https://doi.org/10.21289/KSIC.2021.24.5.531

인공지능 응용을 위한 하이브리드 메모리 설계 탐색 기법

An Design Exploration Technique of a Hybrid Memory for Artificial Intelligence Applications

조두산*

Doo-San Cho*

〈Abstract〉

As artificial intelligence technology advances, it is being applied to various application fields. Artificial intelligence is performing well in the field of image recognition and classification. Chip design specialized in this field is also actively being studied. Artificial intelligence-specific chips are designed to provide optimal performance for the applications. At the design task, memory component optimization is becoming an important issue. In this study, the optimal algorithm for the memory size exploration is presented, and the optimal memory size is becoming as a important factor in providing a proper design that meets the requirements of performance, cost, and power consumption.

Keywords : Matrix Multiplication, Memory Design, Design Space Exploration, Memory Capacity, Low Power, Performance

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

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

1. 서 론

딥러닝 알고리즘의 개발로 가속화된 인공지능 응용 연구는 GPU의 기술 발달로 그 활용 영역의 폭이 우리 생활권에 깊숙이 들어오게 되었다. 삼성 빅스비, 구글의 텐서플로우, 마이크로소프트의 코타나, IBM의 왓슨, 아마존의 알렉사, 페이스북의 레블 등의 인공지능은 다양한 분야에서 이용되고 있으며, 대표적인 서비스는 자연어를 인식하여 필요한 서비스를 제공하는 루틴으로 구성되어 있다. 이러한 인공지능은 다양한 학습을 통하여 언어를 이해하는 뉴럴 네트워크 회로를 구성하고, 이렇게 구성된 네트워크를 스마트 디바이스에 탑재하여 자연어를 처리하는 서비스로 이용되고 있다. 이러한 인공지능 서비스는 실시간 학습을 진행하기 때문에 사용시간이 길수록 더 높은 정확도의 서비스를 제공할 수 있다. 특히, 인공지능 제품은 이미지 인식/분류에서 괄목할 만한 성과를 거두어 해당 분야에서는 사람보다 높은 정확도를 제공하기에 이르렀다. 이미지 인식/분류 응용에서 사람보다 높은 정확도를 제공하기 위해서는 대량의 이미지 데이터를 기반으로 학습하여 인공지능 신경망을 구축해야 한다. 이러한 학습 단계는 대량의 데이터를 기반으로 진행되기 때문에 대용량 메모리를 이용하게 된다. 대용량 메모리는 시스템의 에너지 소비, 성능, 비용에 가장 큰 영향을 주는 요소이다. 따라서 응용 분야에 최적으로 구성된 메모리 설계는 인공지능 제품의 성공을 결정하는 매우 중요한 컴포넌트가 된다.

Fig. 1은 하나의 인공지능 서비스를 구성하기 위하여 거치는 과정을 단순하게 나타낸 것이다. 하나의 서비스가 완성될 때까지 대량의 데이터가 지속해서 사용되기 때문에 메모리의 성능이 중요하다. 따라서 메모리의 성능, 속도, 용량, 에너지 소모량 등의 변수들이 전체 시스템의 성능에 큰 영향을 미치게 된다. 여기서 문제는 큰 사이즈의 메모리는 충분한 공간을 제공하지만 에너지 소모량과 속도에서 요구되는 조건을 만족하기 어렵다. 따라서 목표 응용 서비스에서 요구되는 최적 사이즈의 메모리를 탐색하고, 이에 맞는 에너지 소모량과 성능을 맞추어 설계하는 것이 중요하다. 다음 장에서는 본 연구에서 제안하는 메모리 사이즈 탐색 기법에 대하여 설명한다. 3장에서는 실험 결과에 대하여 기술하고, 4장에서 결론을 맺도록 하겠다.

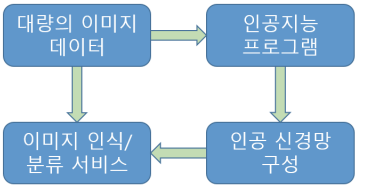

Fig. 1 A simplified process to provide an image classification artificial intelligence service

2. 메모리 사이즈 탐색 기법

이미지 파일은 기본적으로 행렬 형태의 데이터 형식으로 저장되어 사용된다. 따라서 이미지 처리 연산은 대부분 배열의 행렬 연산으로 구현된다. Fig. 2는 이미지 프로세싱 응용에서 주로 사용되는 행렬 곱셈 코드를 나타내고 있다. 2차원 행렬

```
void multiply(int mat1[][N],
     int mat2[][N],
     int res[][N])
{    int i, j, k;
     for (i = 0; i < N; i++) {
          for (j = 0; j < N; j++) {
               res[i][j] = 0;
               for (k = 0; k < N; k++)
                 res[i][j] += mat1[i][k] * mat2[k][j];
 }}}
```

Fig. 2 Matrix multiplication

mat1과 mat2가 곱해져서 res행렬이 계산된다. 이때 캐시 메모리의 관점에서 두 행렬의 계산 순서를 먼저 살펴보겠다.

Fig. 3에 나타난 바와 같이 mat1과 mat2가 주요 읽기 액세스의 대상이 되는 데이터이다. mat1과 mat2는 루프의 k 인덱스 변수에 의하여 행의 순서와 열의 순서로 데이터가 참조된다. 이때 행의 순서로 데이터가 참조되면 캐시 메모리와 같은 온칩 메모리에 의하여 성능이 대폭 개선된다. 하지만 저가형 캐시 메모리의 경우 열의 순서로 사용되는 mat2의 경우 데이터 사용 지역성이 상대적으로 낮기 때문에 캐시 히트율이 대폭 떨어진다. 이를 개선하기 위해서는 열 순서로 사용되는 데이터를 행순서로 변경하거나 데이터 변경이 용이하지 않은 경우 메모리 상의 데이터 배치를 변경하는 방법으로 캐시 히트율을 개선하는 방법을 사용할 수 있다.

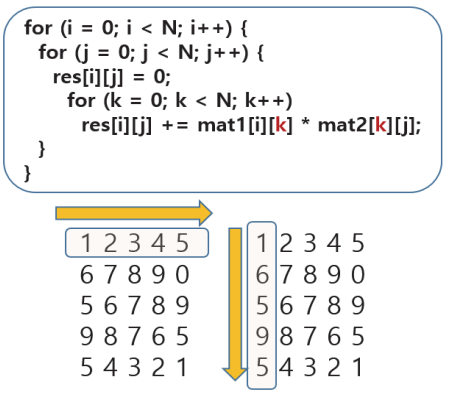

Fig. 3 Data access pattern

이렇게 소프트웨어적인 최적화는 컴파일러에 의하여 진행되며, 이를 다양한 응용 코드를 대상으로 구현하려면 해당 기법을 일반화하여 개발해야 한다. 본 연구에서는 컴파일러를 사용하여 데이터 지역성을 개선하는 방식보다는 하드웨어를 이용하여 캐시 메모리의 히트율을 개성하는 방법을 제안한다.

최근의 메모리 아키텍처는 소프트웨어 제어 캐시 (software controlled cache memory, SCCM)와 하드웨어 제어 캐시 (hardware controlled cache memory, HCCM)를 혼합하여 하이브리드 메모리 형태로 설계하는 방식을 주로 이용한다. 각 메모리의 단점을 상호 보완해주는 효과를 이용하여 메모리 성능을 극대화할 수 있기 때문이다. 메모리 설계 단계에서 대상 응용에 최적화된 메모리 사이즈 및 구조를 탐색하게 되는데, 하이브리드 메모리에 맞는 최적 사이즈 및 구조를 찾는 것은 성능, 에너지 소모량, 비용 결정에 큰 영향을 주게 된다.

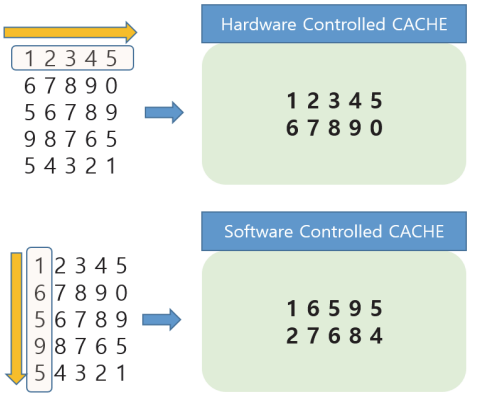

Fig. 4 An example of data caching

Fig. 4를 보면 하드웨어 제어 캐시와 소프트웨어 제어 캐시를 볼 수 있는데, 하드웨어 제어 캐시는 mat1의 액세스 순서에 따라 데이터를 프리페칭하여 메모리에 업로드한다. 반면 소프트웨어 제어 캐시는 컴파일러에 의하여 데이터 액세스 순서를 미리 프로파일링하고, 프로파일링 정보에 따라 사용되는 데이터를 미리 메모리에 프리페칭 한다. 하드웨어 제어 캐시는 하드웨어에 의하여 자동으로 데이터 캐싱이 수행되고, 소프트웨어 제어 캐시는 컴파일러에 의하여 삽입된 명령어로 데이터가 사용되기 이전에 메모리로 데이터가 캐시(업로드)되는 점이 다르다.

하이브리드로 구성되는 메모리의 각 사이즈를 목표 응용에 최적으로 맞추어 구성한다면 적은 비용으로 최적의 성능을 제공하는 메모리를 설계할 수 있게 된다. Fig. 4의 예제를 보면 mat1이 HCCM 그리고 mat2은 SCCM에 데이터가 배치될 수 있다. 이러한 결정은 데이터 사용 지역성을 이용하여 결정할 수 있다. 본 연구에서는 데이터 지역성을 참조되는 데이터들 간격의 메모리 상위에서 계산되는 거리로 정의한다. 예를 들어서, 배열 mat2는 액세스되는 2개 데이터 참조들 사이의 거리가 6-1=5가 된다. 각 참조되는 데이터는 5개의 데이터를 간격으로 참조되는 것이다. 이러한 데이터 참조 간격은 한 개 루프 코드안에서 결정되며, 데이터 액세스가 선형함수, Fig. 3에서 [i][k] 혹은 [k][j],로 표현되지 않을 경우 데이터 참조 간격의 평균 값으로 지역성을 계산한다. 계산 방식은 아래 식과 같다.

Equation (1)

spatial data locality =
AVERAGE(accessed_address(i_th_data) -
accessed_address(i+1_th_data))

eq. (1)은 임의의 루프 코드 안에서 참조되는 i번째 데이터의 메모리 위에서의 주소와 i+1번째 데이터의 주소의 차들의 평균 값을 나타낸다. 위 식 eq. (1)에 의하여 계산된 공간 데이터 지역성(spatial data locality)의 크기가 캐시 라인 사이즈보다 크다면 해당되는 데이터는 SCCM에 저장되는 것이 바람직하다. 현재 mat2는 5개의 정수 데이터, 즉 20 bytes가 공간 데이터 지역성 값이기 때문에 캐시 라인 사이즈가 20바이트보다 작다면 mat2는 SCCM에 저장되도록 프로그램 코드를 컴파일러가 생성하게 한다.

이러한 방식으로 배열 데이터의 할당에 따라 메모리 크기를 구성한다면 메모리의 사이즈를 설계 단계에서 적절하게 결정할 수 있게 된다. 이러한 절차를 나타내는 알고리즘을 Fig. 5에 기술하였다.

```
Input: an application code
Output: optimized memory size
Memory_size_exploration(APP_CODE)
{
  list_access_patterns =
Get_data_access_pattern(array_dimension,
array_index_variable, access_stride);
  ratio = HCCM_hit_rate(access_pattern);

  SCCM_size=0;
  HCCM_size=0;
  while( current=pick_one(list_access_patterns) ){
    if(ratio < 50)
      placement(current, SCCM);
      SCCM_size += get_size(current);
    else
      placement(current, HCCM);
      HCCM_size += get_size(current);
  }
}
```

Fig. 5 An algorithm for memory size exploration in pseudo code.

Fig. 5의 알고리즘을 보면 먼저 list_access_patterns 변수를 이용하여 프로그램에서 사용하는 변수의 데이터 액세스 패턴 정보를 추출한다. Get_data_access_pattern 함수는 프로그램에서 사용되는 모든 배열의 차원, 인덱스 변수, 데이터 참조(액세스) 간격 정보를 모두 액세스 패턴 정보로 제공한다. 제공된 정보를 이용하면 하드웨어 제어 캐시를 이용하였을 때 캐시 히트 비율을 시뮬레이션하여 그 결과를 받을 수 있기 때문에 해당 변수가 HCCM에 저장되는 것이 유리할지 혹은 SCCM에 저장되면 유리할 지를 결정할 수 있다. 변수 ratio는 HCCM에 저장하였을 때 얻을 수 있는 캐시 히트 비율의 평균값을 나타낸다.

본 알고리즘에서는 HCCM 캐시 히트 비율이 50% 이하인 경우 해당 데이터를 SCCM에 배치하여 전체 시스템 성능 개선을 유도하고 있다. SCCM은 소프트웨어 제어 캐시이다. 다시 말하면, 컴파일러가 SCCM을 위한 데이터 업로드 (캐시)를 위한 명

령어를 프로그램 안에 생성하고 데이터의 캐시 타이밍을 결정하기 때문에, HCCM에서 성능을 저하시키는 데이터의 지역성 문제에 기인하는 낮은 캐시 히트율 문제를 해결할 수 있게 된다.

while문에서 list_access_patterns으로부터 데이터 액세스 패턴을 한 개씩 추출하여 current에 저장한다. 따라서 current는 현재 계산할 데이터의 액세스 패턴을 저장하고 있다. while문은 더 이상 할당할 데이터가 없을 때 까지 반복한다. if문에서 각 배열 데이터의 메모리 할당, 즉, SCCM 혹은 HCCM로의 할당을 placement함수로 결정하고, SCCM_size와 HCCM_size 변수에 할당된 변수들의 크기를 모두 합하여 대상 프로그램에서 요구되는 메모리 크기를 결정하게 된다. 메인 메모리에서 SCCM으로의 데이터 이동은 DMA에 의해서 진행된다. 따라서 DMA 명령어가 SCCM과 메인 메모리 사이의 데이터를 관리하도록 생성되어야 하는데, 이것은 컴파일러에 의하여 생성되게 한다.

3. 결과 및 고찰

본 논문에서는 이미지 분류용 인공지능에 사용되는 이미지 배열 데이터를 프로세싱하는 행렬 곱연산 애플리케이션 코드를 대상으로 요구되는 메모리 사이즈 탐색 기법을 평가하였다. 행렬 변수는 대량의 데이터를 담고 있기 때문에 해당 행렬을 사용하는 액세스 패턴에 따라 HCCM의 히트 비율이 가변적이다. 따라서 HCCM 캐시 히트 비율이 낮은 행렬들을 SCCM에서 참조 및 사용하도록 한다면 전체 시스템 성능이 개선될 수 있다. 본 연구는 이러한 부분에 착안하여 각 응용 프로그램에서 요구되는 최적 메모리 크기를 탐색하고 그 결과를 Table 1에 기술하였다. 실험을 위해서 랜덤하게 생성된 이미지 행렬 데이터를 사용하였다.

Table 1. Experimental result

	Performance (cycle)	Energy consumption (nJ)	#accesses	Size
NO-OPT	3932160	1035469	262144	64
OPT	327672	675020	262144	64
NO-OPT	31457280	8283750	2097152	128
OPT	16777216	675020	2097152	128

Table 1은 행렬 사이즈 64/128에 대하여 총 액세스 횟수, 에너지 소비, 성능을 제안된 기법을 적용하였을 때 (OPT)와 아닌 경우 (NO_OPT)로 구분하여 정리하고 있다. NO_OPT는 HCCM만으로 실행되며, OPT는 Fig. 5에 기술된 알고리즘에 따라 입력된 행렬 데이터가 HCCM과 SCCM에 분할 할당된다. 한 개 응용 프로그램에서 사용되는 배열 데이터로부터 알고리즘에 의하여 할당된 데이터 사이즈들의 총합이 해당 프로그램의 최적 메모리 사이즈가 된다.

제안된 알고리즘을 입력된 이미지 프로세싱 프로그램을 대상으로 수행한 결과 이미지 데이터로 구성된 행렬 64/128 사이즈 배열 각각에 대하여 SCCM과 HCCM에 50%, 50%로 할당되었다. Table 1에 나타난 바와 같이, 제안된 알고리즘에 의하여 입력된 이미지 프로세싱 프로그램의 에너지가 평균 63% 개선되었고, 성능은 평균 69% 개선될 수 있었다.

IoT/에지 컴퓨팅에서 저전력 메모리 아키텍처[1], 메디컬 컴퓨팅에서 메모리 설계[2], 특정 메모리 구조에서 데이터 배치[3], 메모리 설계를 위한 머신러닝/인공지능 기법[4,5] 등의 기존 연구는 프로그램 코드의 특성보다는 메모리 특성에 맞는 설계를 탐색하는 연구였다. 설계 단계 이외에 프로그램 명령어 생성 단계에서 메모리 성능을 개선하는[6][7], [8][9]와 같이 응용 프로그램을 메모리 구조에 최적화하는 설계 툴의 개발과 설계에 대한 연구들은 본 연구에서 제안된 기법과 함께 다양한

응용에서 사용이 가능할 것으로 예상된다. 멀티미디어 응용과 같은 데이터 스트리밍으로 메모리 대역폭이 제한된 하이브리드 메모리 아키텍처에서 성능 개선 기법[10]과 하이브리드 멀티뱅크 메모리에서 에너지 절감 기법[11]에서도 본 연구 결과를 동시 적용 가능할 것으로 예상한다.

4. 결 론

본 연구에서는 이미지 인식 관련된 인공지능 프로그램 응용을 위한 시스템 칩 설계를 진행할 때 최적 메모리 설계를 지원하기 위한 알고리즘을 제안하고 있다. 이미지 응용 프로그램은 일반적으로 이미지를 저장한 배열 데이터를 대상으로 행렬 연산을 수행한다. 이러한 대량의 행렬 연산을 수행할 때 응용 프로그램에 요구되는 최적의 메모리 사이즈 결정은 설계 단계에서 수행하는 중요한 태스크 중의 하나이다. 제안된 기법을 사용하면 설계 단계에서 얻을 수 있는 하이브리드 메모리 사이즈 파라미터 값을 얻을 수 있다. 제안된 기법으로 결정된 메모리 사이즈 파라미터는 배열 데이터의 액세스 패턴을 기반으로 계산된 캐시 히트율을 고려하여 결정된 것이다. 이것을 사용하면 최적 메모리 설계에 도움이 될 수 있을 것으로 예상된다.

Acknowledgement

This work was supported by a Research promotion program of SCNU.

참고문헌

[1] D. Cho, A Study on Improvement of Low-power Memory Architecture in IoT/edge Computing. Journal of the Korean Society of Industry Convergence, Vol. 24, No. 1, pp. 69-77, (2021).

[2] J. Cho, J. Lee, D. Cho, Efficient memory design for medical database. Basic & Clinical Pharmacology & Toxicology, Vol. 125, pp. 198, (2019).

[3] J. Cho, JM Youn, D. Cho, An Automatic Array Distribution Technique for Multi-Bank Memory of High Performance IoT Systems. World, Vol. 3, No. 1, pp. 15-20, (2019).

[4] D. Cho, Study on Memory Performance Improvement based on Machine Learning. The Journal of the Convergence on Culture Technology, Vol. 7, No. 1, pp. 615-619, (2021).

[5] D. Cho, Memory Design for Artificial Intelligence. International Journal of Internet, Broadcasting and Communication, Vol. 12, No. 1, pp. 90-94, (2020).

[6] J. Yoon, D. Cho, A spill data aware memory assignment technique for improving power consumption of multimedia memory systems. Multimedia Tools and Applications, Vol. 78, No. 5, pp. 5463-5478, (2019).

[7] J. Cho, J. Lee, D. Cho, Low-End Memory Subsystem Optimization Process. International Journal of Smart Home, Vol. 13, No. 2, pp. 11-16, (2019).

[8] J. Cho, D. Cho, Y. Kim, Study on LLVM application in Parallel Computing System. The Journal of the Convergence on Culture Technology, Vol. 5, No. 1, pp. 395-399, (2019).

[9] J. Cho, D. Cho, Development of a Prototyping Tool for New Memory Subsystem. International Journal of Internet, Broadcasting and Communication, Vol. 11, No. 1, pp. 69-74, (2019).

[10] J. Yoon, D. Cho, Improving memory system performance for multimedia applications. Multimedia Tools and Applications, Vol. 76, No. 4, pp. 5951-5963, (2017).

[11] J. Cho, J. Yoon and D. Cho, Improvement Energy Efficiency for a Hybrid Multibank Memory in Energy Critical Applications. Technical gazette, vol.27, no. 6, pp. 1946-1955, (2020).

(접수: 2021.07.16. 수정: 2021.07.29. 게재확정: 2021.07.30.)