머신러닝 기반 데이터 프리페처

Data prefetcher based on machine learning

요 약

컴퓨터의 성능 향상을 위해 도입된 데이터 프리페처는 메모리 접근의 시/공간적 지역성을 활용하여 다음 접근 메모리 주소를 예측하게 된다. 본 논문에서는 시계열 데이터를 다루는데 유리한 LSTM 모델을 적용한 데이터 프리페처를 제안한다. 메모리 주소들의 offset을 이용하여 연속된 메모리 접근 주소 간의 차이(delta)를 계산하고, 현재 접근한 주소를 2진법으로 나타낸 뒤, 이를 학습 데이터로 사용하여 LSTM 모델을 학습시킨다. 제안된 데이터 프리페처는 메모리 접근 패턴이 학습된 LSTM모델을 사용하여 다음에 접근하게 될 메모리 주소를 예측하게 된다. 제안된 데이터 프리페처의 성능은 ChampSim 시뮬레이터와 SPEC CPU2006 벤치마크를 사용하여 검증하였다. 실험 결과를 통해 LSTM기반 프리패처가 메모리 접근 패턴이 복잡한 벤치마크에 대해서는 기존 프리패처에 비해 높은 예측 정확도를 보임을 확인하였다.

1. 서 론

프로그램을 실행하거나 데이터를 참조하기 위해서 용량은 크지만, 속도가 느린 메인 메모리에서 사용할 데이터를 처리속도가 빠른 프로세서로 전송해야 한다. 프로세서에서 처리하는 속도에 비해 메인 메모리에서 데이터를 가져오는 속도가 현저히 느리기 때문에 프로세서는 오랜 시간 데이터를 기다리게 되고 속도의 격차로 인한 컴퓨터의 성능 저하를 초래한다[1].

이를 해결하기 위해서 메모리를 계층화시키는데 이때 프로세서와 메인 메모리 사이에 캐시메모리를 추가하여 속도 차이를 해결하게 했다. 그러나 캐시 미스가 발생 된다면 먼저 메인 메모리 같은 메모리 계층의 하부에 접근하여 그 데이터를 가져와서 캐시에 저장시켜서 그 데이터를 사용하게 되므로 캐시 미스가 많을수록 메모리 latency가 늘어난다.

이러한 캐시 미스가 발생하지 않게 미리 사용될 가능성이 높은 데이터를 예측하여 캐시에 저장해 놓을 수 있다면 프로세서는 데이터를 요청하고 기다리는 시간이 줄어들게 되는데 이러한 방식을 데이터 프리페치라고 한다. 프로세서는 처리해야 할 데이터를 빠르게 받아서 처리할 수 있어 메모리 latency를 감출 수 있게 된다.

데이터 프리페처들은 프로그램이 가지고 있는 공간적, 시간적 지역성을 활용하여 메인 메모리에 있는 데이터를 캐시 메모리에 저장한다. 지역성을 이용해 패턴을 분석하여 현재 메모리 접근에 대하여 그다음에 사용될 메모리 주소를 예측하게 되는데, Delta 기법이 이러한 프리페처들의 근간이 된다. 메모리 접근 주소의 차이를 Delta 값으로 두어 학습을 진행하고 다음에 접근할 주소를 예측하게 된다.

기존 데이터 프리페처들은 일정한 알고리즘에 의해 다음 주솟값을 예측한다. 가장 기본이 되는 데이터 프리페처는 Next Line Prefetcher[2]이다. 이는 단순히 현재 접근한 주소에 대하여 그다음 주소 값을 프리페치하게 된다. Next Line Prefetcher를 발전시킨 모델로는 Best Offset Prefetcher[3]이 있다. 이는 바로 다음 주소가 아닌 자주 사용되는 offset 값으로 다음 주소 값을 예측하여 데이터를 프리페치하게 된다.

머신러닝 기술은 많은 양의 데이터 가운데 비슷한 것끼리 묶어내고 서로 관계있는 것들의 상하 구조를 인식하여 이것을 바탕으로 앞으로 행동을 예측하게 된다. 특히 RNN(Recurrent Neural Network) 계열은 순서, 시간 등의 시계열 데이터를 다루는 데 강점을 보이고 input과 output 길이를 다양하게 적용할 수 있다.

본 연구에서 제안하는 데이터 프리페처는 동일 Page 내에서 연속된 메모리 접근에 대한 메모리 주소의 차이 (delta)를 메모리 주소의 offset을 사용하여 계산한 뒤 이를 table에 저장하게 된다. 이후 delta 값들에 대한 정보와 현재 접근한 메모리 주소를 2진법으로 나타내 이를 학습데이터로 사용하여 LSTM 모델을 학습시킨다. 학습된 LSTM은 delta값을 예측하게 되고 이를 최근에 접근한 메모리 주소에 적용하여 다음 메모리 주소를 예측하게 된다.

2. 프리페처 구성

2.1 Data 구성

```
11, 145, 28e837c87040, 406bc9, 0
12, 160, 28e837c89f00, 406bd0, 0
31, 375, 28e837c87000, 407040, 0
27, 407, 28e837c86240, 406d87, 0
```

그림 1 data의 구성

LSTM 모델을 학습시키기 위해 다양한 벤치마크의 메모리 접

근 Trace를 사용하였으며 Trace에는 그림 1과 같은 정보가 포함되어 있다. 그림 1의 각 행은 개별 메모리 접근 명령어에 대한 정보를 나타내며, Unique Instruction ID, Cycle Count, Load Address, Instruction Pointer, LLC hit/miss로 구성된다. 이 정보 중 연접한 메모리 접근의 Load Address 간 차이(Delta)를 계산하여 데이터 프리페처를 위한 LSTM 모델을 학습시켰다.

2.2 Model의 구성

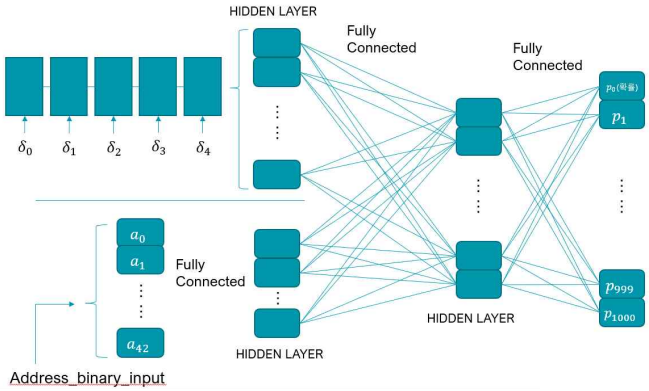

그림 2 LSTM_prefetcher의 시각화

모델은 그림 2과 같이 구성하였다. 먼저 LSTM 모델의 입력은 δ값(동일 Page에서 연접한 메모리 접근의 주소 차이)과 직전에 접근한 Page의 정보이고 Output은 다음에 올 δ값을 나타내 준다. 즉, Page 크기가 128이라 가정하면, Load Address 0 ~ 127 (실제로는 0 ~ 127*64)이 첫 번째 Page를 이루고 128 ~ 255(실제로는 128*64 ~ 255*64)가 2번째 Page를 이룬다. 그렇다면 만약 Page 내부에서만 Load Address값이 바뀐다면, 나올 수 있는 delta값은 -127(−PageSize + 1) ~ 127(PageSize - 1)이 될 것이다. 이를 이용하여, 어느 Page 내부에 있는 Address에 접근하면, Offset과 Delta값이 하나씩 추가되는 형식이다.

각 블록에 대해 하나씩 살펴보자. 먼저 Address에 접근하면 Page의 번호 값이 들어간다. Offset은 Page의 첫 번째 부분부터 얼마나 떨어져 있는지를 나타낸다. 또 Delta에는 현재 Offset과 이전의 Offset 값의 차이가 들어가게 된다. 예를 들어 Page 크기가 100이라고 가정하고, 210, 250, 350, 230, 330, 810, 820, 220, 340(실제로는 210 * 64, 250 * 64 .....인 곳에 접근)에 접근했다고 가정해보자. 처음에 Input이 아무것도 없을 때, 210 Load Address에 접근했을 경우, Page 1에 해당하는 블록에 3이라는 값이 쓰이면서 Offset 1은 10, Delta 1은 0이 쓰이게 된다. 그 후, 250에 접근했을 때 Page의 값은 3이고, Page 1에 3이 이미 존재하므로 Page 1의 offset 2에는 50이, delta 2에는 40 (50 - 10)이 쓰이게 된다. 350에 접근하게 되면, Page 2에는 4라는 값을 써주고 새로운 Page 정보를 만들게 된다. 그 후, Page 2에 해당하는 offset 1, delta 1에는 각각 50, 0이라는 값이 들어간다. 모든 동작을 실행하면 그림 3과 같은 data가 생성된다.

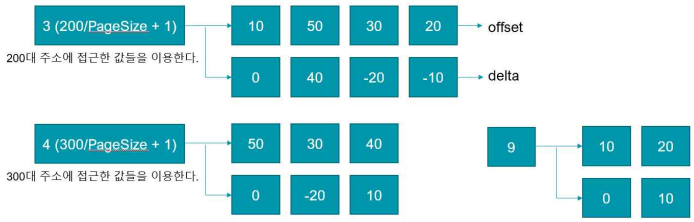

그림 3 LSTM prefetcher의 input예시

그림 3과 같은 데이터를 이용하여 δ input과 output을 제작할 수 있다. 그림 4와 같이 LSTM의 길이가 3일 때, 3개의 δ값이 사용되고, 하나의 Load Address 접근 당 확률이 가장 높은 2개의 값들을 예측하여 프리페치해주기 때문에 여러 개의 δ값을 Output으로 넣어주었다.

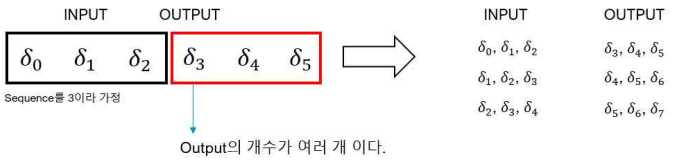

그림 4 여러개의 output

또한 Address_binary_input을 2번째 Input으로 넣어주었는데 이 값은 직전에 접근한 Address의 Page 번호를 Binary 값으로 나타낸 것이다. 이러한 이유는 Page 번호가 너무 많아 Node를 다 추가해줄 수 없으며, Page에 대한 정보를 넣어주어야 특정 페이지에 해당하는 Delta Sequence를 예측할 수 있기 때문이다.

3. 실험

3.1 실험 환경

우리는 Tensorflow로 LSTM모델을 구성하였고 본 프리페처를 평가하기 위해서 ChampSim 시뮬레이터[4]를 사용하였다. 이 프리페처의 성능을 측정하기 위해 Machine Learning Data Prefetching Competition[5]에서 제공된 SPEC CPU2006 benchmark traces를 사용하였다. 제안된 LSTM 모델을 학습하기 위해서 각 벤치마크 당 처음 1억 개의 명령어를 사용하였고 평가를 위해 그다음 1억 개의 명령어를 사용하였다.

데이터 프리페처가 성능적인 측면에서 좋은지에 대한 몇 가지 기준이 있는데 우리는 주된 기준으로 IPC와 이에 대한 보조적 지표로 ACCURACY, COVERAGE를 사용했다. 먼저 IPC는 instruction per cycle로 한 사이클 당 완료 가능한 명령어의 개수를 뜻한다. ACCURACY는 프리페처로 미리 읽어온 데이터 중 몇 개의 데이터가 캐시미스를 해결했는지에 대한 비율을 뜻하고, COVERAGE는 총 캐시 미스 중에 프리페치로 인해 줄어든 캐시미스에 대한 비율을 뜻한다.

IPC는 ACCURACY와 COVERAGE의 조절 통해 값이 나타나

게 되는데, 만약 다음에 올 확률이 아주 작은 데이터 값이라도 다량을 프리페치를 해오게 된다면 총 프리페치 해온 데이터의 수가 늘어나지만 해결한 캐시미스의 수는 소량 늘어나게 될 것이므로 ACCURACY는 줄어들게 된다. 그러나 동일한 수의 캐시미스 내에서 프리페치로 인하여 해결한 캐시미스의 수가 증가하게 되므로 COVERAGE는 증가하게 되고 일정 수준 까지는 IPC의 향상을 나타내게 된다. 또한 만약 한 번에 대량의 데이터를 프리페치하면 이는 캐시 오염도를 증가시키게 되어 IPC를 감소시키게 된다. 본 연구에서는 오버 프리페치로 인한 문제를 방지하기 위해, 데이터 프리페처로 한 번에 읽어오는 데이터의 수를 2개로 제한하였다.

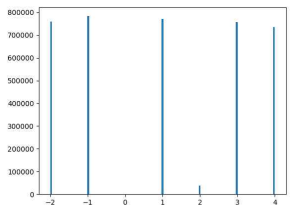

그림 5 lbm의 Delta

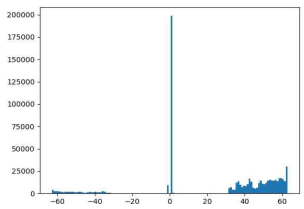

그림 6 astar의 Delta

(x축: 델타값, y축: 델타값의 개수)

3.2 결과 분석

본 연구에서 제안한 프리페처는 프리페처를 사용하지 않았을 때 보다 성능(IPC)이 19% 정도 상승하였다. 기존 프리페처와의 비교에서는 두 가지 경우로 나뉜다. Page 내부에서 Delta값의 종류가 다양하지 않을 경우, Best Offset Prefetcher 및 Next Line Prefetcher와 비교했을 때 LSTM기반 데이터 프리페처의 성능이 비슷하거나 떨어졌고, Delta값의 종류가 다양할 경우, LSTM기반 프리페처의 성능이 우수했다. 그림 5와 같이 Delta값의 종류가 다양하지 않은 lbm 벤치마크에 대해서는 LSTM기반 데이터 프리페처는 Next Line Prefetcher에 비해 약 0.6%의 성능 향상을 보여주었고, Best Offset Prefetcher에 비해서는 대략 4.6% 감소하였다. 그림 6과 같이 Delta값의 종류가 다양한 astar 벤치마크에 대해서는 LSTM기반 데이터 프리페처는 Next Line Prefetcher에 비해 약 4.7%의 성능 상승을 보여주었고 Best Offset Prefetcher에 비해 1.1% 상승 향상을 보여주었다. 이는 Delta값의 종류가 다양할 때 프리페치하기 위해 유동적인 Offset값을 설정해야 하는데, 이러한 측면에서 본 연구에서 제안한 LSTM기반 데이터 프리페처가 더 우수하다고 할 수 있다.

4. 결론 및 향후 연구

본 논문에서는 시계열 데이터의 패턴을 다루는 데 강점을 보이는 LSTM 모델 기반 데이터 프리페처를 제안하였다. LSTM 모델 기반 데이터 프리페처를 적용할 경우, 시스템 성능을 약 19% 향상할 수 있음을 실험을 통해 확인하였다. 이러한 성능 향상은 전통적인 데이터 프리페처와 비슷한 수준이다. 하지만, 메모리 접근 패턴이 복잡한 벤치마크에 대해서는 LSTM 기반 데이터 프리페처가 기존 데이터 프리페처에 비해 약간 더 우수한 성능을 보여줌을 확인하였다.

현재 본 논문에서 제시한 프리페처는 성능향상이 크지 않거나 오히려 떨어지는 부분이 있는데, 이는 같은 페이지 내에 속해 있는 offset에 관한 패턴을 학습하여 다음 주소를 예측하지만, 페이지가 변화할 때는 예측할 수 없다. 이를 위해 페이지 접근 패턴을 분석하고 학습하여 다른 페이지에 있는 값을 예측할 수 있다면 더 많은 성능 향상이 있을 것으로 보인다. 또한 Timeliness가 고려되지 않았으므로 예측된 메모리 주솟값을 프리페치 해오기 전에 이미 그 주솟값이 필요로 된 경우 또한 존재하게 되므로 이를 고려하기 위해서 일정 시간 뒤의 값을 프리페치 하는 방식을 도입하여 프리페처의 성능을 향상할 수 있을 것으로 보인다.

참고문헌

[1] J. L. Hennessy, and D. A. Patterson “Computer Architecture: A Quantitative Approach,” 5th Ed. Morgan Kaufmann Publishers, pp. 72−95, 2012.

[2] Alan Jay Smith, “Sequential program prefetching in memory hierarchies”. In: Computer 11.12 (1978), pp. 7-21.

[3] PP. Michaud, “Best−offset hardware prefetching,” in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 469-480.

[4] “ChampSim,” https://github.com/Quangmire/ChampSim, 2021, [Online;accessed 18−October−2021].

[5] “Machine Learning Data Prefetching Competition,” https://sites.google.com/view/mlarchsys/isca−2021/ml−prefetching−competition?authuser=0, 2021, [Online; accessed 18−October−2021].