https://doi.org/10.21289/KSIC.2021.24.1.69

IoT/에지 컴퓨팅에서 저전력 메모리 아키텍처의 개선 연구

A Study on Improvement of Low-power Memory Architecture in IoT/edge Computing

조두산[1*]

Doosan Cho[1*]

〈Abstract〉

The widely used low-cost design methodology for IoT devices is very popular. In such a networked device, memory is composed of flash memory, SRAM, DRAM, etc., and because it processes a large amount of data, memory design is an important factor for system performance. Therefore, each device selects optimized design factors such as function, performance and cost according to market demand. The design of a memory architecture available for low-cost IoT devices is very limited with the configuration of SRAM, flash memory, and DRAM. In order to process as much data as possible in the same space, an architecture that supports parallel processing units is usually provided. Such parallel architecture is a design method that provides high performance at low cost. However, it needs precise software techniques for instruction and data mapping on the parallel architecture. This paper proposes an instruction/data mapping method to support optimized parallel processing performance. The proposed method optimizes system performance by actively using hardware and software parallelism.

Keywords : Data mapping, architecture graph, architecture, system performance, low power

1* 정회원, 교신저자, 국립순천대학교, 교수,
E-mail: dscho@scnu.ac.kr

1* Dept. of Electrical & Electronic Engineering, Sunchon National University

1. 서 론

IoT나 에지 컴퓨팅은 적은 전력으로 고성능을 제공하도록 설계되었다. 적은 비용으로 개발되었기 때문에 값싼 저성능 프로세싱 유닛 다수를 연결하여 고성능을 제공토록 하는 매니코어 칩 형태 혹은 재구성형 프로세서 형태가 대부분을 차지한다. 상용화된 제품으로 Movidius Myriad X, Huawei Kirin SoC, Coarse grained reconfigurable array [1, 2, 3] 등이 있다. 이러한 칩들의 성능은 병렬 매니 코어 하드웨어를 충분히 이용하도록 구성된 소프트웨어에 의하여 결정된다. 병렬 하드웨어를 충분히 이용하도록 구성된 다양한 소프트웨어 기법들이 지난 수십년간 개발되어 사용되고 있는데, 대부분 슈퍼 컴퓨팅과 같은 고전력 고성능을 위한 기법들이기 때문에 IoT나 에지 컴퓨팅 시스템 소프트웨어 최적화에 적합하지 않다. 이를 해결하고자 컴파일러 분야에서 명령어 레벨 병렬화 혹은 데이터 레벨 병렬화 등 다양한 기법들이 제안 되어왔다. 명령어 레벨 기법으로 다양한 스케쥴링 기법[4, 5, 6] 들이 제안되었고, 데이터 레벨 최적화 기법으로 지역성 기반 코드 변환 기법들 [7, 8, 9]들이 제안되었다. 이러한 기법들은 다양한 형태의 병렬 시스템에 일반화된 기법들로 특정 응용에 특화된 형태로 설계된 IoT 혹은 에지 컴퓨팅 단말에서는 최적 성능을 제공하기에 적합하지 않다. 그림 1은 IoT 디바이스에서 흔히 사용될 수 있는 병렬 코어 칩 형태의 하나를 나타낸다. 저비용으로 개발되도록 기본적인 연산이 대량으로 실행 가능하도록 지원하는 프로세싱 유닛 4x4 형태로 구성되어 있으며, 저전력을 지원하기 위하여 메모리 포트가 프로세싱 유닛 1열, 2열과만 연결되어있다. 메모리 전체를 하나로 하여 데이터 버스를 전체 프로세싱 유닛과 연결하면 멀티플렉스 사이즈 또한 2^2로 네배 커지기 때문에 칩 사이즈, 개발비용, 전력 소모면에서 효율성이 떨어지게 된다. 따라서 아래와 같은 설계는 응용에 특화하여 흔하게 선택할 수 있는 선택지 중에 하나가 된다. 이러한 상용화 칩으로 GAP-8, Mr. Wolf, Intel Myriad VPU [10, 11, 1]가 있다. 이러한 설계에서 문제는 제한된 메모리 포트로 인하여 명령어 스케쥴링, 데이터 배치 등이 영향을 받게 되는데, 즉, 컴파일러가 전체 시스템 성능을 결정하게 된다. GAP-8, Myriad VPU에 아키텍처 특화 최적화 기법이 적용되어 있다.

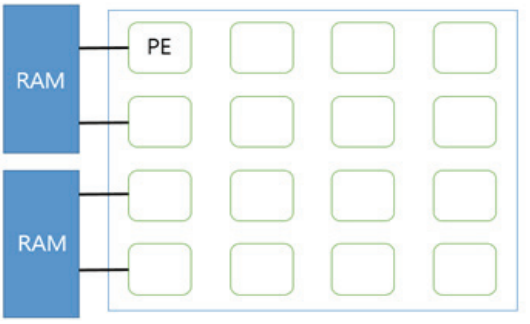

Fig. 1 an example of the architecture

2. 배경지식

2.1 IoT AND EDGE COMPUTING ARCHITECTURES

Myriad architecture [1]는 드론, 보안 및 로봇을 위한 각종 영상처리 프로세서 아키텍처로 개발되었다. 이것은 12개의 128bit VLIW 프로세서로 구성되어 있는데, 1개의 프로세서가 8개 코어를 포함하여 총 96개의 코어로 동작한다. 인공지능 영상 인식에 특화되어 개발되었는데, 특히 저전력 설계가 강조되어 있다.

Coarse grained reconfigurable architecture [3]는 64~256개 이상의 프로세싱 코어를 메쉬 구조로 연결한 성능 가속기이다. 보통 응용 프로그램 실행시간의 90%이상을 차지하는 루프 코드를 고속 수행하는데 사용된다. 데이터 스트림을 고속으로 처리하기 때문에 영상 처리와 같은 응용에 많이 채택되고 있으며, 저전력 설계를 위하여 메모리와 연결망에 응용 특화된 최적화 연구가 많이 진행되고 있다.

Nvidia의 GPGPU Fermi [12]의 경우 총 512개의 코어가 32개씩 16그룹으로 구성되어 있다. 코어의 개수가 32개 이상인 경우 각 코어간 연결망과 메모리 대역폭의 한계로 인하여 코어간 그룹 클러스터를 구성한다. 그룹으로 코어들을 연결하고 각 그룹에 온칩 메모리를 부착하여 메모리 대역폭을 공유하게 하는 것이 스트리밍 어플리케이션의 성능 및 전력 소모 측면에서 유리하다.

앞서 살펴본 아키텍처들이 IoT, Edge computing에서 흔하게 사용되는 매니코어 아키텍처들이다. 각각의 시스템에서 성능 개선을 위하여 복수의 코어들을 그룹핑하고 각 그룹별로 메모리와 코어간 연결망을 구성한다. 타겟 응용에 맞게 그룹 사이즈, 연결망 복잡도가 정해지며 성능과 전력 소모량의 측면에서 적당한 크기의 온칩메모리가 포함된다. 결과적으로 최종 설계된 시스템의 최종 성능은 코어, 연결망, 메모리를 효과적으로 사용 가능 하도록 고려하여 설계된 소프트웨어에 의하여 결정된다. 시스템 성능이 최적이 되기 위해서는 응용 코드를 분석하고, 코드에 내재된 명령어 레벨 병렬성, 데이터 레벨 병렬성을 극대화하기 위한 명령어/데이터 분할 및 할당, 스케쥴링이 이루어져야 한다. 이러한 일련의 작업은 컴파일러가 진행하기 때문에 하드웨어에 특화된 컴파일러 분석 및 최적화 기법이 요구된다.

2.2 CODE AND DATA MAPPING TECHNIQUES

이상 검토된 아키텍처들에서 코드 할당 및 데이터 배치, 스케쥴링을 위한 컴파일러 기법으로는 [13, 14]이 있다. 대표적인 명령어 배치 기법으로는 소프트웨어 파이프라이닝 [15, 16] 이 있다. 소프트웨어 파이프라이닝은 여러 프로세싱 유닛이 동시에 명령어를 실행할 수 있도록 명령어들 사이의 의존도를 분석하여 명령어를 배치하는 기술이다. 특히 병렬 형태의 하드웨어의 컴퓨팅 파워를 충분히 사용할 수 있도록 프로그램안의 명령어 병렬성을 극대화 할 수 있도록 소프트웨어 최적화를 적용한다. 소프트웨어 파이프라이닝은 상당히 일반화된 기술로 다양한 특화 플랫폼안에서 기술이 재설계 될 수 있는데, 메디컬 시스템 [17], 시스템 설계 도구 [18], 저전력 플랫폼 [19], 컴파일러 플랫폼 [20], 멀티 뱅크 메모리 플랫폼 [21], 하이브리드 메모리 플랫폼 [22] 등의 연구에서 다양한 최적화를 시도하였다.

메디컬 시스템 [17]의 경우 명령어 재배치를 고려할 때 실시간 제약조건을 고려해야 한다. 하드리얼 타임 조건을 맞추기 위해 명령어 배치를 예측 가능한 범위안에서 진행한다. 시스템 설계 도구 [18]는 프로그램의 데이터 플로우 분석과 제어 흐름 분석을 진행하는데 이러한 분석 정보는 기본적인 컴파일러 최적화 기법에 사용되지만 특히 명령어 및 데이터의 재배치에 필수적으로 그 정확도 확보가 매우 중요하다. 대부분의 컴퓨팅 디바이스에서 저전력 기술 [19]이 매우 중요하게 다루어지고 있다. 현재 많은 제품이 배터리 구동 형태로 개발되었기 때문에 제품의 성공을 결정하는 매우 중요한 요소이기 때문이다. 명령어 배치의 방법론에 따라 배터리 사용량의 상당부분을 최적화하는 것이 가능하다. 현재 개발되는 신규 플랫폼의 반

이상이 LLVM이라는 컴파일러 플랫폼 [20]을 사용하는데, 이것은 기본적으로 사용되는 명령어 및 데이터 배치 기술을 포함하고 있다. 개발자의 필요에 따라 적합한 최적화 기법을 설계 및 개발할 수 있도록 지원하고 있다. 저비용 고성능 제품을 개발할 때 다양한 설계 방법론을 채택할 수 있는데, 그중에 대표적인 메모리 개발 방법론이 멀티 뱅크 메모리 플랫폼 [21]이다. 멀티 뱅크 메모리는 저렴한 비용으로 복수개의 메모리 접근을 지원하기 때문에 고성능을 보장할 수 있게 된다. 하이브리드 메모리 메모리 플랫폼 [22]은 반도체 집적도가 향상될수록 누설 전류가 지수증가하여 동작 전력이 증가하게 됨으로 저전력 설계에 적합하지 않아 개발된 기술이다. 누설전류 특성이 거의 0에 가까운 메모리 소자와 그 단점을 커버하기 위한 기존 소자를 혼합 사용하여 고집적 공정에서 누설 전류 특성을 크게 개선할 수 있게 된다.

3. 제안하는 기법

본 연구에서는 다양한 저전력 IoT 및 에지 컴퓨팅 플랫폼에서 최적의 코드 및 데이터 배치를 결정할 수 있는 기술을 제안하고 있다. 기존의 다양한 기법이 있었으나 다양한 플랫폼을 위한 최적의 솔루션을 제시할 수 있는 기술은 우리가 아는 선에서 없었기 때문에 본 기술의 기여가 중요하다.

이 연구는 아키텍처 그래프로 저전력 IoT 및 에지 컴퓨팅 플랫폼의 특성을 정형화하고, 이에 맞는 코드 및 데이터 배치를 결정하기 위해서 로드/스토어 그래프를 이용한다. 로드/스토어 그래프를 아키텍처 그래프에 매핑하여 최적의 데이터 배치를 결정할 수 있게 된다.

먼저 아키텍처 그래프는 그림 2와 같이 아키텍처가 있을 때 이를 요약 표현하는 데이터구조를 말한다. 그림 2의 상단에 일반적인 병렬 형태의 메시 연결된 프로세싱 유닛과 더블 버퍼링이 제공되는 메모리를 포함한 아키텍처를 나타내었다. 그림 2의 하단에 해당 아키텍처에 해당하는 아키텍처 그래프를 나타내었다. 이 아키텍처 그래프는 프로세싱 유닛, 메모리, 그리고 이러한 컴포넌트들의 연결 네트워크를 나타낸다. 아키텍처 그래프의 정의는 아래와 같다.

DEFINITION 1. ARCHITECTURE GRAPH
아키텍처 그래프는 노드와 에지로 구성된 그래프이다. 노드는 두가지 형태로 프로세싱 유닛과 메모리를 나타낸다. 에지는 노드들의 연결을 나타내는 비방향성 에지로 정의한다.

본 연구에서는 위에 정의된 아키텍처 그래프를 상대로 로드/스토어 그래프를 매핑시킴으로서 최적의 데이터 배치를 결정한다. 로드/스토어 명령어를 실행하는 프로세싱 유닛으로 데이터를 복사 전송하기 때문에 이러한 로드/스토어 명령어의 배치에 따라 데이터의 배치가 따라서 결정되는 결과를 얻을 수 있다. 정의된 로드/스토어 그래프는 아래와 같다.

DEFINITION 2. LOAD/STORE GRAPH
이 그래프는 에지와 노드의 집합으로 구성된다. 집합 N의 각 노드는 로드/스토어 명령을 나타낸다. 에지 집합의 각 에지는 노드들 사이의 종속성 관계를 나타낸다. 에지의 가중치는 합산된 노드의 수를 나타낸다.

그래프 구성 과정은 다음과 같다. 먼저 데이터 종속성 그래프가 구성된다. 종속성 그래프는 최적화 컴파일러에서 사용되는 데이터 흐름 분석의 기본

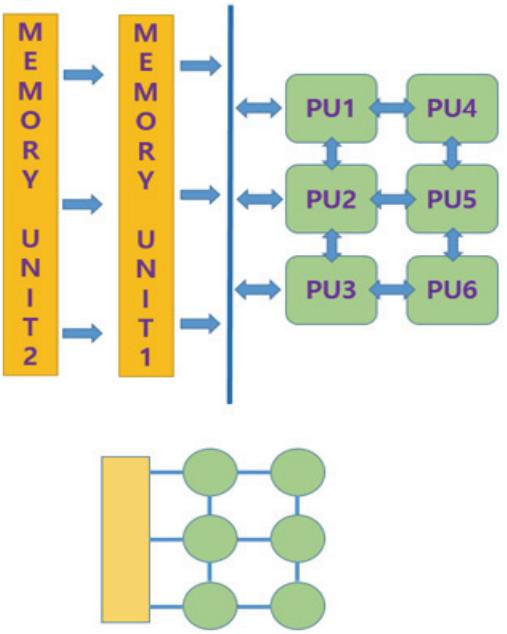

Fig. 2 The target architecture and its graph

데이터 구조로서, 본 연구에서는 기존 알고리즘을 사용한다 [23]. 그래프 구성 대상이되는 프로그램 코드를 입력으로 구현된 데이터 의존성 그래프는 우리가 제안한 로드/스토어 의존성 그래프를 구현하기 위한 출발점이다. 데이터 의존성 그래프에서 시작점은 그래프의 모든 산술 노드를 로드-스토어 노드에 병합하는 것이다. 로드 명령에 의존하는 산술 노드는 로드에 해당하는 노드로 병합됩니다. 이때 다른 노드에 연결된 에지는 해당 로드 노드에 연결됩니다. 이 작업을 반복하면 종속성 그래프가 로드/스토어 명령어로만 구성된다. 그림 3은 종속성 그래프 작성 프로세스를 나타낸다.

그림 3의 가장 왼쪽의 그래프가 일반적으로 컴파일러에서 생성된 데이터 종속성 그래프이다. 처음 두 개의 로드 명령어는 변수 A,B를 만들고 순서대로 ADD, MULTIPLY, DIVIDE 명령어를 실행하여 최종 계산 결과를 스토어 명령어를 실행하여 저장한다. 그림 3의 두 번째 그래프에서 ADD 노

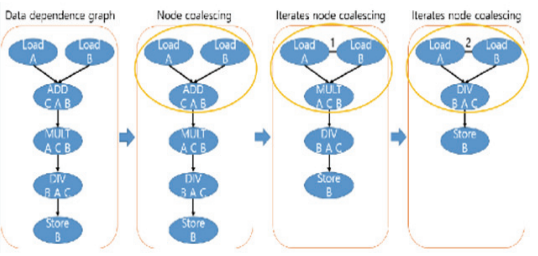

Fig. 3 The building process of the load/store dependence graph

드는 로드 A와 로드 B에 의존하여 실행되어 있다. 따라서 두 개의 로드 명령어와 ADD 명령어는 병합할 수 있다. 병합을 진행할 때 두 노드들 사이에 Add를 병합하여 에지를 생성한다. 가중치 1은 통합된 노드가 두 개의 로드에 의해 병합되었음을 나타낸다. 이 프로세스를 반복하여 MULTIPLY 및 DIVIDE 명령어의 노드를 병합한다. 결국 가중치 3은 두 개의 로드 명령어 노드 사이에 생성됩니다. 이는 로드가 3개의 노드를 병합했기 때문에 3개의 작업 전에 동시에 실행되어야 함을 나타낸다.

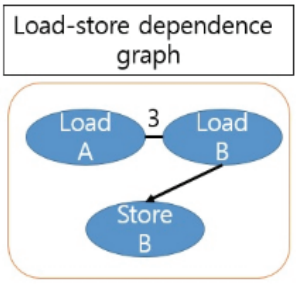

Fig. 4 An example of load/store dependence graph

그림 4는 최종적으로 생성된 로드/스토어 종속성 그래프를 나타낸다. 스토어 노드는 변수 B 만 저장하므로 에지는 로드 B 노드에만 연결된다. 제안된 기술은 이 로드/스토어 그래프를 사용하여

IoT 및 에지 컴퓨팅 시스템에서 명려어/데이터 배치를 보다 효과적으로 결정할 수 있다.

4. 결과 및 고찰

로드/스토어 종속성 그래프를 기반으로 본 논문에서 제안된 기술은 여러 프로세싱 유닛에 명령어와 데이터를 효과적으로 배치하도록 프로그램 코드를 최적화한다. 제안된 기술의 목표는 시스템에서 실행되는 프로그램의 실행 시간 및 전력 사용량을 개선하는 것이다. 이를 위해 본 연구에서는 이러한 명령어/데이터 매핑 문제를 간단한 그래프 매핑문제로 변형하여 정의한다.

PROBLEM :
MINIMIZE_PERFORMANCE(MAPPING(ARCHITECTURE_GRAPH, LOAD_STORE_GRAPH))

제안된 기법은 전체 (exhaustive) 검색 기법을 이용하여 로드/스토어 의존성 그래프의 각 노드를 아키텍처 그래프에 할당하고 그 결과를 실행시간 변수로 측정한 이후 다른 매핑 결과와 비교하여 최적의 매핑을 찾아내는 전체 검색을 진행합니다. 노드의 수가 증가할수록 검색 공간이 지수 증가하는 문제가 있으나 본 연구에서는 우선 전체 검색을 사용하고 향후 연구에서 이를 개선하도록 할 계획이다.

그림 5는 여기서 정의한 문제에 대한 솔루션을 찾기 위하여 제안한 알고리즘을 나타낸다. 제안된 알고리즘은 로드/스토어 그래프와 아키텍처 그래프를 입력으로 실행된다. 기본적으로 사용된 매핑 알고리즘은 전체 검색공간을 탐색하는 전체 탐색(exhaustive search)를 이용하고 있다. 로드/스토어 그래프의 각 노드를 아키텍처 그래프에 하나씩 할당하고 모든 로드/스토어 그래프를 할당하였을 때 그 매핑 결과를 바탕으로 성능을 측정하여 저장한다. 제안된 알고리즘은 또 다른 매핑 결과의 성능 측정 결과와 이전 결과를 비교하여 더 나은 매핑 결과를 저장하는 과정을 더 이상 다른 매핑 결과가 없을 때 까지 진행하여 최적의 성능을 제공하는 매핑을 최종 솔루션으로 제공한다. 이때 이전 성능 측정 결과를 PREV_BEST에 저장하고 현재의 측정 결과를 BEST_SOLUTION에 저장하여 비교한다. 매핑 결과는 Result에 저장하여 사용한다. 전체 검색 공간 탐색 방식은 그래프의 노드의 수가 N 증가할 때 검색해야 할 공간의 크기가 N의 지수 크기 만큼 증가하기 때문에 그래프의 사이즈에 따라 검색시간이 오래 걸릴 수도 있으나 이 부분은 향후 연구로 남겨둔다.

```
MAPPING ALGORITHM
LSG = Load_Store_Dependence_Graph
AG = Architecture_Graph
PREV_BEST = 0
WHILE(1)
        if(Result !=Empty){
         if(N!=Empty)
                N= Get_Node(LSG)
                Result=Mapping(N, AG)

        BEST_SOLUTION =
        Estimate_Performance(Result)

        IF (BEST_SOLUTION >= PREV_BEST)
                PREV_BEST = BEST_SOLUTION
        }ELSE

Return BEST_SOLUTION
```

Fig. 5 The proposed mapping technique

그림 4에 나타난 로드/스토어 그래프를 그림 2에 나타낸 아키텍처에 매핑하는 경우 그 결과는 그림 6의 오른쪽 결과와 같이 얻을 수 있다. 해당 아키텍처는 동시에 2개의 읽기 그리고 1개의 쓰

기 각각의 병렬 처리를 지원하는데, 로드 A와 로드 B는 동시에 수행되어야 성능 측면에서 최적이며, 스토어 명령어는 각종 연산 명령어를 3개를 수행한 이후 실행 가능하기 때문에 로드와 다른 슬롯에서 실행되는 것이 유리하다. 따라서 두 개의 로드 명령어는 같은 라인에 한 개의 스토어는 다른 라인에 할당되어 최적의 성능을 얻을 수 있게 된다. 그림 6 왼쪽 매핑 결과와 같이 반대의 경우를 예를 들어 보면, 스토어 B 명령어가 로드 A와 직접 연결된 노드에 매핑될 수 있다. 이때는 LOAD A가 실행되고 동시에 스토어 B를 시행하고자 할 때 B의 위치가 LOAD B와 같이 한칸 아래쪽이기 때문에 스토어 처리를 위해 데이터가 아래 유닛 한칸, 왼쪽 유닛 한칸 이렇게 2칸의 이동이 발생하여 성능 저하를 초래한다. 따라서 최적의 배치는 스토어 B가 LOAD B와 같은 행 다른 열에 위치하는 것이다. 따라서 제안된 기법이 최적의 명령어 및 데이터 배치를 제공할 수 있게 된다.

제안된 기법을 그림 7의 GSM (Global System for Mbile) 코드의 메인 루프에 매뉴얼하게 적용해 보자. 1, 2, 4, 6번 명령어가 로드이고, 9가 스토어 명령어 그리고 나머지가 일반 연산 명령어이다. 일반적인 매핑은 번호 순서대로 명령어를 프로세싱 유닛에 할당하기 때문에, 그림 1의 4x4 병렬처리 칩에서는 순서대로 1행에 1-4번 명령어, 2행에 5-8번 명령어, 3행에 9번 명령어가 할당된다.

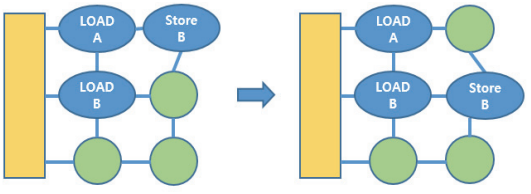

Fig. 6 The mapping result

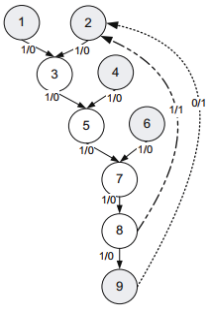

Fig. 7 GSM example

단순하게 각 명령어가 1싸이클 소비한다면 이러한 매핑에서 총 10싸이클이 소비되지만, 제안된 기법으로는 5싸이클로 실행을 완료할 수 있다. 따라서 해당 루프 코드에서 성능이 2배 개선될 수 있고, 루프코드가 전체 실행시간의 90%를 차지함을 감안한다면 전체 시스템 성능이 크게 개선될 수 있다.

이러한 기법을 드론의 메인 제어부 [24], 로봇의 메인 제어부 [25]에 적용한다면 배터리 효율 개선 뿐만 아니라 리얼타임 제약조건 개선 및 전체 시스템의 성능 개선을 달성할 수 있을 것으로 기대한다.

5. 결 론

본 논문에서는 병렬 프로세싱 유닛을 가진 다양한 프로세서 아키텍처에 대한 명령어/데이터 위치 최적화 문제를 해결하는 기법을 제안한다. 명령어/데이터 할당 문제를 그래프 노드 매핑 문제로 변환하여 기법을 제안하였다. 제안된 방법의 장점은 리소스 할당 문젱 해당하는 NP 문제를 매우 간단한 그래프 노드 매핑 문제로 정의하여 다양한 아키텍처 플랫폼에서 효율적인 결과를 얻을 수 있다는 것이다.

Acknowledgement

This work was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF - 2018R1D1A1 B07050054).

참고문헌

[1] M. Ionica and D. Gregg, : The Movidius Myriad Architecture's Potential for Scientific Computing. IEEE Micro. 35. 1-1, (2015)

[2] Huawei GPU Turbo Technology Research. [Online] Available: https://www.researchgate.net/publication/336890831_Huawei_GPU_Turbo_Technology_Research. (2019)

[3] A Coarse Grain Reconfigurable Array (CGRA) for Statically Scheduled Data Flow Computing. [Online]. Available: https://wavecomp.ai/wp-content/uploads/2018/12/WP_CGRA.pdf. (2018)

[4] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S. Amarasinghe : Space-time scheduling of instruction-level parallelism on a raw machine. SIGOPS Oper. Syst. Rev. 32, 5, 46–57 (1998)

[5] J. L. Lo and S. J. Eggers : Improving balanced scheduling with compiler optimizations that increase instruction-level parallelism. SIGPLAN Not. 30, 6, 151–162, (1995)

[6] N. P. Jouppi and D. W. Wall : Available instruction-level parallelism for superscalar and superpipelined machines. SIGARCH Comput. Archit. News 17, 2, 272–282, (1989)

[7] I. Sung, J. A. Stratton and W. W. Hwu : Data layout transformation exploiting memory-level parallelism in structured grid many-core applications. International Conference on Parallel Architectures and Compilation Techniques. 513-522. (2010)

[8] M. E. Wolf and M. S. Lam : A data locality optimizing algorithm. In Proceedings of the ACM SIGPLAN conference on Programming language design and implementation, 30–44. (1991)

[9] K. S. McKinley, S. Carr, and C.W. Tseng : Improving data locality with loop transformations. ACM Trans. Program. Lang. Syst. 18, 4, 424–453, (1996)

[10] E. Flamand, V. Bonnot, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, L. Benini : GAP-8: A RISC-V SoC for AI at the Edge of the IoT. International Conference on Application-specific Systems, Architectures and Processors, pp. 1-4. (2018)

[11] A. Pullini, D. Rossi, I. Loi, G. Tagliavini and L. Benini : Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing. IEEE Journal of Solid-State Circuits, vol. 54, no. 7, pp. 1970-1981, (2019)

[12] Whitepaper, NVIDIA's Next Generation CUDA Compute Architecture: Fermi. [Online] https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[13] D. Cho, S. Pasricha, I. Issenin, N. D. Dutt, M. Ahn and Y. Paek : Adaptive Scratch Pad Memory Management for Dynamic Behavior of Multimedia Applications. in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 28, no. 4, pp. 554-567, April (2009)

[14] Y. Kim, J. Lee, A. Shrivastava, J. W. Yoon, D. Cho and Y. Paek : High Throughput Data Mapping for Coarse-Grained Reconfigurable Architectures. in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 11, pp. 1599-1609, Nov. (2011)

[15] B. Rau : Iterative modulo scheduling. HP Laboratories Technical Report, HPL94115, (1995)

[16] D. Lavery and W. Hwu : Unrolling-Based Optimizations for Modulo Scheduling. Proceedings of the 28th annual international symposium on

Microarchitecture, pp.327-337, (1995)
[17] J. Cho, J. Lee, D. Cho : Efficient memory design for medical database. Basic & Clinical Pharmacology & Toxicology, 125, pp. 198, (2019)
[18] J Cho, D Cho : Development of a Prototyping Tool for New Memory Subsystem. International Journal of Internet, Broadcasting and Communication, 11 (1), pp. 69-74, (2019)
[19] D Cho : Technology of the next generation low power memory system. International Journal of Internet, Broadcasting and Communication, 10 (4), pp. 6-11, (2018)
[20] J Cho, D. Cho, Y Kim : Study on LLVM application in Parallel Computing System. The Journal of the Convergence on Culture Technology (JCCT), 5 (1), pp. 395-399, (2019)
[21] J Cho, JM Youn, D Cho : An Automatic Array Distribution Technique for Multi-Bank Memory of High Performance IoT Systems. World, 3 (1), pp. 15-20, (2019)
[22] J Youn, D Cho : A spill data aware memory assignment technique for improving power consumption of multimedia memory systems. Multimedia Tools and Applications, 78 (5), pp. 5463-5478, (2019)
[23] Cho D., Ravi A., Uh GR., Paek Y. : Instruction Re-selection for Iterative Modulo Scheduling on High Performance Multi-issue DSPs. (2006)
[24] Cho, D. : A Study on software performance acceleration for improving real time constraint of a VLIW type Drone FCC. Journal of the Korean Society of Industry Convergence, 20(1), 1–7. (2017)
[25] Hyun-Seok Sim, Ho-Young Bae, Du-Beum Kim, Sung-Hyun Han. : A Study on Flexible Control and Design of Robot Hand Fingers with Eight Axes for Smart Factory. Journal of the Korean Society of Industry Convergence, 21(4), 183-189. (2018)

(접수: 2021.01.19. 수정: 2021.02.10. 게재확정: 2021.02.15.)