스트라이드와 쉬프트를 사용한 데이터 값 예측기

최재혁, 정진하, 윤완오, 신광식, 최상방
인하대학교 전자공학과

Data Value Predictor using Stride and Shift

Jae-Hyeok Choi, Jin-Ha Cheong, Wan-Oh Yoon, Kwang-Sik Shin, and Sang-Bang Choi
Department of Electronic Engineering, Inha University
E-mail : sangbang@inha.ac.kr

Abstract

Conventional stride predictor is useful for predicting data values which vary by a constant value. However, when the data values of shift, multiplication, and division instructions are predicted, the stride predictor can't show the best performance. Thus, we propose predictor using stride and shift to improve predictability. The predictor using stride and shift takes advantage of shift values as well as stride values, so that the overall coverage of prediction increases.

I. 서론

단일프로세서 상에서의 병렬로 여러 명령어를 실행하기 위해 사용하는 명령어 수준 병렬성(Instruction Level Parallelism, ILP) 처리는 현재 컴퓨터 구조에서의 커다란 관심분야이다. 미래의 프로세서에서는 명령어 수준 병렬처리(ILP)를 주로 프로세서 제조에 사용될 것이다.

명령어 수준 병렬성(ILP) 처리에서의 주요 장애는 데이터 종속성(data dependence)에 있다. 만약 어떤 명령어가 먼저 실행된 명령어에 데이터 종속성(data dependence)이 있다면, 먼저 실행되는 명령어의 결과가 나온 이후에만 그 명령어가 실행될 수 있다. 최근 연구에서 데이터 값 예측(data value prediction)을 사용하여 데이터 종속성(data dependence)에 발생하는 장애를 극복할 수 있음을 보여주고 있다 [1]. 이는 명령어의 결과를 과거의 정보를 바탕으로 예측하고, 이 결과에 종속되는 연속적인 명령어들을 계속 실행해 나아가는 것이다. 그 후에 명령어의 실제 결과가 나왔을 때 옳은 결과와 미리 예측한 값을 비교한다. 만약 값이 일치하면, 예측한 결과에 종속되는 액티브 윈도우 안에 있는 모든 명령어들에게 일치했다는 정보를 보낸다. 만약 값이 일치하지 않으면, 옳은 결과를 그 값을 필요로 하는 명령어들에게로 보내고 그 명령어들을 다시 실행시킨다.

본 과제(결과물)는 정보통신부의 정보통신기초기술연구지원사업(정보통신연구진흥원)으로 수행한 연구결과입니다(C1-2003-2000-0198).

II. 데이터 값 예측기의 모델들

일반적으로 데이터 값 예측기는 입력, 테이블 접근, 예측 생성과 같은 순서로 동작하게 된다. 그 뒤에 미래의 예측을 위해 상태 정보를 테이블에 갱신하게 한다. 이 상태 정보에는 최신에 발생했던 값들, 레지스터 값들, 프로그램 카운터 값들, 명령어 영역들, 여러 파이프라인 단계에서의 제어 비트 등을 포함하고 있다. 이 상태 정보의 종류와 조합은 거의 제한이 없다.

데이터 값 예측기에는 예측하는 방법에 따라 2가지 유형이 있다. 하나는 계산에 의한 예측을 하게 되는 예측기이고 또 하나는 전후 관계를 기초로 한 예측기이다 [2].

2.1. 계산에 의한 예측기

계산에 의한 예측기(Computational Predictors)는 명령어에 의해 발생한 이전 값으로 어떠한 연산을 수행해서 앞으로 이 명령어를 수행했을 때 발생하게 될 데이터 값을 예측하는 것이다.

(1) 최근값 예측기

최근값 예측기(Last Value Predictor)는 바로 전의 값을 그대로 사용하는 예측기이다. 예를 들어 7, 7, 7, 7, …와 같이 변화 없이 반복되는 데이터가 발생할 경우 유용한 예측기가 된다 [2][3].

예측된 값을 p_value라고 하고 갱신할 값을 u_value라고 하고, 바로 전의 값을 last_value에 기억하는 2^n개의 테이블로 구성되어 있는 최근값 예측기의 동작을 다음과 같이 나타낼 수 있다.

```
Predictor : {tag, last_value} LV[2^n]
Predict :  p_value = LV[index(PC)].last_value
Update : LV[index(PC)].last_value = u_value
```

(2) 스트라이드 예측기

스트라이드 예측기(Stride Predictor)는 가장 최근에 발생한 값에 최근 발생한 두 개의 값의 차를 더해서 다음 값을 예측하는 예측기이다. 스트라이드 예측기는 가장 최근에 발생한 값과 스트라이드라고 불리는 가장 최근의 값과 두 번째로 최근 값의 차이 값을 저장하고 있다. 예를 들어 2, 4, 6, 8, 10, …와 같이 고정된 값

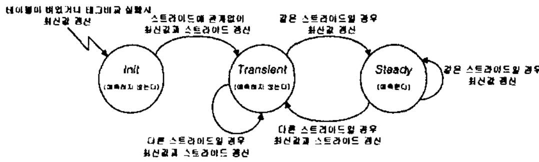

그림 1. 스트라이드 예측기의 상태 변화.

(스트라이드)를 가지고 데이터가 발생할 경우에 유용한 예측기가 된다. 그리고 스트라이드 예측기는 최근값 예측기를 포함하게 된다. 스트라이드 값이 0일 경우에 최근값 예측기와 동일한 동작을 하게 된다.

예측률을 높이기 위해 고정된 스트라이드 값에 대해서만 예측을 하기 위해서 초기(Init), 과도기(Transient), 안정기(Steady) 3상태를 두어 안정기 상태에서만 예측을 하게 된다 [2][3].

최근값 예측기와 동일한 값들을 기억하고 있고 추가적으로 스트라이드 값을 stride에 기억하고 상태를 state에 기억하는 2^n개의 테이블로 구성되어 있는 스트라이드 예측기의 동작을 다음과 같이 나타낼 수 있다.

```
<Predictor>
{tag, state, last_value, stride} St[2^n]
<Predict>
if (St[index(PC)].state == Steady)
  p_value = St[index(PC)].last_value + St[index(PC)].stride
else don't predict
<Update>
if (St[index(PC)].state == Empty)
{
  St[index(PC)].last_value = u_value
  St[index(PC)].state = Init
}
else if (St[index(PC)].state == Init)
{
  St[index(PC)].stride = u_value - St[index(PC)].last_value
  St[index(PC)].last_value = u_value
  St[index(PC)].state = Transient
}
else if (St[index(PC)].state == Transient)
{
  temp = u_value - St[index(PC)].last_value
  St[index(PC)].last_value = u_value
  if (St[index(PC)].stride == temp)
    St[index(PC)].state = Steady
  else
    St[index(PC)].stride = temp
}
else if (St[index(PC)].state == Steady)
{
  temp = u_value - St[index(PC)].last_value
  St[index(PC)].last_value = u_value
  if (St[index(PC)].stride != temp)
  {
    St[index(PC)].stride = temp
    St[index(PC)].state = Transient
  }
}
```

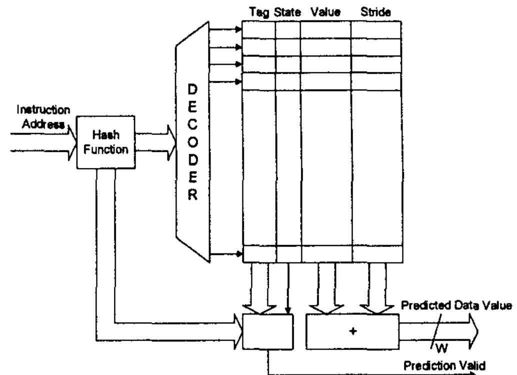

그림 2. 스트라이드 예측기.

2.2 전후 관계를 기초로 하는 예측기

전후 관계를 기초로 하는 예측기(Context Based Predictors)는 제한된 순서의 전의 값의 특별한 전후 관계를 이용해서 예측하게 된다. 예를 들어 5, 13, 7, 5, 13, 7, …와 같이 일정한 값(스트라이드)으로는 변하지는 않지만, 일정한 형을 가지고 변하는 데이터 값을 예측하는데 유용하다.

(1) FCM 예측기

FCM 예측기(Finite Context Method Predictor)는 한정된 수의 이전 값을 기초로 해서 다음 값을 예측한다. p순번 FCM 예측기는 p개 이전 값을 사용한다. FCM 예측기는는 2단계로 구성되어 있다. 첫번째 테이블(Value History Table)에는 이전에 발생했던 값들과 그 값들의 p개의 전후 관계를 기억하고 있고 두 번째 테이블(Pattern History Table)은 어떤 전후 관계를 따르게 되는 특정한 상태가 발생하는 것을 세는 계수기로 구성되어 있다. 이와 같이 각각의 전후 관계에 대한 값을 세기 위한 많은 계수기가 있어야만 하고 최대의 값을 기억하고 있는 계수기가 가리키는 값이 예측하는 값이 된다.

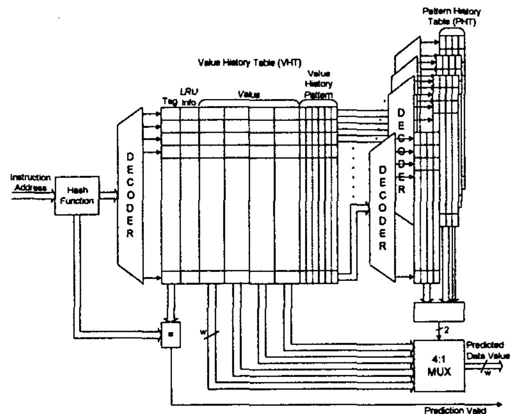

그림 3. FCM 예측기.

실제의 실행에서는 정확한 계수기 값을 가지고 있는 것은 불가능하므로 작은 크기의 계수기가 사용된다. 작은 크기의 계수기가 최대의 계수기 값에 도달하게 되면, 같은 전후 관계를 위한 계수기들 모두를 반으로 고쳐 놓는다. 작은 크기의 계수기는 전체의 내력보다는 최근의 내력에 더 큰 가중치를 주게 되는 장점이 있다 [2].

(2) 2단계 예측기

2단계 예측기(Two-Level Predictor)는 FCM 예측기와 비슷하게 2단계로 구성되어 있다. FCM은 PHT이 각각의 엔트리마다 각각의 PHT가 존재하나 2단계 예측기는 공통된 PHT를 사용하므로 작은 크기의 PHT를 가지고도 데이터 값을 예측할 수 있는 예측기이다 [3].

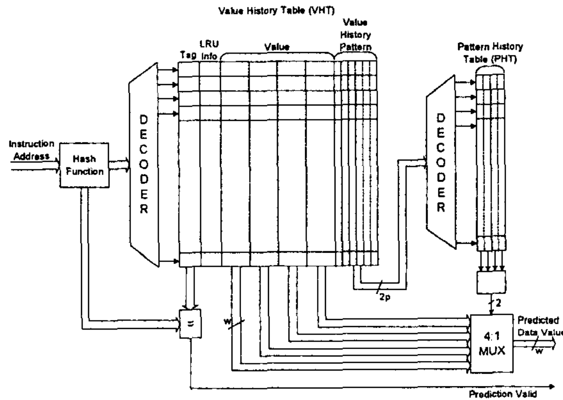

그림 4. 2단계 예측기.

Ⅲ. 스트라이드와 쉬프트를 사용한 예측기

기존의 스트라이드 예측기는 고정된 값(스트라이드)으로 증가나 감소할 때는 예측할 수 있으나 쉬프트 연산이나 곱셈이나 나눗셈 연산에서는 예측하기가 어려웠다. 이를 개선해서 스트라이드와 쉬프트를 사용한 예측기는 기존의 스트라이드 예측기와 동일한 동작을 하지만 과도기 상태에서 스트라이드 값을 이용해 최근값과 더한 값을 사용하지 않은 대신에 이때 쉬프트에 값이 있을 경우 쉬프트 값을 가지고 최근값에 쉬프트 연산을 통해서 나온 값으로 예측을 하도록 한다. 그러기 위해서는 쉬프트 값을 찾기 위한 동작을 더 추가해야 한다.

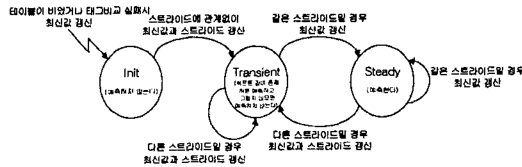

그림 5. 스트라이드와 쉬프트를 사용한 예측기의 상태 변화.

스트라이드 예측기와 동일한 값들을 기억하고 있고 추가적으로 쉬프트 값을 shift에 기억하는 2^n개의 테이블로 구성되어 있는 스트라이드 쉬프트 예측기의 동작을 다음과 같이 나타낼 수 있다. 스트라이드와 쉬프트를 사용한 예측기에서 추가된 동작들은 이탤릭체로 나타냈다.

```
<Predictor>
{tag, state, last_value, stride, shift} Sh[2^n]
<Predict>
if (Sh[index(PC)].state == Steady)
  p_value = Sh[index(PC)].last_value + Sh[index(PC)].stride
else if (Sh[index(PC)].state == Transient)
{
  if (Sh[index(PC).shift != 0)
    p_value = Sh[index(PC)].last_value << Sh[index(PC)].shift
}
else don't predict
<Update>
if (Sh[index(PC)].state == Empty)
{
  Sh[index(PC)].last_value = u_value
  Sh[index(PC)].state = Init
}
else if (Sh[index(PC)].state == Init)
{
  Sh[index(PC)].stride = u_value - Sh[index(PC)].last_value
  Sh[index(PC)].shift = find_shift(Sh[index(PC)].last_value, u_value)
  Sh[index(PC)].last_value = u_value
  Sh[index(PC)].state = Transient
}
else if (Sh[index(PC)].state == Transient)
{
  temp = u_value - Sh[index(PC)].last_value
  Sh[index(PC)].shift = find_shift(Sh[index(PC)].last_value, u_value)
  Sh[index(PC)].last_value = u_value
  if (Sh[index(PC)].stride == temp)
    Sh[index(PC)].state = Steady
  else
    Sh[index(PC)].stride = temp
}
else if (Sh[index(PC)].state == Steady)
{
  temp = u_value - Sh[index(PC)].last_value
  Sh[index(PC)].shift = find_shift(Sh[index(PC)].last_value, u_value)
  Sh[index(PC)].last_value = u_value
  if (Sh[index(PC)].stride != temp)
  {
    Sh[index(PC)].stride = temp
    Sh[index(PC)].state = Transient
  }
}
```

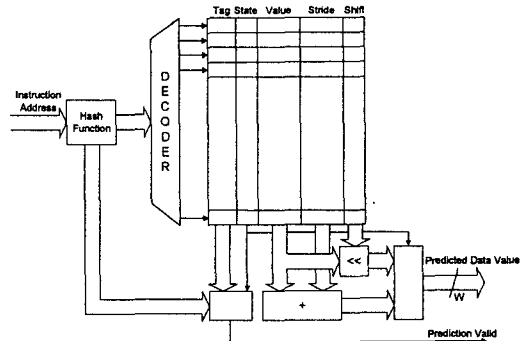

그림 6. 스트라이드와 쉬프트를 사용한 예측기.

표 6. 시뮬레이션 환경

Predictor	Index Bit	Tag	State or LRUinfo	Value	Stride	Shift	History	PHT	Threshold	Table Size
Stride	10	22bit	State 2bit	32bit	32bit	N/A	N/A	N/A	N/A	88Kbit
Stride & Shift	10	22bit	State 2bit	32bit	32bit	4bit	N/A	N/A	N/A	92Kbit
FCM	10	22bit	LRU 8bit	4*32bit	N/A	N/A	2*6	$2^{10}*2^{12}**3*4$	4	49300Kbit
2 Level	10	22bit	LRU 8bit	4*32bit	N/A	N/A	2*6	$2^{12}*3*4$	4	196Kbit

Ⅳ. 모의 시뮬레이션 및 성능분석

스트라이드와 쉬프트를 사용한 예측기의 성능을 측정하기 위해서 SimpleScalar 3.0 툴셋에 벤치마크 프로그램으로 SPEC95를 사용하여 시뮬레이션을 하고 성능을 측정하였다.

스트라이드 예측기, 스트라이드와 쉬프트를 사용한 예측기, FCM 예측기, 2단계 예측기에 대해서 측정하였다. 각 예측기에 대한 환경조건은 표1에 나타나 있다. 모든 예측기에는 동일하게 인덱스 비트로서 10을 사용하여 1024개의 엔트리를 가지고 있도록 했다. 최근값과 스트라이드 값은 모두 동일하게 32비트를 사용했고, 상태를 저장하기 위한 상태비트는 2비트를 사용했다. 스트라이드와 쉬프트를 사용한 예측기에는 쉬프트 값을 기억하기 위해서 4비트를 사용했다. FCM과 2단계 예측기는 동일하게 최근값 4개를 기억하게 했고, 동일하게 6개의 내력을 기억하도록 했다. PHT에는 3비트 계수기 4개를 사용했으며, 문턱값으로 4를 사용해서 시뮬레이션을 하였다.

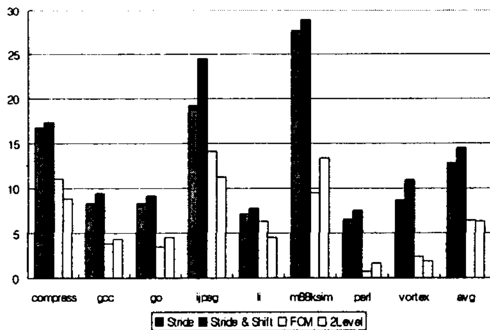

그림 7. 각 데이터 값 예측기들의 예측률.

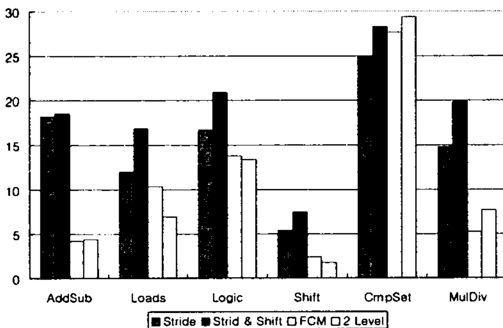

그림 8. 명령어별 데이터 값 예측기들의 예측률.

그림 7의 시뮬레이션 결과를 보면 전체적으로 예측률에서는 스트라이드 예측기와 스트라이드와 쉬프트를 사용한 예측기가 FCM 예측기와 2단계 예측기보다 좋은 성능이 나오는 것을 볼 수 있다. 스트라이드와 쉬프트를 사용한 예측기는 기존의 스트라이드 예측기보다 전체적으로 12.8%에서 14.4%로 1.6% 더 예측률이 높아짐을 볼 수 있다.

그림 8을 보고 명령어별로 향상한 것을 살펴보면 AddSub에서는 18.2%에서 18.6%로 작은 성능향상을 보이나 Loads에서는 12.0%에서 16.9%로, Logic에서는 16.7%에서 20.9%로, Shift에서는 5.4%에서 7.4%로, CmpSet에서는 24.9%에서 28.3%로, MulDiv에서는 14.8%에서 19.8%로 눈에 띄게 향상되었을 볼 수 있다. Lui는 명령어 읽을 값이 포함되어 있으므로 스트라이드 예측기나 스트라이드와 쉬프트를 사용한 예측기에 상관없이 동일한 결과가 나오는 것을 볼 수 있다.

테이블 크기를 구해보면 스트라이드 예측기는 88Kbit, 스트라이드와 쉬프트를 사용한 예측기는 92Kbit, FCM 예측기는 49300Kbit, 2단계 예측기는 196Kbit이다. 테이블 크기를 비교해보면 1대 1.045대 560.23대 2.227의 비율임을 알 수 있다.

Ⅴ. 결론

본 논문에서는 기존의 스트라이드 예측기에 쉬프트 값을 찾는 부분과 4.5%정도의 테이블의 크기 증가를 통해서 스트라이드와 쉬프트를 사용한 예측기를 제안하고, 실험을 통해서 스트라이드와 쉬프트를 사용한 예측기가 스트라이드 예측기보다 12.8%에서 14.4%로 1.6%의 성능향상을 보였다.

참고문헌

[1] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Exceeding the Dataflow Limit via Value Prediction," Proceedings of 29th International Symposium on Microarchitecture (MICRO-29), pp. 226-237 1996

[2] Y. Sazeides and J. E. Smith, "The Predictability of Data Values," Proceeding of the 30th Annual ACM/IEEE International Symposium on Microcrchitecture pp. 248-258 1997.

[3] K. Wang and M. Franklin, "Highly Accurate Data Value Predictions using Hybrid Predictor," Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, pp. 281-290 1997.