논문 2019-14-13

연결기반 명령어 실행을 이용한 재구성 가능한 IoT를 위한 온칩 플래쉬 메모리의 클라우드화

(Cloudification of On-Chip Flash Memory for Reconfigurable IoTs using Connected-Instruction Execution)

이 동 규, 조 정 훈, 박 대 진*
(Dongkyu Lee, Jeonghun Cho, Daejin Park)

Abstract : The IoT-driven large-scaled systems consist of connected things with on-chip executable embedded software. These light-weighted embedded things have limited hardware space, especially small size of on-chip flash memory. In addition, on-chip embedded software in flash memory is not easy to update in runtime to equip with latest services in IoT-driven applications. It is becoming important to develop light-weighted IoT devices with various software in the limited on-chip flash memory. The remote instruction execution in cloud via IoT connectivity enables to provide high performance software execution with unlimited software instruction in cloud and low-power streaming of instruction execution in IoT edge devices. In this paper, we propose a Cloud-IoT asymmetric structure for providing high performance instruction execution in cloud, still low power code executable thing in light-weighted IoT edge environment using remote instruction execution. We propose a simulated approach to determine efficient partitioning of software runtime in cloud and IoT edge. We evaluated the instruction cloudification using remote instruction by determining the execution time by the proposed structure. The cloud-connected instruction set simulator is newly introduced to emulate the behavior of the processor. Experimental results of the cloud-IoT connected software execution using remote instruction showed the feasibility of cloudification of on-chip code flash memory. The simulation environment for cloud-connected code execution successfully emulates architectural operations of on-chip flash memory in cloud so that the various software services in IoT can be accelerated and performed in low-power by cloudification of remote instruction execution. The execution time of the program is reduced by 50% and the memory space is reduced by 24% when the cloud-connected code execution is used.

Keywords : High performance, Low power, Remote instruction, IoT, Cloudification, Fog computing

*Corresponding Author (boltanut@knu.ac.kr)
Received: Feb. 24, 2019, Revised: Mar. 24, 2019, Accepted: Apr. 01, 2019.
D. Lee, J. Cho, D. Park: Kyungpook National University.
※ 본 논문은 교육부의 재원으로 한국연구재단의 기초과학연구 프로그램의 지원을 받아 수행된 연구결과임(No.2014R1A6A3A04059410, 2016R1D1A1B03934343).
※ 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2019R1A2C2005099).

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2019.14.2.103

Ⅰ. 서 론

IoT 기술들은 우리의 생활 속에 점점 녹아들고 있다. IoT 기술은 주변 환경을 인식하고 정보를 전달해 주는 센서와 프로그램을 수행하고 정보를 처리해주는 장치 등이 네트워크를 통해 연결되어 서비스를 사용자에게 제공한다 [1]. 이 IoT 장치들은 주변의 데이터를 수집하는 센서와 데이터를 처리하고 동작을 결정하는 임베디드 시스템, 결정된 동작을 수행하는 액츄에이터 등으로 이루어져 있다. 이러한 구조 속에서 임베디드 시스템은 제한된 크기

의 메모리, 제한된 하드웨어 공간 그리고 전력 공급의 불안정성 등의 제약이 있다. 임베디드 시스템은 가전이나 시설물 등 공간의 제약이 없고, 전력의 공급이 용이한 장소나 숲 속이나 바다 속 등 전력 공급이 어렵고 공간의 제약이 많은 곳에 사용될 수 있다. 대부분의 임베디드 장치들은 하드웨어의 크기가 제한되어 있다. 이러한 제약조건 속에서 낮은 성능 임베디드 장치들과 낮은 전달 속도의 연결로 만들어진 IoT가 충분한 성능을 낼 수 있다는 연구가 있다 [2, 3]. 본 논문은 IoT에서 센서의 입력을 받고 정보를 처리하는 엣지(Edge) 단의 임베디드 시스템을 대상으로 고성능 저전력을 달성할 수 있는 프로세서 구조를 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에 관련된 연구에 대해 소개하고, 3장에서는 제안하는 구조에 대해 소개할 것이다. 4장에는 구현과 실험을 통해 제안하는 구조에 대한 결과를 표현하고, 5장에서는 결론을 맺을 것이다.

Ⅱ. 관련 연구

하드웨어의 크기가 제한되어 있다는 것은 코드가 적재되는 메모리의 크기가 줄어든다는 것을 의미하며 그로 인해 적재할 수 있는 코드의 크기가 제한되어 있다는 것을 의미한다. 다르게 표현하자면 하나의 장치가 제공할 수 있는 서비스의 개수가 제한되어 있다는 것을 의미한다. 이러한 메모리 크기 제약을 해결하기 위해 다양한 연구가 진행되어왔다. 임베디드 장치에는 코드를 저장하는 코드 메모리와 코드를 실행하기 전 임시로 저장하는 캐시 메모리가 있다. 이 구조에서 착안해 캐시 메모리 대신 스크래치 패드 메모리를 사용해 임베디드 시스템의 캐시 메모리가 차지하는 공간을 줄이고 전력 소모를 적게 만드는 방향으로 연구가 진행되었다 [4, 5]. 다른 방면으로는 실행 코드를 컴파일러로 최적화하여 코드의 크기를 줄이거나 아키텍처를 사용 용도에 맞게 수정하여 코드의 효율성을 높이고 최적화하는 연구가 진행되었다 [6].

임베디드 시스템은 제한된 공간에 제한된 전력을 사용하는 프로세서를 만들어야 한다. 그러므로 연산 능력이 일반적인 프로세서에 비해 떨어진다. 임베디드의 연산 능력을 높이고 전력 소모를 줄이는 방법 중의 하나로 하드웨어 가속기를 이용하는 방법이 있다. 여러 분야에서 연산 속도를 높이면서 전력 소모를 최소화하기 위해 임베디드 가속기를 사용하는 연구도 진행되었다 [7, 8].

코드 메모리의 크기를 줄이기 위해 코드의 크기를 줄이면 IoT 장치가 제공할 수 있는 서비스의 수가 줄어든다. 스크래치 패드 메모리를 사용해 캐시 메모리를 없애는 방법은 프로세서의 면적을 줄일 수 있지만, 상대적으로 캐시 메모리만을 제거할 수 있으므로 면적을 줄이는데 한계가 있다. 적재되는 코드를 최적화하고 아키텍처를 수정하여 코드의 크기를 줄이는 방법은 코드를 유지 보수할 때마다 최적화를 다시 해 주어야 하므로 서비스 업데이트가 어렵다 [9]. 임베디드 장치의 성능을 높이기 위해 가속기를 추가하게 되면 하드웨어의 크기가 커지고 그로 인해 전력 소모도 증가한다.

제공하는 서비스의 수를 유지하고, 코드의 크기를 줄이기 위한 방법으로 클라우드에 가상 메모리를 만들어 필요한 코드를 가져오는 연구도 진행되었다 [10]. 이 방법은 클라우드를 임베디드 칩의 메모리처럼 사용하는 것으로 클라우드에 존재하는 코드를 수행해야 하는 부분만 가져와 수행하므로 서비스를 수행할 때 필요한 메모리의 크기가 작다. 서비스의 업데이트가 필요한 상황에는 클라우드의 코드만 수정하면 되므로 서비스를 제공하는 임베디드 장치에 접근하지 않아도 된다. 그러나 위의 논문에서 제안한 방법은 서비스 코드가 임베디드 장치에서 실행되므로 코드를 가져오는 통신 시간에 의해 서비스 실행 시간이 더 늘어나게 된다.

이전에 진행되었던 연구는 임베디드 장치 내부에서의 최적화만을 다루었다. 본 논문에서는 이전에 진행되었던 연구와는 달리 임베디드 시스템에 원격 코드 실행 시스템을 적용하여 코드 메모리 사용량을 줄이고, 프로그램을 서버에서 실행함으로써 실행 시간을 줄이는 고성능 저전력 프로세서 구조를 제안한다. 또한, 이전 연구의 원격 코드 실행 시스템을 반대로 적용해 임베디드 장치의 코드를 서버에서 실행할 수 있도록 한다. 임베디드 장치의 코드를 적재할 때에는 실행되는 프로그램의 코드를 분석하여 코드 내에서 가속화할 부분을 파악하고 인라인 어셈블리 등을 통해 원격 명령어를 포함하는 코드로 바꿔 준다. 서버에는 원격 실행 코드의 요청을 받아 실행하는 가속기를 두어 프로그램 수행 시간을 줄인다.

Ⅲ. 제안하는 구조

본 논문에서는 원격 명령어 실행 시스템을 이용해 임베디드 시스템의 한계인 메모리와 하드웨어 크기 제한으로 인한 서비스의 제약을 해결하고, 임

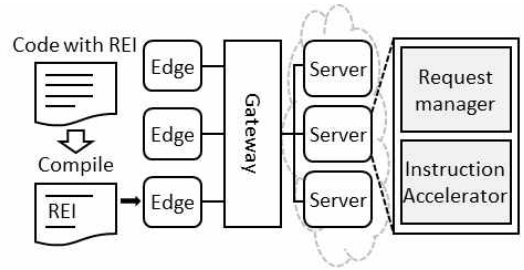

그림 1. 원격 명령어 실행 구조의 개념
Fig. 1 Concept of remote executable instruction structure

베디드 장치에서 프로그램 수행 시간 단축을 목표로 한다. 메모리와 하드웨어 크기 제약은 원격 실행 시스템을 사용해 해결하였다. 원격 실행 시스템은 서버가 실제 실행하는 코드를 가지게 되고, 클라이언트인 IoT 엣지는 원격 실행 코드가 적재된다. 원격 실행 코드는 실제 실행되는 코드에 비해 작은 코드 크기를 가지므로 하나의 프로그램에 필요한 메모리의 공간이 줄어든다. 제안하는 구조에서는 원격 실행 시스템에 더해 서버에 가속기를 두고 가속기를 구동하는 원격 명령어를 이용해 임베디드 시스템을 가속화하였다.

원격 명령어 실행 시스템의 전체 구조는 그림 1과 같다. 임베디드 시스템인 엣지에서 제공하는 서비스는 원격 실행 명령어(Remote Executable Instruction)를 포함하는 코드로 만들어진다. 만들어진 코드는 엣지의 아키텍처에 맞게 컴파일 된 후 메모리에 적재된다. 엣지는 적재된 코드를 실행하다가 원격 실행 명령어를 만나면 게이트웨이를 거쳐 서버에 원격 실행을 요청한다. 서버는 게이트웨이에서 요청을 받아 가속기를 통해 명령어를 실행한 후 결과를 반환한다.

1. 원격 실행 명령어 구성

원격 실행 명령어는 명령어 아키텍처에서 하드웨어 가속기를 위한 명령어를 추가하는 방법과 동일한 과정을 거친다. 먼저, 코드 내부에서 하드웨어 공간을 많이 차지하는 명령어 또는 빠른 연산을 필요로 하는 코드를 선택한다. 그 후 명령어 가속을 위한 가속기를 서버에 설계한다. 원격 실행 명령어는 일반 명령어를 대체하여 사용되며 실행되었을 때 서버의 가속기를 호출하게 된다.

본 논문에서는 행렬 곱셈 연산을 예제로 환경을 구성하였다. 행렬 곱셈 연산은 $O(n^3)$의 연산 복잡

Algorithm Matrix multiplication

```
Set matrix A_{n×n}, B_{n×n}
Mat_mul() begin
for i = 0, 1, ..., n-1
    for j = 0, 1, ..., n-1
        for k = 0, 1, ..., n-1
            C[i][j] += A[i][k] * B[k][j]
        end
    end
end
end
```

그림 2. 원격 실행 명령어를 위한 행렬 곱셈연산 예제
Fig. 2 Example of matrix multiplication for remote execution instruction

Algorithm Matrix remote instruction

```
Set matrix A_{n×n}, B_{n×n}
Create_client
R_mat_mul(A_{n×n}, B_{n×n})
Destroy_client
```

그림 3. 원격 명령어를 적용한 엣지 측 행렬 곱셈 연산
Fig. 3 Matrix multiplication with remote instruction in edge device

도를 가지고 있어 데이터 전송에 비해 많은 연산을 하는 간단한 예이다. 그림 2는 행렬 곱셈을 위한 간단한 예제 코드이다. 코드 내부에서 행렬의 곱 연산은 그림 2의 알고리즘으로 동작한다. 우리는 그림 2의 코드에서 n개의 행을 가지는 정방행렬 두 개를 곱하는 함수를 가속하기 위해 행렬 곱셈 함수를 원격 명령어로 만들었다.

그림 3은 원격 명령어를 적용한 행렬 곱셈 연산을 나타낸다. 원격 명령어를 실행하기 전 엣지는 서버와의 연결을 위한 클라이언트를 만든다. 클라이언트가 연결을 생성하면 원격 명령어가 요청되어 필요한 데이터를 전송하고, 서버에서는 데이터를 받아 처리한 후 결과를 반환한다. 결과를 받은 엣지는 클라이언트를 종료해 연결을 끊는다. 서버와 엣지 간의 연결은 원격 절차 호출 (Remote procedure call, RPC)로 이루어져 있으며, 클라이언트가 통신 연결과 데이터 통신을 프로그램의 뒷단에서 수행해 준다. 그림 2와 그림 3을 비교해 보았을 때, 원격 명령어는 일반 명령어를 사용하는 플랫폼에서 많은 수정을 거치지 않고 사용할 수 있다.

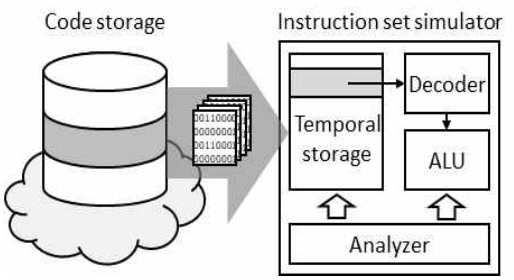

그림 4. 명령어 시뮬레이터의 개념
Fig. 4 Concept of the instruction set simulator

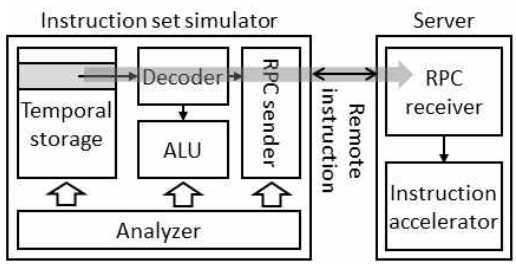

그림 5. 원격 명령어의 실행 흐름
Fig. 5 Execution flow of the remote instruction

2. 프로세서 동작 시뮬레이터

다음으로 제안한 구조를 시뮬레이션 할 수 있는 플랫폼을 구성하였다. 임베디드 시스템인 엣지는 명령어 시뮬레이터 (Instruction Set Simulator, ISS)로 구성하였다. ISS는 바이너리 형태의 코드를 받아 명령어를 해석하고 실행함으로써 작은 프로세서를 모방하는 프로그램이다. ISS의 개념은 그림 4처럼 나타낼 수 있다. 클라우드나 HDD와 같은 코드 저장소에는 많은 양의 코드가 적재되어 있다. ISS는 코드 저장소에서 실행해야 할 코드를 받아온다. 받아온 코드는 ISS에서 캐시의 역할을 하는 임시 저장소에 저장되고, 저장된 코드는 프로그램이 수행될 때의 PC값 변화에 따라 기능이 해석되고, 수행된다. 우리는 ISS에서 코드가 실행될 때 명령어가 수행되는 시간과 전력소모를 알기 위해 분석기를 추가하였다. 분석기는 각 명령어가 실행될 때 소모하는 전력과 수행하는데 필요한 클럭 수를 가지고 있다. ISS는 코드를 받아 실행하며 프로그램이 수행되는데 필요한 시간과 소모 전력을 시뮬레이션 할 수 있다.

3. 원격 명령어 실행 과정

그림 5는 하나의 서버와 하나의 엣지가 통신을 하는 모습을 나타낸 것이다. 엣지는 ISS로 이루어져 있으며 프로그램을 실행하여 실행 시간과 소모 전력을 연산해 예측할 수 있다. 프로그램을 구성하는 코드는 엣지에서 실행하는 일반 명령어 코드와 원격으로 실행되는 원격 실행 명령어 코드가 있다. 원격 실행 명령어를 포함하는 코드는 타겟이 되는 임베디드 시스템에 맞게 컴파일되어 엣지의 코드 메모리에 적재된다. ISS는 시간적 지역성과 공간적 지역성으로 대표되는 캐시 알고리즘에 따라 코드 메모리에 적재된 코드를 임시 저장소로 옮긴다.

ISS에서 임시 저장소에 저장된 코드는 해독 과정에서 일반 명령어와 원격 명령어로 분류된다. 일반 명령어 코드는 해독을 한 후 엣지의 연산장치에서 바로 연산이 된다. 원격 명령어가 호출될 경우 ISS는 서버와의 연결을 지원하는 RPC sender를 호출하게 된다. RPC sender는 서버와 ISS 사이의 연결을 만들고 명령어를 전달한다. 서버에는 ISS와의 연결을 만들고 요청을 받는 리시버가 있다. 요청을 받은 서버는 명령어 가속기에서 연산을 수행한 후 결과를 ISS에 반환한다.

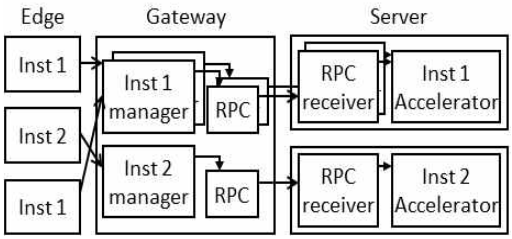

그림 6. 원격 명령어 실행 과정
Fig. 6 Remote instruction execution process

IoT 환경에서는 서버 하나에 여러 개의 클라이언트가 존재하게 된다. 그림 6은 다수의 엣지가 서로 다른 원격 명령어를 요청하고 서버에서 실행되는 과정을 나타낸 그림이다. 다수의 엣지와 다수의 서버 간의 통신을 중개하기 위해 게이트웨이를 사용했다. 코드의 원격 명령어가 실행되면 엣지는 게이트웨이에 원격 명령어의 정보를 전달한다. 전달된 명령어의 정보를 통해 게이트웨이는 명령어를 분류하고 원격 명령어를 처리할 수 있는 서버를 찾는다. 게이트웨이는 서버와 엣지 간 통신을 만들어 데이터를 주고받을 수 있도록 해 준다.

그림 7은 여러 개의 엣지에서 서버에 원격 명령어를 요청할 경우를 고려한 구조이다. 원격 명령어가 실행되면 ISS는 게이트웨이에 요청을 전달한다. 게이트웨이는 전달 받은 요청을 받아 어떤 명령어인지 분류한다. Gateway는 서버가 가지고 있는 가

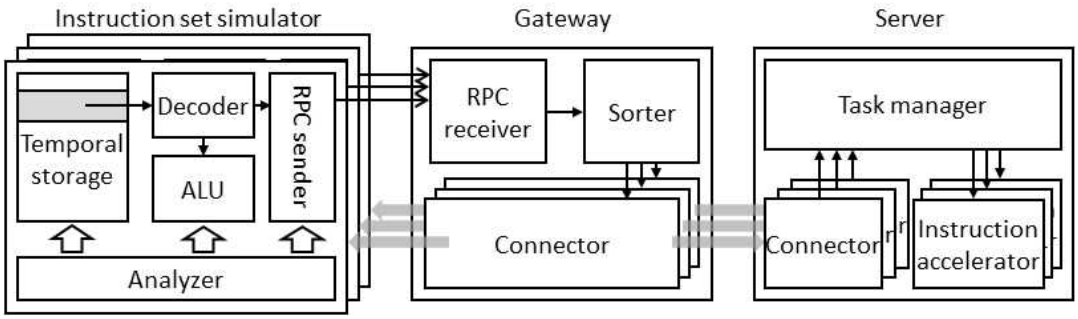

그림 7. 원격 명령어 다중 호출을 위한 게이트웨이 모델
Fig. 7 The gateway model for support remote instruction multi-call

속기의 종류를 테이블로 가지고 있으며, 엣지에 어떤 서버를 연결할지 결정한다. 그 후 엣지와 서버간 통신을 중개해 준다. 서버는 게이트웨이가 중개하는 연결을 통해 원격 명령어 요청과 데이터를 받아 처리한다. 다수의 연결 요청을 받을 경우 관리자에 의해 처리 순서가 정해지고 가속기에서 연산을 수행한다.

Ⅳ. 실험 및 결과

제안한 원격 명령어 실행 구조의 실효성을 검증하기 위해 제안한 구조를 실제 구현하여 실험을 진행하였다. 서버는 ODROID-XU4, 게이트웨이는 ARTiGO A820을 사용하였으며 Ubuntu 16.04 Linux 운영체제에서 구현한 구조를 동작하고 프로그램 수행 시간과 코드의 크기 관점에서 분석하였다. 그림 8은 실제 실험에서 사용한 ARTiGO A820 게이트웨이와 ODROID-XU4이며, 표 1, 2는 ARTiGO A820과 ODROID-XU4의 CPU, 메모리 사양을 나타낸다.

표 1. ARTiGO A820 사양
Table 1. Specification of ARTiGO A820

	ARTiGO A820
CPU	Cortex-A9 @ 1GHz
Memory	1GB DDR3 SDRAM

표 2. ODROID-XU4 사양
Table 2. Specification of ODROID-XU4

	ODROID-XU4
CPU	Cortex-A15 @ 2GHz quad-core and Cortex-A7 quad-core
Memory	2GB LPDDR3 RAM 14.9GB/s

그림 8. 시뮬레이션 환경 (ARTiGO A820과 ODROID)
Fig. 8 Simulation environment (ARTiGO A820 and ODROID)

그림 8의 시뮬레이션 환경에서 명령어 실행 구조를 구현하고, 행렬 곱셈 연산을 예제로 실효성을 검증해 보았다. 벤치마크는 n차원 정방 행렬 데이터를 준비하고 실행하는 프로그램이다. 코드는 행렬을 초기화하는 코드와 행렬을 연산하는 코드로 구성된다. ARTiGO A820 게이트웨이에서 통신 중개를 위한 게이트웨이와 ISS로 모사되는 엣지를 시뮬레이션 하였고, ODROID-XU4에서 서버의 역할을 시뮬레이션 하였다. 원격 명령어를 사용할 대상은 간단한 행렬 곱 연산으로, 수행할 프로그램을 C언어로 표현한 후 ARM 크로스 컴파일러로 컴파일을 했다.

그림 9는 원격 명령어를 사용하지 않았을 때와 사용하였을 때의 코드 크기를 나타낸다. 기존의 코드를 ARM 크로스 컴파일러로 컴파일을 하였을 때와 비교했을 때 원격 명령어를 사용하면 약 24%의 코드 크기 감소를 얻을 수 있었다.

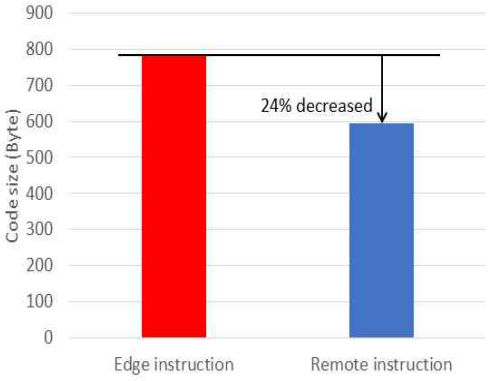

그림 9. 원격 명령어 사용으로 인한 코드 크기 변화
Fig. 9 Code size change due to remote instruction

실험은 행의 개수와 열의 개수가 동일한 정방행렬을 곱하는 프로그램을 입력으로 진행하여 원격 명령어를 사용하였을 때의 수행 시간과 원격 명령어를 사용하지 않았을 때의 수행 시간을 비교해 보았다. 대표적인 임베디드 프로세서인 Cortex-M4는 180MHz로 동작한다. 그러므로 ISS를 약 200MHz 동작으로, 서버를 2GHz 동작으로 가정해 서버의 가속기를 사용하였을 때 하나의 연산을 처리하는 속도가 엣지에서 연산을 처리하는 속도의 약 10배라고 가정하고 시뮬레이션을 수행하였다.

본 논문에서는 엣지와 게이트웨이 시뮬레이션을 ARTiGO 게이트웨이에서 실행하였고, 서버는 ODROID 기기에서 실행하였다. 실제 상황에서 임베디드 장치인 엣지와 서비스를 제공하는 서버는 멀리 떨어져 있으므로 시뮬레이션 코드 내부에 임의로 딜레이를 넣어 통신 지연 시간을 표현해 주었다.

그림 10은 행렬의 행 개수에 따른 프로그램 수행 시간을 비교한 그래프이다. 이 때, 가로축은 프로그램에서 연산하는 정방행렬 행의 개수이고, 세로축은 프로그램 수행 시간을 나타낸다. 행의 개수가 10개일 때에는 줄어드는 연산 시간보다 데이터를 전송하는 시간이 더 오래 걸려 엣지에서 연산을 하는 것이 더 빠른 것을 볼 수 있다. 그러나 행의 개수가 50개가 되어 프로그램이 수행해야 하는 연산이 더 많아질수록 원격 명령어를 사용했을 때 프로그램 수행 시간이 더 많이 줄어드는 것을 볼 수 있다. 시뮬레이션 결과 행이 50개인 정방행렬 곱셈 연산을 원격 명령어로 실행했을 경우 실행 시간은 6.35ms로 엣지에서 프로그램을 수행한 시간 12.56ms에 비해 프로그램 수행 시간이 약 50% 감소한 것을 볼 수 있다.

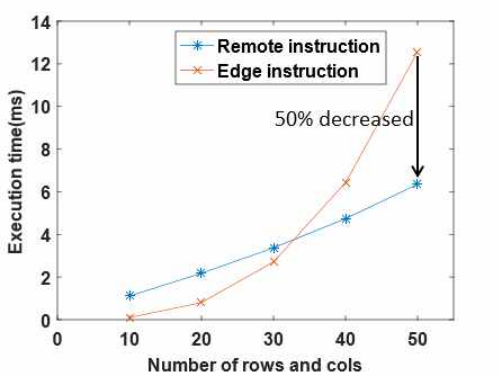

그림 10. 행렬 행의 개수에 따른 연산 수행 시간
Fig. 10 Execution time according to the number of matrix rows

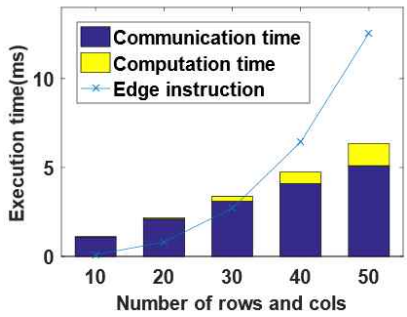

그림 11. 원격 명령어 수행 시간 분류
Fig. 11 Remote instruction execution time classification

그림 11은 위의 원격 명령어 수행 시간을 통신 시간과 연산 시간으로 나눈 그래프이다. 막대그래프는 원격 명령어의 수행시간을 통신 시간과 연산 시간으로 나누어 표현한 그래프이고, 선은 엣지에서 연산했을 때의 시간을 나타낸다. 그림 11에서 볼 수 있듯이 데이터가 적을 때는 통신 시간에 의한 실행 시간 증가가 연산 시간의 감소보다 크기 때문에 원격 명령어가 일반 명령어보다 수행 시간이 길다. 반면에 데이터가 많아지면 통신 지연 시간보다 연산 시간의 감소가 커져 전체 수행시간이 줄어드는 것을 볼 수 있다. 시뮬레이션의 결과로 원격 명령어 구조를 적용할 때 통신이 많고 연산이 적은 명령어보다 연산이 많은 명령어를 원격 명령어로 만드는 것이 더 효과적임을 알 수 있다. 위의 시뮬레이션 결과에서 50개의 행을 가진 정방행렬의 곱셈 연산 프로그램에 원격 명령어를 적용하면 코드

의 크기가 약 24% 감소하고, 수행 시간이 약 50%가 감소하는 것을 볼 수 있다.

Ⅴ. 결론

본 논문에서는 제한된 임베디드 환경에서 하드웨어 추가 없이 원격 명령어를 통해 서버의 가속기를 사용함으로써 고성능 저전력 임베디드 구조를 제안하고 시뮬레이션을 통해 구조의 실효성을 제시하였다. 제안한 구조에서 엣지로 표현되는 임베디드 장치는 가속이 필요한 동작을 원격 명령어로 표현하고 실행한다. 엣지에서 실행된 원격 명령어는 게이트웨이를 거쳐 서버에 전달된다. 서버는 각 원격 명령어에 대응하는 하드웨어 가속기를 가지고 있다. 임베디드가 원격 명령어를 실행하면 서버에서 가속기가 동작하고 연산 결과를 게이트웨이에 의해 만들어진 연결을 통해 반환한다. 제안한 구조를 적용한 임베디드 환경에서 연산량이 많은 프로그램을 수행할 경우 원격 명령어를 사용하면 하드웨어 가속기를 사용할 때처럼 프로그램 수행시간이 빨라진다. 이러한 경우 임베디드 환경에 명령어를 위한 하드웨어 추가가 필요하지 않으므로 제한된 크기의 임베디드에서 무제한에 가까운 종류의 가속기를 이용할 수 있다. 임베디드는 연산 속도가 빨라지고 하드웨어의 크기가 작아지므로 상시 전류가 줄어들어 전력 소모가 줄어든다. 그리고 원격 명령어는 임베디드에서 실행되지만 실제 연산은 서버에서 수행하므로 연산에 필요한 전력 소모가 줄어들어 저전력 고성능 임베디드 환경을 만들 수 있다. 하지만 연산량이 적은 프로그램을 원격 명령어로 만들 경우 통신 지연 시간으로 인해 프로그램 실행 시간이 더 길어지고, 이로 인한 추가적인 전력 소모가 발생할 수 있다. 본 논문에서 제안하는 구조는 다수의 엣지와 게이트웨이, 서버가 엮인 복잡한 시스템이다. 현재 시뮬레이션에서는 연산 가속이 필요한 부분을 직접 찾아 원격 명령어로 변환하였다. 향후 연구에서는 시뮬레이션을 통해 원격 명령어로 서버에서 실행하는 코드와 엣지에서 실행하는 코드의 최적 비율을 찾고, 코드를 생성하는 플랫폼에 대한 연구를 할 것이다.

References

[1] I. Lee, K. Lee, “The Internet of Things (IoT): Applications, Investments, and Challenges for Enterprises,“ Journal of Business Horizons, Vol. 58, No. 4, pp. 431-440, 2015.

[2] B. Karg, S. Lucia, "Towards Low-energy, Low-cost and High-performance IoT-based Operation of Interconnected Systems," Proceedings of IEEE 4th World Forum on Internet of Things (WF-IoT), pp. 706-711, 2018.

[3] G. Fortino, A. Guerrieri, W. Russo, C. Savaglio, “Integration of Agent-based and Cloud Computing for the Smart Objects-oriented IoT,” Proceedings of the IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 493-498, 2014.

[4] R. Banakar, S. Steinke, Bo. Lee, M. Balakrishnan, P. Marwedel, "Scratchpad Memory: a Design Alternative for Cache On-chip Memory in Embedded Systems," Proceedings of the Tenth International Symposium on Hardware/Software Codesign. pp. 73-78, 2002.

[5] R.K. Sharma, “Embedded Systems Dilemma of Chip Memory Diversity by Scratchpad Memory for Cache On-chip Memory," Journal of Engineering, Pure and Applied Sciences, Vol. 1, No. 1, pp. 1-4, 2016.

[6] E. Kusmenko, B. Rumpe, S. Schneiders, M.V. Wenckstern, “Highly-Optimizing and Multi-Target Compiler for Embedded System Model,” Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp447-457, 2018.

[7] K. H. Lee, N. Verma, "A Low-Power Processor with Configurable Embedded Machine-Learning Accelerators for High-Order and Adaptive Analysis of Medical-Sensor Signals," Journal of Solid-State Circuits, Vol. 48, No. 7, pp. 1625-1637, 2013.

[8] G.C. Cardarilli, L.D. Nunzio, R. Fazzolari, M. Re, F. Silvestri, S. Spano, “Energy Consumption Saving in Embedded Microprocessors Using Hardware Accelerators,“ Journal of Telkomnika, Vol. 16,

No. 3, pp. 1019-1026, 2018.

[9] G. Gracioli, A.A. Frohlich, “An Operating System Infrastructure for Remote Code Update in Deeply Embedded Systems,” Proceedings of the 1st International Workshop on Hot Topics in Software Upgrades, Article 3, pp 1-5, 2008.

[10] D. Park, J. Cho, "Cloud-Connected Code Executable IoT Device with On-cloud Virtually Memory Controller for Dynamic Instruction Streaming," Proceedings of International Conference on Cloud Computing and Big Data (CCBD), pp. 29-30, 2015.

Dongkyu Lee (이 동 규)

He received the B.S degree in electronics engineering at Kyungpook National university, Daegu, Korea in 2018. He is currently a M.S student in department of electronics engineering at Kyungpook National university, Daegu, Korea. His research interests include high performance acceleration on big data processing. His research was focused on designing energy-efficient cloud-connected processors in silicon chip level.

Email: ehdrbxp@gmail.com

Jeonghun Cho (조 정 훈)

He received the B.S. degree in EE, the M.S. and the Ph. D degree in EECS from the Korea Advanced Institute of Science and Technology (KAIST), Deajeon, Korea in 1996, 1998, and 2003, respectively. He was a senior engineer at Hynix Semiconductor from 2003 to 2005. Main role was development of a C compiler for 8-bit microcontrollers. He is currently a professor with the School of EE of Kyungpook National University, Daegu, South Korea since 2005. His research interest includes a binary translation, software safety and security for automotive, AUTOSAR, run-time monitoring and logging for embedded system. Dr. Cho is a member of IEEE.

Email: jcho@knu.ac.kr

Daejin Park (박 대 진)

He received the B.S. degree in electronics engineering from Kyungpook National University, Daegu, Korea in 2001, the M.S. degree and Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2003, and 2014, respectively. He was a Research Engineer at Major Semiconductor Companies such as SK Hynix Semiconductor, Samsung Electronics over 12 years from 2003 to 2014, respectively and have worked on processor architecture design and low-power ASIC implementation with custom designed software algorithm optimization. Dr. Park is now with School of Electronics Engineering as full-time assistant processor in Kyungpook National University, Daegu, Korea and presidential research fellow.

Email: boltanut@knu.ac.kr