논문 2016-11-24

하이브리드 SPM을 위한 버퍼 공유를 활용한 새로운 버퍼 매핑 기법

(New buffer mapping method for Hybrid SPM with Buffer sharing)

이 대 영, 오 현 옥*
(Daeyoung Lee, Hyunok Oh)

Abstract : This paper proposes a new lifetime aware buffer mapping method of a synchronous da taflow (SDF) graph on a hybrid memory system with DRAM and PRAM. Since the number of writ e operations on PRAM is limited, the number of written samples on PRAM is minimized to maxim ize the lifetime of PRAM. We improve the utilization of DRAM by mapping more buffers on DRA M through buffer sharing. The problem is formulated formally and solved by an optimal approach of an answer set programming. In experiment, the buffer mapping method with buffer sharing imp roves the PRAM lifetime by 63%.

Keywords : Hybrid memory, Algorithms, Reliability, Synchronous dataflow graph, Mapping

I. 서 론

PRAM (Phase change RAM), STT-MRAM (Spin-transfer torque magnetoresistive RAM) 등의 비휘발성 메모리를 가지고 메인 메모리나 캐쉬 메모리 같은 기존의 메모리를 대체하는 새로운 메모리로 만드는 연구가 최근 활발하게 이뤄지고 있다. 비휘발성 메모리는 DRAM. SRAM 등의 휘발성 메모리에 비해 전력 소모가 적고, 높은 집적도를 갖는 장점이 있다. 그러나 비휘발성 메모리중 하나인 PRAM의 읽기 성능은 DRAM의 읽기 성능과 비슷하지만 PRAM의 쓰기 성능은 큰 레이턴시를 보이고 있어 PRAM을 메인 메모리로 완전히 대체하는 것에 한계를 보이고 있다 [1]. PRAM은 비트를 쓰기 위해 내부 구성 물질의 상태를 변화시키므로, 읽기에 비해 쓰기 동작 시, 높은 전력소모와 레이턴시가 발생된다. 또한, PRAM과 플래쉬 메모리의의 셀 당 쓰기 횟수(수명)은 제한적이다.

플래쉬 메모리의 경우 핫 데이터를 셀로 분산하는 방법을 통하여 수명을 늘림으로써 저장장치로 쓰이는 기법이 제안되었다 [2]. 그러나 PRAM을 DRAM을 대체하는 경우, 메모리가 사용되는 동안, 쓰기 동작이 저장장치의 쓰기 동작에 비해 매우 빈번하게 발생하므로, PRAM의 짧은 수명은 치명적인 단점으로 작용하게 된다. 일반적으로 PRAM의 쓰기 수명은 10^5~10^8회. 플래쉬 메모리의 쓰기 수명은 10^4~10^5회로 알려져 있으며, PRAM 수명의 제약조건이 고려되지 않은 경우, PRAM을 메인 메모리로 사용하였을 때는 고작 하루만 사용 가능한 것으로 알려져 있다 [2].

비휘발성 메모리의 수명을 늘리고, 에너지 소모를 줄이기 위해서 휘발성 메모리(DRAM)와 비휘발성 메모리(PRAM)으로 구성되는 하이브리드 메모리 구조가 제안되었다 [3]. 이러한 하이브리드 메모리 구조에서는 쓰기가 자주 발생하고, 업데이트가 잦은 데이터는 휘발성메모리로, 프로그램 코드나 상수 같이 읽기가 빈번한 데이터는 비휘발성 메모리로 할당하는 것이 성능 및 수명의 측면에서 유리하다. 임베디드 시스템에서는 어플리케이션이 짧게는 며칠, 길게는 수십 년의 기간 동안 안정적으로 동작할 것

*Corresponding Author (hoh@hanyang.ac.kr)
Received: 27 June 2016, Revised: 11 July 2016, Accepted: 26 July 2016.
D. Lee, H. Oh: Hanyang University
※ 본 논문은 2013년도 한국연구재단 (2013R1A1A1013384) 및 지식경제부/한국산업기술평가관리원(No. 10041608)의 지원을 받아 작성하였음.

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2016.11.4.209

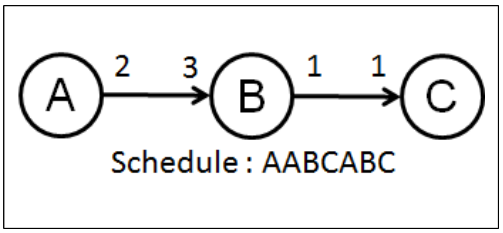

그림 1. SDF 그래프 예제
Fig. 1 Example of SDF Graph

을 요구한다. 임베디드 시스템은 일반적으로 전력 공급이 제한적인 장비들이 많아 에너지 소모를 최소화 하는 것은 비용이나 시간 못지않은 중요한 제약 조건으로 꼽히고 있다. 그러므로 장기적인 관점에서 비휘발성 메모리와 휘발성 메모리로 이뤄진 하이브리드 메모리 아키텍처는 메모리의 수명을 고려하여 전체 시스템의 수명 및 최소화된 에너지 소모를 보장하는 대안으로 사용될 것이다. 이러한 하이브리드 메모리 아키텍처에서 DRAM의 크기와 어떤 데이터가 DRAM으로 할당되는지 결정하는 문제를 풀기 위한 연구가 진행되었다 [4].

본 논문은 SDF(Synchronous DataFlow) 그래프로 기술된 어플리케이션을 대상으로 한다 [5]. SDF 모델은 주기적으로 메모리를 접근하는 멀티미디어 같은 스트리밍 어플리케이션을 명세하는데 매우 적합하다. SDF 그래프는 크게 노드(액터)와 엣지로 구성이 되는데, 노드는 C언어와 같은 프로그래밍 언어에서의 함수와 같은 역할을 담당하고, 엣지는 두 노드간의 통신 채널의 역할을 처리하게 된다. 각각의 엣지에서 노드에서 생성하고 대상 노드에서 소비되는 샘플(토큰)의 개수를 표기하고, 이를 각각 생성 비, 소비 비라 부른다. 이 비는 정수로 정해지므로, SDF 그래프를 분석하여 정적 스케줄, 즉 노드의 실행순서를 만들고, 메모리 크기가 정해진 코드를 생성할 수 있게 된다. 또한 SDF 그래프를 스케줄에 따라 수행하여 어플리케이션을 합성할 때, 각 엣지 마다 노드 사이에 처리될 샘플 데이터를 저장할 메모리 공간 또는 버퍼 배열을 할당할 수 있다. 버퍼 공유는 샘플의 크기가 크거나, 샘플의 비 변화가 심할 때, 할당되는 버퍼의 크기를 줄이기 위한 기법이다. [6]에서는 글로벌과 로컬 버퍼로 구성된 버퍼를 사용하여, 글로벌 버퍼는 실제 버퍼를, 로컬 버퍼는 글로벌 버퍼의 오프셋을 나타내는 포인터를 저장한다. 각 엣지에서 처리되는 샘플들은 글로벌 버퍼를 공유하여 저장되고, 로컬 버퍼에는 이에 대한 포인터 값만 갖게 되므로 버퍼 공유

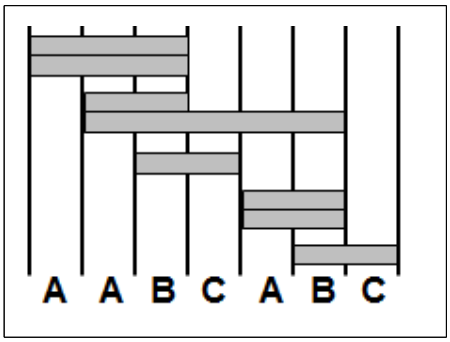

그림 2. 그림 1의 버퍼 수명 주기 차트
Fig. 2 Lifetime chart of Fig. 1

를 하지 않은 할당 결과에 비하여 실제로 할당되는 버퍼의 크기를 줄일 수 있었다.

그림 1은 SDF 그래프의 예제이다. 이 그래프는 3개의 노드 A, B, C와 2개의 엣지 AB, BC로 구성되어 있음을 알 수 있다. 노드에는 각각 생성되고 소비되는 샘플의 수가 명시되어 있다. 노드 A에서는 그래프가 수행될 때 마다 2개의 샘플을 노드 B로 보내고 노드 B의 경우 노드 A로부터 3개의 샘플을 입력 받아 1개의 샘플을 생성하여 노드 C로 보내는 역할을 담당한다. 노드 C는 노드 B에서 1개의 샘플을 받아 소모한다. 이러한 그래프를 가지고 각 노드가 몇 회씩 수행되어야하는지 계산한 뒤 어떤 순서로 수행할 지를 정함으로써 각각 다른 버퍼 사이즈를 갖는 여러 개의 유효한 스케줄을 얻을 수 있다. 그림 1의 그래프에 속한 노드의 반복 횟수는 순서대로 3회, 2회, 2회씩 계산된다. 본 논문에서는 가능한 스케줄 후보 중에 버퍼 크기를 최소화하는 스케줄을 대상으로 한다. 모든 엣지는 각각 4개,1개의 최소 버퍼 크기를 가진다.

버퍼 공유를 적용하기 위해서는 각 버퍼의 수명 주기(Lifetime)이 필요하다. 그림 2는 그림 1의 SDF 그래프에 대해서 수명 주기 (Lifetime)을 계산한 결과이다. 첫 번째 노드 A가 수행될 때 2개의 샘플이 생성되며, 이는 스케줄 “AABCABC”에서 “AAB”까지 샘플이 유지됨을 볼 수 있다.

[7]에 따르면, 하이브리드 메모리 구조를 갖는 시스템에서는 더 많은 버퍼가 DRAM 공간으로 할당되기 위해서는 그래프의 버퍼 사이즈가 최소화되는 스케줄이 선택되어야 한다. 또한, 스케줄에 관계없이 그래프가 수행되는 분기마다 모든 노드에서 생성되는 샘플의 개수는 변하지 않기 때문에, 샘플을 덜 생성하는 엣지가 PRAM으로 할당이 된다면,

쓰기 동작의 횟수가 줄어들고, 이로 인해 PRAM의 수명이 증가할 것으로 예측할 수 있다. 그러나 [7]에서는 버퍼 공유를 적용하지 않아서 버퍼 사이즈를 줄이는 것에 한계가 있다.

임베디드 시스템이 널리 쓰이고 있고. 이러한 시스템에는 전통적인 방식의 캐쉬보다 SPM(Scratch Pad Memory) 이 더 적합하다. SPM은 기존의 캐쉬 메모리에 비해 더 적은 공간을 차지하고 낮은 에너지 소모를 보이기 때문에 일반적으로 전력 공급이 제한적인 임베디드 시스템에 더욱 알맞은 것으로 보인다. 그러나 SPM의 전체 전력소모의 33.7%가 트랜지스터 변환에 의한 누출 전력 [8]이므로 이 문제를 해결하기 위한 연구가 진행되고 있다.

SRAM으로만 구성된 기존의 SPM을 대체하여 비휘발성 메모리를 함께 사용한 하이브리드 SPM이 등장하고 있다 [9]. 본 논문에서는 이러한 하이브리드 SPM을 내장한 시스템을 대상으로 하여 SDF 그래프와 주어진 스케줄, 버퍼 사이즈를 제약조건으로 가지고 하이브리드 SPM의 PRAM 수명을 최대화하는 최적화된 버퍼 할당 결과를 찾는 새로운 매핑 기법을 제안한다.

II. 관련 연구

PRAM과 같은 비휘발성 메모리는 낮은 전력소모, 높은 집적도, 빠른 읽기 성능 등의 장점을 가지고 있어 이를 기존의 DRAM을 대체하는 차세대 메모리로 각광받고 있다. 그러나 비휘발성 메모리는 낮은 쓰기 성능과 제한적인 수명 등의 단점이 있다. 이러한 단점을 보완하기 위한 방법으로 DRAM과 PRAM을 혼합한 하이브리드 메모리가 제안되었다 [1].그 외에도 PRAM이나 플래시메모리의 수명을 늘리기 위해 쓰기 작업을 최소화하는 기법이 제안되었는데, 이는 시뮬레이션을 통하여 어플리케이션의 쓰기동작을 미리 분석함으로써 각 시나리오 별로 다른 쓰기 전력을 적용하는 방식이다 [10]. 플래쉬 메모리를 DRAM에 붙여 어플리케이션 종속적인 메인 메모리 아키텍처와 각 프로세서 코어마다 스크래치 패드 메모리를 연결함으로써, 비휘발성 메인 메모리의 쓰기 작업을 줄이는 아키텍처가 제안되었다 [11]. 또한 플래쉬 메모리에 쓰이는 웨어-레벨링 기법을 PRAM에 맞게 수정한 기법이 제안되었다 [12].

DRAM에 SDF 그래프의 버퍼를 더 많이 할당하기 위하여 버퍼 사이즈는 최소화 되어야 한다. 또한, 코드 사이즈와 버퍼 사이즈 사이에 트레이드오프가 있기 때문에 버퍼 사이즈를 최소화 하는 경우 둘을 같이 고려하여 접근해야 한다. APGAN과 RPMC는 최적의 SAS(Single Appearance Schedule)을 찾는 휴리스틱으로써, SAS란 코드 메모리 사이즈를 최소화하기 위하여 노드가 한 번씩 수행되도록 배치되는 스케줄을 뜻 한다 [13]. 동적 루프 카운트 SAS 기법은 런타임에 각 노드의 루프 카운트를 계산하는 오버헤드가 있지만 non-SAS를 허용함으로써 버퍼 사이즈를 최소화하는 기법이다 [14].

하이브리드 DRAM/PRAM 아키텍처를 대상으로 하여 최적화된 버퍼 할당결과를 찾는 연구도 진행되었다. 이는 기존의 버퍼 사이즈 최소화 스케줄이 전체 버퍼 사이즈를 줄이는 것에 중점을 둔 것과 달리 DRAM에 할당되는 버퍼의 사이즈를 줄이는 것에 초점을 맞췄다. 하이브리드 메모리에서는 DRAM은 전력소모가 크기 때문에 DRAM의 크기는 최소화하는 동시에, PRAM의 수명이 제한적이기 때문에 자주 쓰기가 이뤄지는 데이터는 DRAM으로 할당되도록 해야 한다. 이 연구에서는 주어진 DRAM/PRAM 사이즈를 가지고 PRAM의 수명을 극대화 하는 문제와 주어진 PRAM의 수명 제약조건을 가지고 DRAM의 사이즈와 에너지소모를 최소화하는 문제를 정의하고 이를 해결하는 최적화된 버퍼 할당 알고리즘을 제안하고 있다 [7].

SPM은 소프트웨어로 관리되는 메모리로써, 최근 많은 임베디드 시스템에서 기존의 캐쉬 메모리의 대안으로 활용 되고 있다.

SPM을 내장한 시스템에서 SDF 그래프를 대상으로 성능 최적화 알고리즘을 제안하는 연구에서는 SDF 그래프에서 얻는 정적인 정보를 활용하여 SPS과 메인 메모리 사이의 데이터 전송 레이턴시를 줄이거나 없애는 것을 목표로 한다. 그러나 이 연구에서는 메모리 셀의 수명을 고려하지 않았으며, SPM과 메인 메모리 간의 통신에만 초점을 맞춘 것이 본 논문과 차이점이다 [15, 16].

하이브리드 SPM은 기존의 SPM이 SRAM이나 DRAM으로만 구성된 것과 달리 PRAM이나 STT-MRAM 등의 비휘발 메모리를 함께 사용하는 메모리이다. 전통적인 SPM을 대상으로 한 많은 정적/동적 데이터 관리 알고리즘들이 존재하지만 [17, 18], 이러한 알고리즘들은 읽기/쓰기 동작을 구별하여 동작하지 않기 때문에, 쓰기동작에서 약점을 갖는 비휘발성 메모리를 사용하는 하이브리드 SPM에는 적합하지 않다. 따라서 하이브리드 SPM의 특성에 맞

는 새로운 데이터 관리 알고리즘이 필요하다

SPM을 내장한 멀티프로세서 시스템 상에서 SDF 그래프를 대상으로 SPM의 실행 레이턴시를 최소화하는 기법을 제안한 연구가 진행되었다 [19]. 전통적인 SDF 그래프의 멀티프로세서 스케줄링과는 달리, SPM 사이즈가 제한되어 있고, 코드와 데이터의 오버레이가 발생되는 상황에서 발생하는 오버헤드를 줄이는 문제를 Genetic Algorithm을 통하여 최적화된 값을 얻어내는 기법을 제안하고 있다.

비휘발성 메모리의 수명 한계를 극복하기 위하여 PCM 메모리위에 작은 듀얼 DRAM 버퍼를 연결하는 연구가 진행되었다. 이를 통해 기존 DRAM 기반의 메인메모리에 비해 많은 전력 소모 감소 효과를 얻었으나, 약간의 성능저하를 갖는 결과를 보였다 [20].

[21]에서는 NVM 기반 캐시의 쓰기 성능, 전력 및 내구성 향상을 위한 다양한 최적화 기법을 구현, 평가 하기 위한 캐시 시뮬레이터의 설계를 제시하였다. 기존의 캐시 시뮬레이터와 달리 NVM 기반 캐시 시뮬레이터에서 필요한 기능에 대해 다루고 있다.

Ⅲ. 버퍼 매핑 문제 정의 및 알고리즘

1. 문제 정의

본 논문에서는 SDF 그래프로 나타낸 어플리케이션을 대상으로 한정된 크기를 갖는 DRAM과 충분히 큰 크기를 갖는 PRAM으로 구성되는 하이브리드 메모리 구조에서 PRAM의 수명을 최대한 길게 유지할 수 있도록 버퍼 공유를 적용하여 최적화된 버퍼 매핑을 찾는 문제를 정의한다.

2. 입력값

- SDF 그래프
- 주어진 SDF 그래프의 스케줄
- 스케줄에 대한 최소화된 버퍼 크기
- DRAM 크기
- 주어진 스케줄에서 계산된 엣지의 수명 주기

3. 용어 및 변수 정의

표 1은 본 논문에서 정의한 버퍼 매핑 문제를 수식화하기 위해 필요한 용어 및 변수를 정리한 것이다. SDF 그래프 $G=(V,E)$는 노드의 집합 V와 엣지의 집합 E의 조합으로 구성된다. 개별 노드와 엣지는 각각 v_i와 e_i로 정의하며, i는 자연수로

표 1. 문제 수식 용어
Table 1. Problem formulation terms

Term	Meaning
$G=(V,E)$	Graph G consisting of Node V and Edge E.
v_i, e_i	ith Node v, Edge e
$s(e_i,j)$	The number of samples on e_i at jth schedule
$bs(e_i)$	The buffer size of e_i
$offset(e_i)$	The offset value of e_i
$p(e_i)$	The boolean value of memory allocation status of e_i(if this value is 0, e_i is alloacted to DRAM. Otherwise, e_i is allocated to PRAM.)
$t(e_i,j)$	The number of producing samples on $e_i$at jth schedule
$minbuf(e_i)$	The maximum value of $s(e_i,j)$
$maxbuf(e_i)$	The sum of initial sample and total sample which can be produced on e_i at one period.
D_{size}	DRAM size

표기한다. 예를 들어, 그림 1의 SDF 그래프에서 노드 A는 v_1, 노드 B는 v_2, 노드 C를 v_3라고 정의한다. 그리고 노드 A와 B를 연결하는 엣지 AB를 e_1, 노드 B와 C를 연결하는 엣지 BC를 e_2라고 정의한다. 이 그래프에서는 버퍼크기를 최소화하는 스케줄 “AABABCC”를 구할 수 있다. 이 스케줄을 가지고 그래프가 수행되는 동안 그림 2와 같이 버퍼의 수명주기(Lifetime)를 구할 수 있다. $s(e_i,j)$는 그래프가 수행될 때, j번째 스케줄에서 엣지e_i가 샘플을 생성하는 개수를 의미한다. 그림 2에서 “AAB”를 잇는 가장 위의 양방향 화살표가 엣지 e_1의 첫 샘플 생성을 의미하며 이를 $s(e_1,1)$로 표기한다.

$bs(e_i)$는 각 엣지 e_i의 버퍼 크기를 의미한다. 이는 주어진 그래프에서 얻어진 스케줄을 통하여 미리 계산가능하다. 그림 1의 SDF 그래프에서는 $bs(e_1)=4, bs(e_2)=1$로 크기를 정할 수 있다.

$offset(e_i)$는 각 엣지 e_i가 DRAM에 할당 되었을 때, 메모리 영역의 위치 값을 저장한다.

$p(e_i)$는 엣지 e_i가 PRAM으로 할당될지를 결정하는 이진 변수이다. 1의 값을 가지면 PRAM으로,

0의 값을 가지면 DRAM으로 할당이 됨을 의미한다. 본 논문에서는 모든 버퍼는 우선 DRAM에 할당됨을 가정하고, DRAM의 공간이 부족할 경우, 아래에 설명할 제약조건에 따라 PRAM으로 보낼 버퍼를 결정하게 되고, 해당 버퍼는 $p(e_i)$의 값을 1로 갖게 된다.

e_i의 스케줄 j번째 수행에서 생성되는 샘플의 개수는 $t(e_i,j)$로 정의한다. 그림 1 그래프에서 엣지 e_1의 경우, $bs(e_1)=4$로 계산되지만, 실제로 노드가 수행되어 발생되는 샘플의 수는 $(e_1,1)$일 때 2개이므로 이 값을 저장하기 위한 변수로 $t(e_i,j)$을 선언한다. 따라서 $t(e_1,1)=2$임을 알 수 있다.

버퍼 공유를 적용하여 버퍼를 할당하기 위해 각 엣지의 최소 버퍼 크기와 최대 버퍼 크기를 계산한다. 이 값들은 각각 $minbuf(e_i)$와 $maxbuf(e_i)$으로 정의한다. 그림 1 그래프에서 첫 번째 엣지 e_1의 경우, $minbuf(e_1)=4$, $maxbuf(e_1)=6$ 임을 알 수 있다.

D_{size}는 DRAM의 크기를 의미한다. 실제 하이브리드 메모리 구조에서는 PRAM과 DRAM의 크기가 정해져있으나, 본 논문은 문제정의와 풀이를 단순화하기 위하여 PRAM은 모든 SDF 그래프의 버퍼를 전부 담을 수 있을 만큼 충분히 큰 크기를 가진다고 가정하고, DRAM은 미리 정해진 크기를 가지고 있다. PRAM은 DRAM의 크기가 2인 경우, 그림 1의 SDF 그래프의 엣지 e_1은 PRAM, e_2는 DRAM에 할당되게 되며 PRAM에 write되는 전체 샘플의 수는 6이 된다.

4. 제약조건

최적화된 버퍼 할당 결과를 찾기 위한 제약조건들은 다음과 같다.

(1) DRAM 메모리에 할당될 버퍼는 메모리 주소값과 버퍼크기의 합이 DRAM의 크기를 초과할 수 없다.

$$offset(e_i)+bs(e_i) \le D_{size} \tag{1}$$

(2) DRAM 메모리에 할당될 서로 다른 두 엣지의 버퍼 할당 영역이 겹쳐지지 않아야 한다.

$$\begin{aligned}&\neg(\text{if } p(e_i)=0 \text{ and } p(e_i')=0,\\ &lifetime_{s(e_i,j)} \le lifetime_{s(e_i',j')} \le lifetime_{f(e_i,j)}\\ &(offset(e_i) \le offset(e_i') \le offset(e_i)+bs(e_i)\\ &\text{or } offset(e_i') \le offset(e_i) \le offset(e_i')+bs(e_i')))\end{aligned} \tag{2}$$

5. 목표

PRAM의 수명을 늘리기 위한 최적화된 버퍼 할당은 PRAM에 할당될 엣지에서 발생되는 샘플의 수를 최소화하는 것으로 정의할 수 있다. 이를 수식화 하면 다음과 같다.

$$\text{minimize} \sum_{e_i \in E} p(e_i,j)*t(e_i,j) \tag{3}$$

6. 문제 해결

본 논문에서 정의한 버퍼 할당 문제는 Answer Set programming(ASP)을 사용하여 수식으로 표현하고 최적화된 값을 얻었다 [20]. Answer Set Programming은 Interger linear programming(ILP), Genetic Algorithm(GA)와 같이 NP – hard 문제를 수식으로 모델링하여 최적값을 찾는 프로그래밍 언어이다. ASP는 사용자가 정의한 문제를 논리적인 술어로 간략하게 표현하여 주어진 논리 제약조건을 만족하는 Answer set 후보군에서 최적값을 빠르게 찾는다. ASP는 다음과 같은 형태로 표현된다.

```
(a) r:- p1,p2,...,not q1,not q2.
(b) t1.
(c) :- f1, f2.
```

(a)는 만약 p1 과 p2가 참이고, q1과 q2가 거짓일 경우 r은 참임을 나타낸다. 그렇지 않은 경우 해당 조건은 무시된다. (b)는 t1은 참임을 나타내는 간단한 표현이다. (c)는 f1이나 f2 중 하나는 거짓이어야만 하는 조건을 나타낸다. ASP에서는 변수는 항상 대문자로 시작한다.

```
1 #begin_lua
2 function gcd(a,b)
3 if a==0 then return b
4 else     return gcd(b%a, a)
5 end
6 end
7 #end_lua.
8 numEdges(NumEdges) :- NumEdges = #sum[edge(E,_,_,_,_,_)].
9 sdfedgegcd(E,@gcd(P,C)) :- edge(E,_,_,P,C,_).
10 possibleRepetition(Source,C/G,0) :- edge(E,Source,Sink,_,C,_), sdfedgegcd(E,G).
11 possibleRepetition(Sink,P/G,0) :- edge(E,Source,Sink,P,_,_), sdfedgegcd(E,G).
```

```
12 possibleRepetition(Source,C*X/P,N+1) :- possibleRepetition(Sink,X,N), edge(E,Source,Sink,P,C,_), (C*X #mod P) ==0, N < NumEdges, numEdges(NumEdges).
13 possibleRepetition(Sink,P*X/C,N+1) :- possibleRepetition(Source,X,N), edge(E,Source,Sink,P,C,_), (P*X #mod C) ==0, N < NumEdges, numEdges(NumEdges).
14 possibleRepetition(Instance,Rate,N+1) :- possibleRepetition(Instance,R1,N), possibleRepetition(Instance,R2,N), R1! =R2, (R1 #mod R2)!=0, (R2 #mod R1) !=0, Rate=R1*R2/@gcd(R1,R2), N < NumEdges, numEdges(NumEdges).
15 repetition(Instance, X) :- X = #max[possibleRepetition(Instance,R,_)=R ], node(Instance).
16 inconsistence :- repetition(Source,X), repetition(Sink, Y), edge(E,Source,Sink,P,C,_), X*P!=Y*C.
```

최적화된 버퍼 할당 결과를 찾는 과정은 주어진 SDF 그래프에 대해 먼저 최소 버퍼 크기를 갖는 스케줄을 찾고(Step 1), 이 스케줄에 대해 버퍼 공유를 적용하여 다시 최적화된 버퍼 할당 결과를 찾는(Step 2) 2단계 과정으로 구성된다. 다음은 본 논문에서 정의한 할당 문제를 ASP로 수식화 한 것이다.

위의 ASP 코드는 SDF 그래프의 스케줄을 구하는 첫 단계로, 각 노드의 수행 횟수를 계산한다.

```
1 totalInvocations(N) :- N = #sum[repetition(Node,R)=R].
2 time(1).
3 time(T+1) :- time(T), T<N, totalInvocations(N).
4 sample(E,I,0) :- edge(E,_,_,_,_,I).
5 notfireable(B,T) :- sample(E,S,T-1), S<C, edge(E,_,B,P,C,I), time(T).
6 :- fire(A,T), notfireable(A,T), T<N, totalInvocations(N), time(T).
7 sample(E,S+P,T) :- fire(A,T), edge(E,A,_,P,C,I), sample(E,S,T-1), time(T).
8 sample(E,S-C,T) :- fire(B,T), edge(E,_,B,P,C,I), sample(E,S,T-1), S>= C, time(T).
9 sample(E,S,T) :- sample(E,S,T-1), not fire(A,T), not fire(B,T), edge(E,A,B,_,_,_), time(T).
10 1 { fire(I,T) : node(I) } 1 :- time(T).
11 X { fire(I,T) : time(T) } X :- node(I), repetition(I,X).
12 sample(E,I,0) :- edge(E,_,_,_,_,I).
13 notfired(N,T) :- time(T), N = #sum[node(A)].
14 numFired(A,0,0) :- node(A).
15 notfireable(B,T) :- sample(E,S,T-1), S<C, edge(E,_,B,_,C,I), time(T).
16 fire(A,T) :- not notfireable(A,T), time(T),notfired(A,T), numFired(A,T-1,N), repetition(A,X), N<X.
17 numFired(A,T,N+1) :- fire(A,T), numFired(A,T-1,N).
18 numFired(A,T,N) :- not fire(A,T), numFired(A,T-1,N), time(T).
19 notfired(A-1,T) :- not fire(A,T), notfired(A,T), node(A), node(A-1).
20 sample(E,S+P,T) :- fire(A,T), edge(E,A,_,P,C,I), sample(E,S,T-1), time(T).
21 sample(E,S-C,T) :- fire(B,T), edge(E,_,B,P,C,I), sample(E,S,T-1), S>= C, time(T).
22 sample(E,S,T) :- sample(E,S,T-1), not fire(A,T), not fire(B,T), edge(E,A,B,_,_,_), time(T).
```

위의 코드는 이전 코드를 수행하여 얻은 각 노드의 수행 횟수를 가지고 스케줄의 길이 {time(T)}를 구하고 스케줄대로 그래프가 수행되는 동안 각 노드가 실행되는지 {fire(A,T)} 여부와 노드가 실행될 때 생성되는 샘플의 수를 계산한다.

```
1 bufSize(E,U) :- U = #max[sample(E,S,_)=S], edge(E,_,_,_,_,_).
2 totalBufSize(W) :- W = #sum[bufSize(_,U)=U].
3 #minimize [totalBufSize(W)=W].
```

위의 코드는 앞서 계산된 노드의 수행 횟수와 스케줄의 길이, 스케줄에 따라 노드가 수행되는지 확인한 결과, 각 엣지에서 처리될 샘플의 수를 가지고 최소 버퍼 크기를 계산하여 전체 버퍼의 크기를 최소화하는 결과값을 찾는다. 위의 3개 코드를 돌려서 대상 SDF 그래프의 최소 버퍼 크기를 갖는 스

케줄을 찾는 과정인 Step 1에 해당한다. 그림 1의 SDF 그래프를 ASP로 표현하여 Step 1을 수행한 결과는 다음과 같다.

```
1 node(1..3).
2 edge(1,1,2,2,3,0).
3 edge(2,2,3,1,1,0).
4 repetition(3,2). repetition(2,2). repetition(1,3).
5 totalBufSize(5).
```

Step 1에서 계산된 결과는 아래에서 설명할 Step 2의 입력값으로 쓰인다. Step 1과 Step 2를 한 번에 수행할 경우, 스케줄을 찾으면서 버퍼 공유를 적용한 버퍼 할당 결과도 찾게 되면서 수행시간에 오래 소요되기 때문에 스케줄을 찾는 과정과 버퍼 공유를 적용한 버퍼 할당 결과를 나누어 스케줄을 찾은 뒤 이를 입력값으로 하여 버퍼 할당 결과를 찾는다. Step 2는 Step 1에서 구한 SDF 그래프의 스케줄과 버퍼 크기를 가지고 버퍼 공유를 적용하여 PRAM의 수명을 최대화 할 수 있도록 PRAM에 할당되는 엣지에서 처리될 샘플의 수를 최소화 하는 버퍼 할당 결과를 찾는다.

```
1 minBuf(E,Min) :- Min = #max[time(1), sample(E,S,T)=S:time(T)],edge(E,_,_,_,_,_).
2 maxBuf(E,P*X+I) :- edge(E,N,_,P,_,I), repetition(N,X).
3 offset(1).
4 offset(Addr+1) :- offset(Addr), Addr<=dramSize.
5 1 {bufferSize(E,Size) : Size=Min..Max} 1 :- edge(E,_,_,_,_,_), minBuf(E,Min), maxBuf(E,Max).
6 1 {base(E,Offset):offset(Offset)} 1 :- edge(E,_,_,_,_,_), dram(E)
7 readOffset(E,Base,0,Base,Size) :- base(E,Base),bufferSize(E,Size).
8 readOffset(E,Offset+C,T,Base,Size):- readOffset(E,Offset,T-1,Base,Size), fire(BN,T-1),edge(E,_,BN,_,C,_),Offset+C< Base+Size,time(T).
9 readOffset(E,Offset+C-Size,T,Base,Size) :- readOffset(E,Offset,T-1,Base,Size),fire(BN,T-1),edge(E,_,BN,_,C,_),Offset+C<Base+Size,time(T), Offset+C-Size>=0.
10 readOffset(E,Offset,T,Base,Size) :- readOffset(E,Offset,T-1,Base,Size), not fire(B,T-1),edge(E,_,B,_,C,_),time(T).
11 writeOffset(E,O+N-Size,T,Base,Size) :- readOffset(E,O,T,Base,Size), sample(E,N,T), O+N>Size+Base.
12 writeOffset(E,O+N,T,Base,Size) :- readOffset(E,O,T,Base,Size), sample(E,N,T),O+N<=Size+Base.
13 occupied(E,Offset,T,Base,Size) :- readOffset(E,R,T,Base,Size),writeOffset(E,W,T,Base,Size),R<W,R<=Offset,Offset<W, offset(Offset).
14 occupied(E,Offset,T,Base,Size) :- readOffset(E,R,T,Base,Size),writeOffset(E,W,T,Base,Size),R>W,Base<=Offset,Offset<W,offset(Offset).
15 occupied(E,Offset,T,Base,Size) :- readOffset(E,R,T,Base,Size),writeOffset(E,W,T,Base,Size),R>W,R<=Offset,Offset<Base+Size,offset(Offset).
16 occ(E,O,T) :- occupied(E,O,T,_,_).
17 {occ(E,O,T) : edge(E,_,_,_,_,_)} 1 :- offset(O),time(T).
18 :- bufferSize(E,S), base(E,Addr), S+Addr-1 > dramSize.
```

위의 코드는 각 엣지별 최소 버퍼 크기와 최대 버퍼 크기를 계산하고, 버퍼 공유를 적용하여 최종 버퍼 크기와 엣지가 Dram에 할당될 때의 메모리 영역의 기본 위치(base offset)를 설정하는 역할을 담당한다. 이 코드에서 얻어지는 최대 위치값(maximum offset)은 전체 버퍼 사이즈에 해당한다. 앞서 설정한 2개의 제약조건은 이 ASP 코드에 포함되어 있다.

```
1 numSamples(E,S) :- S=P*X, edge(E,A,B,P,C,I), repetition(A,X).
2 0 { dram(E) : edge(E,_,_,_,_,_) }.
3 pram(E):-not dram(E), edge(E,_,_,_,_,_).
4 numTotalSamplesOnPRAM(S):-S=#sum[pram(E)=N:numSamples(E,N)].
5 #maximize [dram(E)=N : numSamples(E,N)].
```

위의 ASP 코드는 엣지가 PRAM에 할당되는

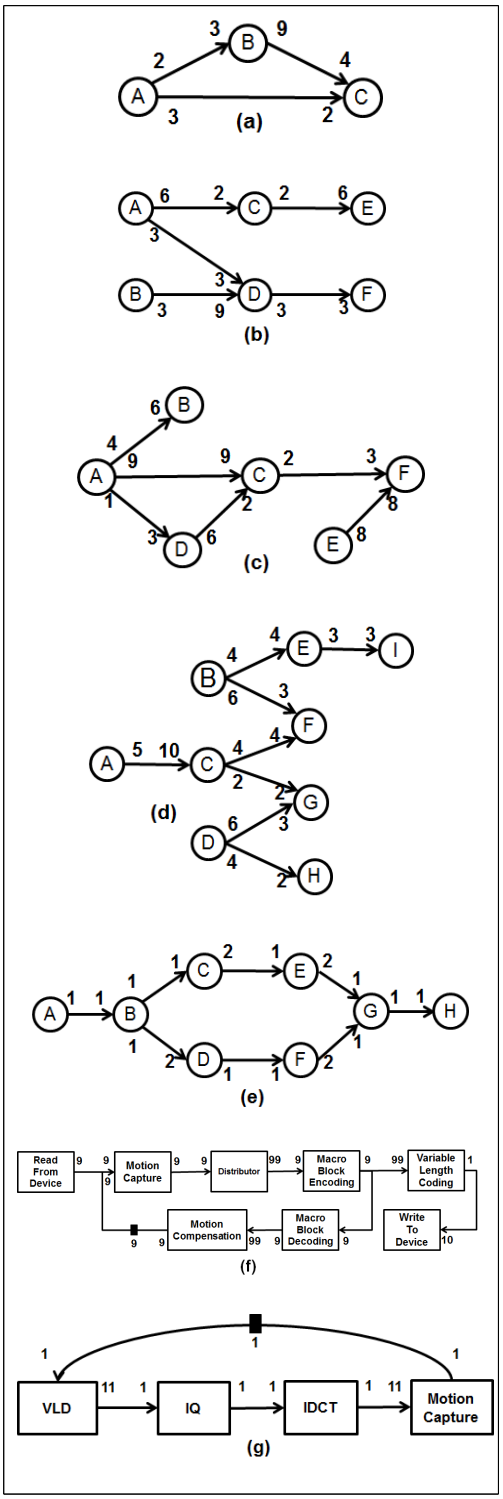

그림 3. SDF 실험 예제
Fig. 3 Experiments of SDF graph

표 2. PRAM 수명 비교
Table 2. PRAM Lifetime comparison

	Dram Size	E1 (Exectime)	E2 (Exectime)
(a)	15	30(2.38)	30(21.6)
(b)	10	18(0.03)	9(0.25)
	15	12(0.02)	3(0.26)
(c)	10	54(0.12)	55(1.52)
	15	45(0.16)	49(1.6)
(d)	10	27(1.15)	12(8.03)
(e)	5	7(0.04)	0(0.24)
(f)	10	63(0.5)	18(0.51)
	20	54(0.5)	0(0.52)
(g)	5	22(0.68)	22(145.13)
(h)	7	8(0.35)	8(0.09)
	10	6(0.27)	6(6.01)

경우, 해당 엣지에서 생성되어 소비되는 샘플의 수를 계산하고 PRAM에 할당되는 샘플의 수를 최소화하는 버퍼 할당 결과를 찾는 수식을 나타내고 있다.

Ⅳ. 실 험

우리는 제안하는 버퍼 공유를 고려한 새로운 매핑기법이 기존의 기법 [7]에 비해 비휘발성 메모리(PRAM)의 수명을 더 향상시킬 수 있음을 보이기 위해 랜덤하게 생성한 SDF 그래프와 SDF 그래프로 표현한 H.263 인코더와 H.264 디코더, MP3 디코더에 대해 실험을 진행하였다.

본 논문에서는 ASP로 수식화된 문제를 풀기위한 solver로써 clingo를 이용하여 버퍼 할당 문제를 ASP 코드로 명세하고 최적값을 도출하였다 [22]. 실험은 Intel i5 2.7GHz의 성능을 가진 컴퓨터를 이용하여 진행 하였다.

다음 그림 3은 실험에 사용된 SDF 그래프를 그림으로 나타낸 것이다.

표 2는 기존의 버퍼 매핑 기법(E1)과 본 논문에서 제안하는 매핑 기법(E2)으로 얻은 매핑 결과에 따라 PRAM에 할당되는 샘플의 수와 계산에 걸린 시간(Exectime)과 DRAM의 크기를 나타내고 있다. 수행시간은 sec 단위, DRAM은 MB 단위로 표기했다.

E2의 수행시간은 Step 1과 Step 2를 더한 시간을 표기했다(평균적으로 Step 1의 수행시간이 Step 2보다 더 오래 소요되었다). 그래프 (c)의 경우 2단계로 나눠서 수행하면서 할당결과가 기존 결과보다 좋지 않게 나온 것으로 보인다. 그래프 (g)의 경우 수행시간이 다른 그래프에 비해 길게 소요됨을 볼 수 있다. 일반적으로 PRAM은 쓰기 동작 횟수가 제한적이기 때문에, PRAM에 할당되는 생성되는 샘플의 수를 줄이는 것은 PRAM의 쓰기 동작을 줄임으로써 비휘발성 메모리의 수명을 늘릴 것으로 생각할 수 있다. 시험 결과 평균 약 63%의 수명 향상(E1-E2/E2)을 보인다.

V. 결 론

PRAM과 같은 비휘발성 메모리는 전력소모와 속도 측면에서 기존의 DRAM에 비해 장점을 갖고 있으나, 읽기에 비해 쓰기 속도가 느리고 횟수가 제한적인 단점을 가지고 있다. 본 논문에서는 하이브리드 SPM을 내장한 시스템 상에서 SDF 그래프를 대상으로 PRAM의 수명을 최대로 늘리기 위하여 버퍼 공유를 적용한 새로운 버퍼 할당 기법을 제안한다. PRAM의 수명을 늘리기 위한 최적화된 버퍼 할당은 PRAM에 할당될 엣지에서 발생되는 샘플의 수를 최소화하는 것으로 정의할 수 있다. 이는 버퍼 공유를 고려하지 않은 기존의 할당 기법에 비하여 약 63%의 수명 향상을 기대할 수 있다.

References

[1] G. Dhiman, R. Ayoub, T. Rosing, “PDRAM: A Hybrid PRAM and DRAM Main Memory System,” Proceedings of ACM/IEEE Design Automation Conference, pp. 664-669, 2009.

[2] Y. Chang, J. Hsieh, T. Kuo, “Endurance Enhancement of Flash-Memory Storage Systems: An Efficient Static Wear Leveling Design,” Proceedings of ACM/IEEE Design Automation Conference, pp. 212-217, 2007.

[3] M. H. Kryder, C. S. Kim, “After Hard Drives -What Comes Next?,” IEEE Transactions on Magnetics, Vol. 45, No. 10, pp. 3406-3413, 2009.

[4] P. Mangalagiri, A. Yanamandra Y. Xie, N. Vijaykrishnan, M. J. Irwin, K. Sarpatwari, O. O. A. Karim, “A Low-Power Phase Change Memory Based Hybrid Cache Architecture,” GLSVLSI, pp. 395-398, 2008.

[5] E. A. Lee, “Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing,” IEEE Transactions on Computers, Vol. 14, pp. 590-35, 1987.

[6] H. Oh, S. Ha, “Efficient code synthesis from extended dataflow graphs for multimedia applications,” Proceedings of ACM/IEEE Design Automation Conference, pp. 275-280, 2002.

[7] D. Lee, H. Oh, “A Lifetime Aware Buffer Assignment Method for Streaming Applications on DRAM/PRAM Hybrid Memory,” ACM Transactions on Embedded Computing Systems, Vol. 12, Issue 1s, No. 36, 2013.

[8] M. Kandemir, M. J. Irwin, G. Ghen, I. Kolcu, “Banked scratch-pad memory management for reducing leakage energy consumption,” Proceedings of IEEE/ACM International Conference on Comput.-aided design, pp. 120-124, 2004.

[9] J. Hu, C. J. Xue, Q. Zhuge, W. -C. Tseng, E. H. Sha, “Data allocation optimization for hybrid scratch pad memory with SRAM and nonvolatile memory,” IEEE Transactions on Very Large Scale Integration Systems, Vol. 21, No. 6, pp. 1094-1102, 2012.

[10] J. Hu, C. J. Xue, W. Tseng, Y. He, M. Qiu, E. H.-M. Sha, “Reducing Write Activities on Non-volatile Memories in Embedded CMPs via Data Migration and Recomputation,” Proceedings of ACM/IEEE Design Automation Conference, pp. 350-355, 2010.

[11] K. Lee, A. Orailoglu, “Application specific non-volatile primary memory for embedded systems,” Proceedings of the IEEE/ACM/IFIP International conference on Hardware/Software codesign and system synthesis, pp. 31-36, 2008.

[12] S. Cho, H, Lee, “Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance,” Proceedings of IEEE/ACM International Symposium on Microarchitecture, pp. 347-357, 2009.

[13] S. S. Bhattacharyya, P. K. Murthy, E. A. Lee, “Synthesis of Embedded Software from Sy

nchronous Dataflow Specifications," Journal of VLSI Signal Processing 21, Vol. 21, No. 2, pp. 151-166, 1999.

[14] H. Oh, N. Dutt, S. Ha, "Memory Optimal Single Appearance Schedule with Dynamic Loop Count for Synchronous Dataflow Graphs," Proceedings of Asia and South Pacific Design Automation Conference, pp. 497-502, 2006.

[15] W. Che, K. Chatha, "Compilation of stream programs onto scratch-pad memory based embedded multicore processors through retiming," Proceedings of ACM/IEEE Design Automation Conference, pp. 122 - 127, 2011.

[16] W. Che, K. Chatha, "Scheduling of Stream Programs onto SPM Enhanced Processors with Code Overlay," Proceedings of IEEE/ACM Symposium on Embedded Systems and Real-time Multimedia, pp. 9-18, 2011.

[17] S. Udayakumaran, A. Dominguez, R. Barua, "Dynamic allocation for scratch-pad memory using compile-time decisions," ACM Transactions on Embedded Computing Systems, Vol. 5, No. 2, pp. 472-511, 2006.

[18] M. Kandemir, J. Ramanujam, M. J. Irwin, N. Vijaykrishnan, I. Kadayif, A. Parikh, "Dynamic management of scratch-pad memory space," Proceedings of ACM/IEEE Design Automation Conference, pp. 690-695, 2001.

[19] J. Choi, H. Oh, S. Kim, S. Ha, "Executing Synchronous Dataflow Graphs on an SPM based Multi-core Architecture," Proceedings of ACM/IEEE Design Automation Conference, pp. 664-671, 2012.

[20] J. H. Lee, "PCM Main Memory for Low Power Embedded System", IEMEK J. Embed. Sys. Appl., Vol. 10, No. 6, pp. 391-397, 2015 (in Korean).

[21] Y. Joo, M. H. Kim, I. K. Han, S. S. Lim, "Cache Simulator Design for Optimizing Write Operations of Nonvolatile Memory Based Caches," IEMEK J. Embed. Sys. Appl., Vol. 11, No. 2, pp. 87-95, 2016 (in Korean).

[22] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, M. Schneider, "Potassco: The Potsdam Answer Set Solving Collection," AI Communications, Vol. 24, No. 2, pp. 105-124, 2011.

Daeyoung Lee (이 대 영)

He received the M.S. and B.S. degree in Information System from Hanyang University, Seoul, Korea, in 2011 and 2009, respectively. He is currently a Ph.D. candidate in Hanyang University. His research interests include Hardware/Software co-design methodology and Dataflow model.

Email: healor@gmail.com

Hyunok Oh (오 현 옥)

He received the M.S. and B.S. degree in Computer Engineering from Seoul National University, Korea, in 1998 and 1996, respectively. and He received the Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University, Korea, 2003. He is currently an associate professor in Hanyang University. His research interests include Cryptography, Non-volatile memory, Dataflow model and Real-time analysis.

Email: hoh@hanyang.ac.kr