는 변수 접근 순서가 주어지면, 버스트 접근 회수를 최대화하는 메모리 상의 변수 정렬을 찾는 것이다. 우리는 이 문제를 (*S, V, m*)의 *MLB 문제*(*Memory Layout with Burst Mode*)라고 부르기로 한다.

정의 4.1: *S*의 방향성을 지닌 접근 그래프는 방향성을 지닌 그래프 *G*(*V, A*)로서 *V*는 *S*에 속한 변수들의 집합이다. 이때, *V*에 속한 두 노드 v_i와 v_j에 대해서, v_i

로부터 v_j로 n개의 아크가 존재한다는 것은 변수 접근 순서 S안에서 변수 v_i와 v_j 의 연속적인 접근이 이 순서로 연속해서 정확히 n번 일어날 조건과 일치한다. 이때, $w\langle v_i, v_j\rangle = n$이다.

$S = a_1, a_2, ..., a_{i-1}, a_i, ... a_N$이라 하자. 앞에서 언급되었듯이, 접근 a_i가 버스트 접근일 조건과 $f_{page}(v(a_i)) = f_{page}(v(a_{i-1}))$이고 $f_{addr}(v(a_i)) = f_{addr}(v(a_{i-1}))+1$일 조건은 서로 필요충분의 관계에 있다. 따라서, 변수 $v(a_{i-1})$와 변수 $v(a_i)$가 이 순서로 연속된 접근이 n번이 있고 $v(a_{i-1})$와 $v(a_i)$가 같은 페이지 안에서 이 순서로 연속된 위치에 저장되어 있다면 a_i 는 버스트 접근이 된다(변수 접근 순서와 그에 따른 방향성을 지닌 접근 그래프의 예는 그림 4(a)에 있다.).

정의 4.2: $G(V,A)$의 경로 커버(path cover)는 V에 속한 어떤 노드도 서로 공유하지 않는 방향성을 지닌 경로들의 집합이다. 또한 이 집합에 속한 경로들은 집합적으로 V에 속한 모든 노드들을 포함한다. 크기가 m인 경로 커버 $C^m_{G(V,A)}$는 $C^m_{G(V,A)}$에 속한 모든 경로들의 크기가 m을 넘지 않는다. (경로의 크기란 경로가 연결하는 노드의 개수를 말한다.)

경로 커버 $C^m_{G(V,A)}$의 이익은 다음과 같이 정의된다.

$$g(C^m_{G(V,A)}) = \sum_{\langle v_i, v_j\rangle \in A,\, v_i \in p,\, v_j \in p,\, p \in C^m_{G(V,A)}} w\langle v_i, v_j\rangle \quad (13)$$

정의 4.3: 크기가 m인 그래프 $G(V,A)$의 최대가중치 경로 커버(maximum weighted path cover: MWPC)는 식 (13)의 값을 최대화시키는 경로 커버 $C^m_{G(V,A)}$이다.

경로 커버에서 유도된 메모리 상의 변수 정렬은 다음과 같다. 즉, $C^m_{G(V,A)}$의 각각의 경로들은 메모리의 서로 다른 페이지로서, 각 경로에 노드가 나열된 순서대로 해당 변수가 페이지에 차례로 저장된다. 예를 들어, 그림 4(c)는 m = 3일 때 그림 4(b)의 경로 커버로부터 유도된 메모리 상의 변수 정렬이다.

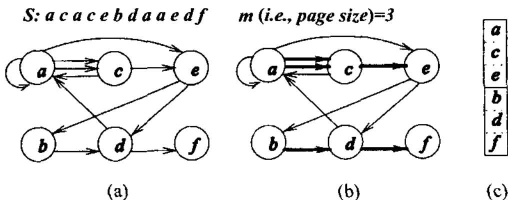


그림 4 (a) 변수 접근 순서와 방향성을 지닌 접근 그래프, (b) 경로 커버, (c) (b)로부터 유도된 메모리 상의 변수 정렬

우리의 ILP 공식은 [3]의 두 개의 정리로부터 기인한다.

정리 4.1: 크기가 m인 최적의 MWPC로부터 유도된 메모리 상의 변수 정렬은 버스트 접근 회수를 최대화한다.

정리 4.2: MLB문제는 NP-hard 하다.

4.2 최적해를 구하기 위한 공식화

이 절에서는 (S, V, m)이 주어졌을 때 방향성을 지닌 그래프의 최적해(경로 커버) $C^m_{G(V,A)}$를 찾아내는 ILP 공식을 이용해서 MLB문제의 최적해를 구한다. 최적 경로 커버를 구하는 것은 3절의 최적 분할을 구하는 것과는 매우 다르다. 따라서, 3절의 공식에 제한식을 더해서 문제를 풀기보다는 경로 커버를 구하는 ILP를 직접 공식화하기로 한다.

(S, V, m)이 주어졌을 때 각각의 열이 V에 속한 $n(=|V|)$개의 변수들인 $v_1, ..., v_n$로 이루어진 m개의 열로 된 그림 5와 같은 그래프를 보자. 방향성을 지닌 접근 그래프의 경로 커버의 각각의 경로는 m열로 된 해당 그래프에서도 서로 노드를 공유하지 않는 경로들로 나타낼 수 있음을 알 수 있다. 이때 풀고자하는 문제는 특정 값이(이 특정 값은 뒤에 정의된다) 최대가 되도록 이 m개의 열로 된 그래프에서 각각의 길이가 m을 넘지 않는 경로들로 이루어진 경로 커버를 찾는 것이다. 그림 5에서 알 수 있듯이 이 그래프에서는 서로 이웃한 열에 속한 변수들 사이에만 아크가 형성되는 것이 허용되므로 가장 긴 경로의 길이도 m을 넘을 수가 없다.[2] 이러한 경로들을 설명하기 위해서 k번째 열에 있는 노드 v_i와 $(k+1)$번째 열에 있는 노드 v_j 간의 관계를 나타내기 위한 ILP 변수 $x_{i_kj_{k+1}}$을 다음과 같이 정의한다.

$$x_{i_kj_{k+1}} = \begin{cases} 1, & \text{아크 } \langle v_i, v_j\rangle \text{ 가 경로 상에 존재하는 경우;} \\ 0, & \text{아크 } \langle v_i, v_j\rangle \text{ 가 경로 상에 존재하지 않는 경우.} \end{cases} \quad (14)$$

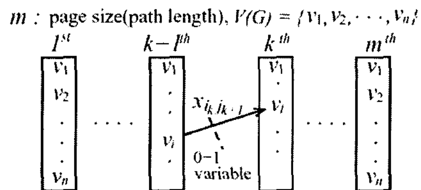


그림 5 노드(즉, 변수)의 개수가 n개일 때, ILP 공식을 위한 m열로 된 그래프 생성

이때, $x_{i_kj_{k+1}}$을 사용한 ILP 공식은 다음과 같다.

$$\text{maximize} \quad \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \sum_{k=1}^{m-1} w_{ij} \cdot x_{i_kj_{k+1}} \quad (15)$$

subject to,

2) 경로의 길이는 경로에 속한 노드들의 개수이다.

$$\sum_{i=1}^{|V|} x_{i_p j_p} + \sum_{i=1}^{|V|} \sum_{k=1, k \neq p}^{m-1} x_{j_k i_{k+1}} \leq 1,$$

$$p=2,\cdots,m, \quad j=1,\cdots,|V| \tag{16}$$

$$\sum_{i=1}^{|V|} x_{j_p i_{p+1}} + \sum_{i=1}^{|V|} \sum_{k=2, k \neq p}^{m} x_{i_k j_k} \leq 1,$$

$$p=1,\cdots,m-1, \quad \forall j=1,\cdots,|V| \tag{17}$$

(16)과 (17)은 어떤 경로가 p번째 열의 노드 v_j를 지난다면 이 경로는 다른 열에 속해있는 어떤 노드 v_j도 지날 수 없음을 의미한다.

만약 문제의 크기가 하나의 ILP 공식을 적용하기에는 너무 크다면, 위의 공식에 제한식을 더함으로써 zone_alignment 방식을 이용한다. 이를 위해서 새로운 0-1 ILP 변수인 y_i를 정의하여 사용한다.

$$y_i = \begin{cases} 1, & \text{어떤 열에 속한 } v_i \text{ 가 어떤 경로에 속하는 경우;} \\ 0, & \text{그렇지 않은 경우.} \end{cases}$$

이때, zone_alignment를 위한 새로운 ILP 공식은 다음과 같다.

$$\text{maximize} \quad \sum_{v_i \in V_{zone}} \sum_{v_j \in V_{zone}} \sum_{k=1}^{m-1} w_{ij} \cdot x_{i_k j_{k+1}} \tag{18}$$

subject to,

$$\sum_{v_i \in V_{zone}} x_{i_p j_p} + \sum_{v_i \in V_{zone}} \sum_{k=1, k \neq p}^{m-1} x_{j_k i_{k+1}} \leq 1,$$

$$p=2,\cdots,m, \; j \in V_{zone} \tag{19}$$

$$\sum_{v_i \in V_{zone}} x_{j_p i_{p+1}} + \sum_{v_i \in V_{zone}} \sum_{k=2, k \neq p}^{m} x_{i_k j_k} \leq 1,$$

$$p=1,\cdots,m-1, \; j \in V_{zone} \tag{20}$$

$$\sum_{k=1}^{m-1} \sum_{v_i \in V_{zone}} x_{j_k i_{k+1}} + \sum_{k=2}^{m} \sum_{v_i \in V_{zone}} x_{i_k j_{k-1}} - 2 \cdot y_j \leq 0,$$

$$j \in V_{zone} \tag{21}$$

$$\sum_{v_i \in V_{zone}} y_i \leq |V_{in}| + cross_bound \tag{22}$$

(19)와 (20)은 (16)과 (17)에 해당한다. (21)은 노드 v_j가 어떤 열에서 어떤 경로에 포함되면 y_i가 1이 됨을 나타낸다. (22)는 페이지에 할당된변수의 개수는 $|V_{in}|$ + *cross_bound*를 넘으면 안 됨을 나타낸다.

5. 실험 결과

제안한 알고리즘의 효과를 확인하기 위해서 몇몇의 실험들을 수행했다. Intel Xeon3 500 X4의 linux 머신에서 *cplex* ILP 솔버를 이용하였으며, 표 1에 나온 벤치마크 프로그램들을([13, 14, 15]) 실험에 사용하였다. 제안된 알고리즘의 결과를 OFU(the order of first use, 프로그램 상에서 변수가 나타나는 순서대로 메모리 상에 변수의 상대적인 위치를 정해주는 방식) 방식의 결과와, [1,2]에 나온 CGB(the closeness-graph-based) 방식의 결과, 또한 Greedy 방식[3]과 비교하였다.

페이지 접근 회수를 최대화하기 위한 메모리 상의 변수 정렬 최적화: 표 2는 OFU, CGB, Greedy에 의한 페이지 접근 회수와 본 논문에서 제안한 zone_alignment 알고리즘에 의한 페이지 접근 회수를 비교하여 보여준다. ELLIP 의 경우, m이 3, 4, 6, 8, 10이었을 때는 변수 접근 순서를 3개의 *access_zone*으로 나누어 zone_alignment를 적용했다. GAULEG의 경우에는, m이 3, 4 또는 6 값을 가질 때, GAUJAC, DBRENT의 경우에는 m이 3, 4, 6, 또는 8일 때 각각의 변수 접근 순서를 두 개의 *access_zone*으로 나누어 zone_alignment를 적용했다. 그 외의 벤치마크 프로그램에 대해서는 변수 접근 순서를 나누지 않고 그대로 이용했다. 요약하면, zone_alignment를 이용했을 때, OFU나 CGB, Greedy 알고리즘을 이용했을 때 비해서 각각 32.2%, 15.1%, 3.5% 씩 페이지 접근 회수가 증가했다. Greedy 알고리즘을 이용하면 모든 벤치마크 프로그램을 1초 이내에 결과를 낼 수 있었다. 3.2 절에 나온 최적해를 구하는 ILP를 이용하면 페이지의 개수가 3개 이상이 되면 실행시간이 400초 이상 걸린다. 특히 ELLIP과 같이 변수의 개수가 많은 설계인 경우에는 수행 시에 호스트 머신의 메모리 부족으로 최적해를 구할 수가 없었다. 그러나 zone_alignment의 방법을 이용하면 GAULEG, GAUJAC와 DBRENT의 경우에는 8초 이내에 해를 구할 수가 있었다. 또한 ELLIP의 경우에는 m이 6 이상일 때는 7초 이내에, m이 3 또는 4 이었을 때는 200초 이내에 해를 구할 수 있었다. zone_alignment는 Greedy [3] 방식에 비해 페이지 접근 회수를 약 3.5%만 증가시킨다. 이렇게 조금만 개선시키는 이유는 프로그램 내에서 변수 접근에는 지역성(locality)이 있기 때문에 간단한 greedy 방식만 써도 좋은 결과의 분할을 얻을 수 있기 때문이다.

표 1 벤치마크 프로그램들(|V|: 변수의 개수, |S|: 변수 접근 순서의 길이)

Benchmark	$\|V\|/\|S\|$	Description
BIQUAD	10/38	benchmarking of an one iir biquad [13]
CHEBEV	12/33	Chebyshev polynomial evaluation [15]
ELLIP	45/100	elliptical wave filter [14]
GAUJAC	23/148	Gauss-Jacobi weights and abscissas [15]
GAULEG	16/47	Gauss-legendre weights and abscissas [15]
DBRENT	24/109	find minimum of a function using derivative [15]

버스트 접근 회수를 최대화하기 위한 메모리 상의 변수 정렬 최적화: 표 3은 OFU, CGB, Greedy에 의한 버스트 접근 회수와 zone_alignment 방식에 의한 버스트 접근 회수의 개수를 비교하여 보여준다. zone_alignment를 이용했을 때, OFU나 CGB, Greedy 알고리즘을 이용했을 때 비해서 각각 84.8%, 113.5%, 10.1% 씩 버스트 접근 회수가 증가했다. 모든 벤치마크 프로그램에 대해서 zone_alignment를 이용하면 3초 이내에 결과를 얻을 수 있었다. 특히, Greedy[3] 방식에 비해서도 10.1%의 버스트 접근 회수 증가라는 꽤 의미있는 결과를 얻었는데, 이는 간단한 greedy방식이 버스트 접근 회수를 최대화시키기에는 부족하다는 점을 명확히 보여준다.

제안된 알고리즘의 수행시간 분석: 그림 6(a)와 (b)는 각각 zone_alignment 방식을 적용했을 때 ELLIP과 DBRENT의 수행 시간을 보여준다. ELLIP의 경우에는 m이 10, 12, 16일 때 변수 접근 순서를 두 개 이상의 *access_zone*으로 나누지 않고도 해를 구할 수 있었다. 따라서 구해진 해는 최적해이다. 그림 6에서 볼 수 있듯이 페이지 접근을 위한 zone_alignment 방식은 버스트 접근을 위한 zone_alignment 방식에 비해 수행시간이 길다. 이는 첫째로, 버스트 접근을 위해서 우리가 제안한 ILP 공식이 매우 간단하기 때문이며, 둘째로, 페이지 접근을 위한 ILP 의 탐색 영역이 버스트 접근을 위한 그것보다 상대적으로 넓기 때문이다.

본 연구의 실험의 데이타는 트래이스(trace)를 추출한 다음 알고리즘을 돌려서 얻은 결과이다. 실제로 구현하려면, 프로그램을 수행하여 프로파일(profile)을 하여 변수의 접근 순서를 추출한 다음, 구해진 변수 접근 순서를 컴파일러의 입력으로 주어, 컴파일러가 본 알고리즘을 돌리도록 한다. 레지스터 할당(register allocation)이 끝난 이후에, 실제로 변수(프로시저의 지역(local) 변수들)가 메모리에 액세스되는 경우들에 한해서, 변수의 오프셋(offset)을 제안된 알고리즘에 의해 정해주면 된다. 제안된 알고리즘은 다른 최적화 단계에 별 영향을 끼치지 않으므로 프로파일을 해야 되는 점을 제외하면 컴파일 시에 큰 오버헤드는 없다고 하겠다.

6. 결 론

본 논문은 DRAM 메모리가 제공하는 접근 방식 중, 페이지 접근 회수와 버스트 접근 횟수를 증가시키기 위한 DRAM 상의 변수 배열에 관한 효과적인 방식을 제안하였다. 구체적으로, (1) 페이지 접근 횟수를 극대화하기 위해서 zone_alignment 라는 거의 최적해에 근접하는 해를 구하는 알고리즘을 제안하였으며, (2) 버스트 접근 횟수의 극대화를 위한 zone_alignment 기법도 제안하였다. 실험을 통해 발견된 것으로 (2)의 수행시간이 (1)의 수행시간보다 훨씬 짧았지만, (2)의 결과가 해당 greedy 방식보다 훨씬 좋은 반면, (1)의 결과는 해당 greedy방식에 비해 별로 개선되지 못하였다. 이는, 페이지 접근 횟수를 최대화시키는 문제와 버스트 접근 횟수를 최대화시키는 문제는 별개의 방식으로 풀어 해결해야 함을 암시한다 하겠다. 본 연구는, 메모리 계층 구성상, 캐쉬 메모리가 없으며, 프로세서와 DRAM이 바로

표 2 OFU, CGB[1,2],Greedy[3] 및 zone_alignment에 의한 페이지 접근 회수

design	페이지 접근 회수(OFU/CGB[1,2]/Greedy[11]/zone_alignment) m:페이지 크기							이득 비교(%)		
	m=3	m=4	m=6	m=8	m=10	m=12	m=16	OFU	CGB	[11]
BIQUAD	18/19/19/20	19/23/23/23	22/27/29/29	24/33/33/33	-	-	-	25.4	3.1	1.3
CHEBEV	14/19/19/19	15/21/23/23	22/24/24/24	24/20/26/28	28/28/26/28	-	-	22.9	9.9	0
ELLIP	19/39/42/43	21/37/45/45	34/39/52/55	38/46/61/61	48/46/63/67	54/52/68/75	58/65/76/80	68.2	30.8	4.1
GAUJAC	56/67/68/71	63/71/80/82	84/81/95/99	104/87/106/113	107/107/115/120	117/112/116/126	125/125/128/129	15.2	14.5	4.5
GAULEG	37/44/43/49	26/25/34/34	31/31/34/38	36/36/41/42	36/42/43/43	43/38/44/44	-	21.9	16.2	3.1
DBRENT	37/44/43/49	43/49/53/58	44/45/64/69	52/64/72/76	56/65/77/81	65/72/82/85	71/89/88/94	39.7	16.2	7.5
avg. gain								32.2	15.1	3.5

표 3 OFU, CGB[1,2],Greedy[3] 및 zone_alignment에 의한 버스트 접근 회수

design	버스트 접근 회수(OFU/CGB[1,2]/Greedy[11]/zone_alignment) m:페이지 크기							이득 비교(%)		
	m=3	m=4	m=6	m=8	m=10	m=12	m=16	OFU	CGB	[11]
BIQUAD	8/9/9/9	10/10/10/11	10/6/11/12	11/6/11/12	-	-	-	12.9	52.5	7.0
CHEBEV	7/7/10/10	9/9/10/11	10/6/11/12	10/6/11/12	11/6/11/12	11/6/12/13	-	22.0	80.3	6.4
ELLIP	9/13/24/31	8/11/29/34	11/13/30/37	10/4/33/38	11/8/33/38	11/10/33/38	11/7/34/38	260.3	354.3	18.1
GAUJAC	13/25/25/32	15/26/29/35	17/28/31/37	17/24/34/38	17/27/34/38	15/20/34/38	17/19/34/38	132.9	47.3	16.4
GAULEG	9/13/13/14	13/11/15/16	14/8/17/18	14/6/17/18	12/11/18/18	15/13/18/18	-	34.3	80.0	5.3
DBRENT	17/16/24/25	18/19/25/27	19/21/28/30	21/16/28/30	22/23/29/31	22/18/29/31	22/14/29/32	46.4	66.3	7.2
avg. gain								84.8	113.5	10.1

연결되어 있는 내장형 시스템에서 가장 좋은 성능을 낼 수 있다. 캐쉬가 있는 시스템에서는, 캐쉬에 의해 대부분의 액세스 레이턴시(latency)가 가려지기 때문이다. 따라서 캐쉬가 있는 시스템에서 본 연구가 미치는 영향은 미미하다고 볼 수 있다. 그러나 그런 경우라 하더라도, 본 연구의 알고리즘을 이용하여 페이지 액세스나 버스트 액세스를 늘리면, DRAM를 액세스 하는데 드는 에너지 소모가 일부 감소할 수 있다. 또한 실시간 시스템에서는 타이밍 유지를 위해서 캐쉬가 없는 경우도 있으므로 이러한 시스템에서 쓰일 수 있다.

참 고 문 헌

[1] P. R. Panda, N. D. Dutt and A. Nicolau, "Exploiting Off-Chip Memory Access Modes in High-Level Synthesis," International Conference on Computer Aided Design, pp. 333-340, 1997.

[2] P. R. Panda, N. D. Dutt and A. Nicolau, "Memory Data Organization for Improved Cache Performance in Embedded Processor Applications," ACM Transactions on Design Automation of Electronic Systems, Vol. 2, No. 4, pp. 384-409, 1997.

[3] Y. Choi, and Taewhan Kim, "Memory Layout Technology for Variables utilizing Efficient DRAM Access Modes in Embedded Systems Design," Design Automation Conference, pp. 881-886, 2003.

[4] N. D. Dutt, "Memory Organization and Exploration for Embedded Systems-on-Silicon," International Conference on VLSI and CAD, 1997.

[5] IBM, "IBM Cu-11 Embedded DRAM Macro," http://www-3.ibm.com/chips/techlib/techlib.nsf/techdocs/ 4CBB96F927E2D6D287256B98004E1D98/$file/Cu11_embedded_DRAM.10.pdf, 2002.

[6] Fujitsu, "CS70DL Embedded DRAM," http://www.fme.fujitsu.com/products/asic/pdf/CS70DLFS.pdf, 1999.

[7] A. Khare, P. R. Panda, N. D. Dutt and A. Nicolau, "High-Level Synthesis with Synchronous and RAMBUS DRAMs," Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI), 1998.

[8] P. Grun, P. Grun, N. D. Dutt and A. Nicolau, "Memory Aware Compilation Through Accurate Timing Extraction," Design Automation Conference, pp. 316-321, 2000.

[9] P. Grun, N. Dutt and A. Nicolau, "APEX: Access Pattern Based Memory Architecture Exploration," International Symposium on Systems Synthesis (ISSS), pp. 25-32, 2001.

[10] K. Ayukawa, T. Watanabe, and S. Narita, "An Access Sequence Control Scheme to Enhance Random-Access Performance of Embedded DRAMs," IEEE Journal of Solid-State Circuits, Vol. 33, No. 5, pp. 800-806, 1998.

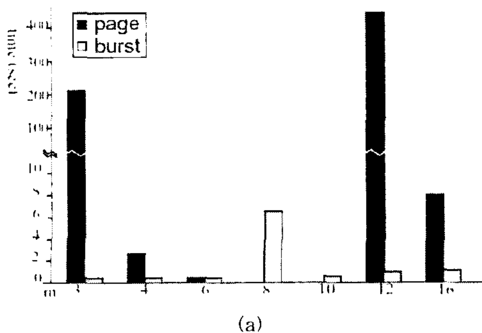
[11] S. Hettiaratchi, P. Cheung, and T. Clarke, "Energy Efficient Address Assignment Through Minimized Memory Row Switching," International Conference on Computer Aided Design, pp. 577-581, 2002.

[12] Laurence A. Wolsey, Integer Programming, Wiley-Interscience, 1998.

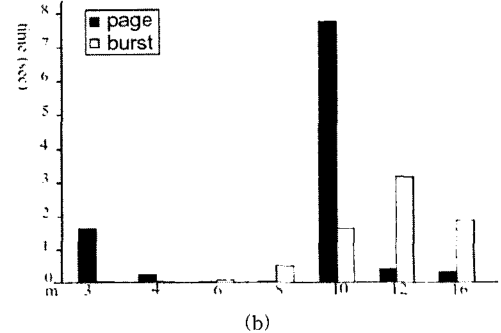
[13] V. Zivojnovic, J. Velarde, and C. Schlager, "Dspstone: A DSP-oriented Benchmarking Methodology," International Conference on Signal Processing Applications and Technology, pp.715-720, 1994.

[14] "Bench mark Archives at CBL," http://www.cbl.ncsu.edu/CBL_Docs/Bench.html

[15] W. H. Press, et al., Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, pp.152,154-155, 1993.



(a)



(b)

그림 6 zone_alignment의 수행시간 비교, (a) ELLIP, (b) DBRENT. (0.02초 이하는 그림으로 보이지 않음.)

최 윤 서

2000년 한동대학교 전산전자학부(학사) 2002년 한국과학기술원(석사). 현재 한국과학기술원 전자전산학과 전산학전공 박사과정. 관심분야는 Embedded systems

김 태 환

1993년 미국 일리노이 주립대 전산학과(박사). 현재 서울대학교 전기.컴퓨터 공학부 부교수. 관심분야는 Embedded system design