Performance Monitor Counter를 이용한 Intel Processor의 Branch Target Buffer 구조 탐구

정주혜, 김한이, 서태원
고려대학교 반도체시스템공학과
E-mail: 88juc@korea.ac.kr, suhtw@korea.ac.kr

Exploring Branch Target Buffer Architecture on Intel Processors with Performance Monitor Counter

Juhye Jeong, Han-Yee Kim, Taeweon Suh
Dept of Semiconductor System Engineering, Korea University

요 약

Meltdown, Spectre 등 하드웨어의 취약점을 이용하는 side-channel 공격이 주목을 받으면서 주요 microarchitecture 구조에 대한 철저한 이해의 필요성이 커지고 있다. 현대 마이크로프로세서에서 branch prediction이 갖는 중요성에도 불구하고 세부적인 사항은 거의 알려지지 않았으며 잠재적 공격에 대비하기 위해서는 반드시 현재 드러난 정보 이상의 detail을 탐구하기 위한 시도가 필요하다.

본 연구에서는 Performance Monitor Counter를 이용해 branch 명령어를 포함한 프로그램이 실행되는 동안 Branch Prediction Unit에 의한 misprediction 이벤트가 발생하는 횟수를 체크하여 인텔 하스웰, 스카이레이크에서 사용되는 branch target buffer의 구조를 파악하기 위한 실험을 수행하였다. 연구를 통해 해당 프로세서의 BTB의 size, number of way를 추정할 수 있었다.

1. 서 론

하드웨어의 구조적 취약점을 발견하여 이를 이용해 중요한 정보를 유출하는 소프트웨어 기반 side-channel 공격[1]에 관한 연구들이 주목을 받으면서 microarchitecture의 구조에 관한 연구의 중요성이 커지고 있다. 이러한 공격은 공격 대상인 하드웨어의 구조 및 동작 원리 등의 세부적인 정보에 대한 철저한 이해를 기반으로 하므로[2], 공격 방식을 이해하고 나아가 대응책을 마련하기 위해서는 먼저 하드웨어의 구조에 대한 이해를 공개된 정보 이상으로 넓히기 위한 작업이 선행되어야 한다.

현대 마이크로프로세서에서 branch prediction은 매우 중요한 부분을 차지하며, branch predictor를 조작해 정보를 유출하는 Spectre를 비롯해 branch prediction history[2, 3] 및 branch target buffer[4, 5]를 이용한 side-channel 공격들이 다양하게 연구되고 있다[1]. 그러나 이에 비해 branch prediction이 실질적으로 어떻게 구현되었는지에 관한 정보는 거의 알려져 있지 않다. 인텔 등 주요 프로세서 제조사에서는 자사 신형 제품들에 사용되는 branch prediction 방식에 대한 정보를 거의 밝히지 않고 있으며, 매뉴얼에도 단편적인 설명만 제공한다. 이러한 정보의 미비는 잠재적 공격의 위험성을 강화할 뿐이므로 branch predictor에 관한 관심과 연구가 시급하다.

본 연구에서는 branch prediction unit 중 branch target buffer의 size, way를 알아내기 위한 실험을 수행하였다. 2번 항목에서는 연구의 배경 지식을 설명하였고 3번 항목에서는 실험 환경 및 결과를 기술하였다.

이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임
(No.2019-0-00533, 컴퓨터 프로세서의 구조적 보안 취약점 검증 및 공격 탐지대응)

2. 배경 지식

2.1 Branch Prediction

Branch prediction은 branch 결과와 target 어드레스를 계산하는 동안 생성되는 파이프라인 stall을 줄이기 위해 계산 결과가 나오기 전에 다음 진행 상황을 예측하고 미리 실행하는 기법이다. 예측이 실패했을 경우 파이프라인을 flush하고 진행 상황을 롤백해야 하는 페널티가 있지만 branch prediction을 통해 전체 throughput을 크게 향상시킬 수 있기 때문에 이는 마이크로프로세서의 핵심적인 피처로 자리 잡았다[5].

프로세서 내에서 branch prediction을 수행하는 Branch Prediction Unit(BPU)은 과거의 기록을 기반으로 branch 결과를 예측하는 predictor와 target 어드레스를 보관하는 Branch Target Buffer로 이루어진다[2].

2.2 Branch Predictions in Intel Processors

인텔의 공식 매뉴얼에서는 BPU의 구조 및 어떻게

prediction이 이루어지는지 등에 관해서 간단하게만 기술하고 있기 때문에 자세한 사항은 연구를 통해 밝혀진 내용에 의존해야 한다. 펜티엄 P6 architecture를 reverse engineering 기법으로 조사한 연구[6]에 따르면 P6의 BPB는 128 set에 4-way의 캐쉬 구조(512 entries)로 구성되어 있으며, BTB index 지정에 instruction pointer의 어드레스 비트[10:4]가 사용된다. 인텔 patent[7]에 따르면 BTB tag는 9비트 길이이며, 그중 7비트는 instruction pointer의 어드레스 비트[29:11]를 압축하여 만들어진다. BTB의 1 set은 4개의 branch entry와 replacement policy bits(LRR bits), branch pattern set table으로 구성되고 각 entry는 해시 태그(9비트), block offset(4비트), branch type, valid 비트, target address(32비트) 등을 포함한다.
인텔 NetBurst architecture의 BTB는 4-way 구조의 1024 sets(4096 entries)로 구성되며 instruction pointer의 어드레스 비트[13:4]가 index 지정에 사용된다[6].

2.3 Assumptions

이상의 정보에 근거해 본 연구에서는 하스웰, 스카이레이크의 BTB가 최소 512개 이상 entry를 가진 4-way 구조의 캐쉬 형태로 이루어져 있으며, instruction pointer의 어드레스 하위 비트를 인덱스 및 태그 지정에 사용한다고 가정한다.

3. 실험 환경 및 결과

3.1 실험 방식

본 연구에서는 인텔 프로세서에서 지원하는 Performance Monitor Counter(PMC)를 활용하였다. 실험 프로그램은 리눅스 환경에서 구현되었으며, 커널 모드 드라이버를 통해 유저 레벨에서 프로세서 칩 내부의 PMC에 접근할 수 있다. PMC의 선택에 따라 실행된 명령어의 개수, 캐쉬 miss 발생 횟수, branch 명령을 잘못 예측한 횟수 등 프로세서 내부의 다양한 이벤트를 관찰할 수 있다. 본 연구에서는 특히 BACLEARS.ANY 카운터에 주목하였다.

Event Num.	Umask Value	Event Mask Mnemonic	Description
E6H	01H	BACLEARS. ANY	Number of front end re-steers due to BPU misprediction.

(그림 1) BACLEARS.ANY PMC description[8]

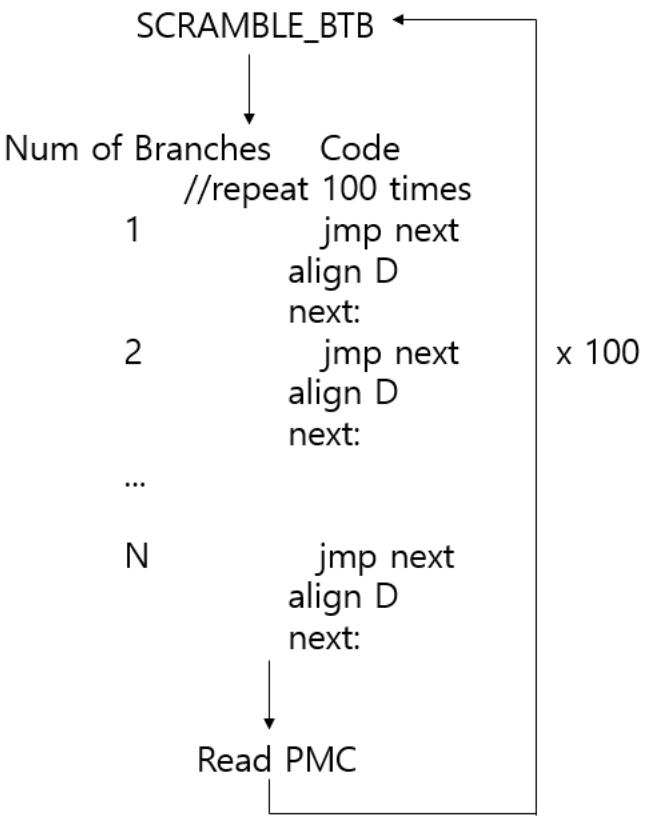
branch target buffer가 정확한 예측에 실패했을 경우 BTB의 내용을 업데이트 하기 위해 front-end에서 resteer가 발생하며, BACLEARS.ANY 카운터는 resteer의 횟수를 기록한다. 처리해야 할 branch 명령의 수가 BTB의 용량을 초과하는 capacity miss나 branch들이 BTB 내의 같은 위치를 놓고 경쟁하는 conflict miss가 발생할 경우 BTB가 예측에 실패할 가능성이 매우 높다. branch 명령의 개수와 branch 명령어 사이의 간격을 조절하며 위와

```
%macro OneJump 0
jmp %%next
align {align}                    //align value(D)
%%next:
%endmacro

jmp BtbLoop
align 4 * 1024 * 1024; align to a 4MB boundary
BtbLoop:
%rep {num_branches}    // num of branches(N)
OneJump
%endrep

%rep 64
nop
%endrep
```

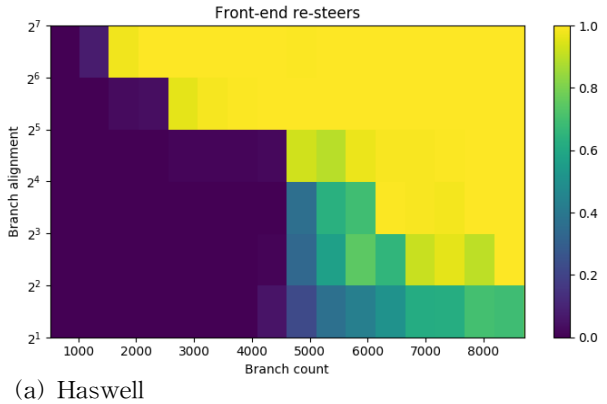
(그림 2) test code 예제



(그림 3) test algorithm

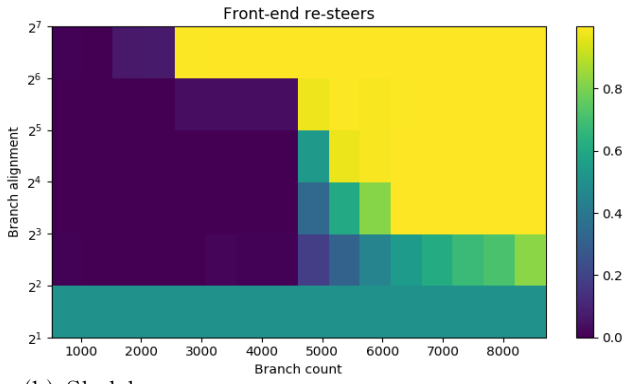
같은 miss가 발생하는 조건을 찾을 수 있다면 이를 토대로 BTB의 구조를 추측할 수 있을 것이다.
BTB의 size와 way를 구하기 위해 [9, 10]과 같은 방식으로 총 2번의 실험을 수행하였다. test code는 N개의 unconditional branch(2 byte length)를 각각 D 값으로 align(NOP is default for align)[11]한다. 정해진 횟수(100번)만큼 실행하며 얻은 BACLEARS.ANY 카운터 값 중

(그림 4) BTB size test results



(a) Haswell

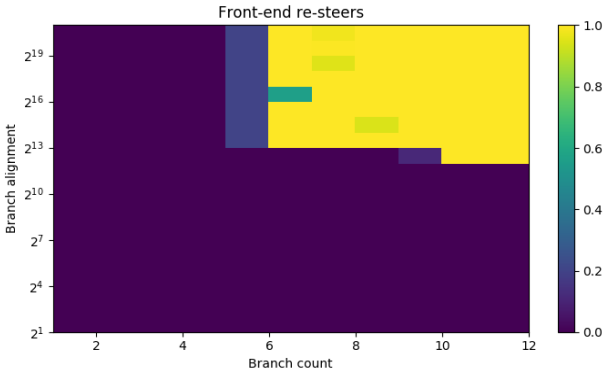
[실험 환경: Intel i7-4700, Ubuntu 16.04, single-core]



(b) Skylake

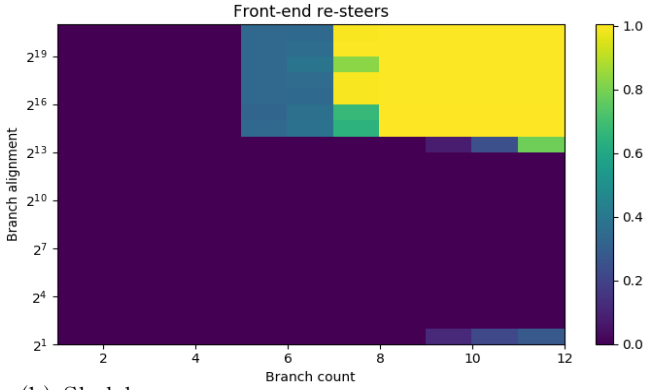
[실험 환경: Intel i7-6700, Ubuntu 18.04, single-core]

(그림 5) BTB way test results



(a) Haswell

[실험 환경: Intel i7-4700, Ubuntu 16.04, single-core]



(b) Skylake

[실험 환경: Intel i7-6700, Ubuntu 18.04, single-core]

가장 작은 값을 선택하는 방식으로 각 실험마다 N과 D값에 따른 BTB 예측 miss 확률(resteer 횟수/N)을 구하였다. 테스트를 반복해서 수행하는 동안 BTB에 저장된 정보 때문에 miss prediction 확률이 낮아지는 문제를 해결하기 위해 SCRAMBLE_BTB 코드를 이용하였다. 해당 코드는 임의의 점프 명령을 반복적으로 수행하여 BTB에 남아있는 정보를 모두 제거한다. 테스트를 한 번 수행한 후 항상 SCRAMBLE_BTB를 실행함으로써 매번 일정한 예측 miss 확률을 유지할 수 있다.

3.2 BTB size

BTB의 size보다 많은 branch 명령어가 포함된 코드를 실행할 경우 capacity miss로 인한 resteer횟수가 매우 높게 나올 것이다. 해당 조건을 구현하기 위해 branch 명령의 개수 N의 범위는 512부터 8704까지 넓게 잡았고, align 간격 D는 상대적으로 적은 2~2^7 사이의 값으로 하였다.
하스웰의 경우 D = 2부터 D = 2^4 사이에서 최대 N = 4096까지 0~5% 가량의 낮은 resteer율을 보였으며, 4096보다 큰 N 구간에서는 resteer확률이 급속도로 상승하여 4096을 기점으로 명백한 차이를 보였다. 이로 보아 하스웰의 BTB는 4096개의 entry들을 보유하고 있으며 그 이상의 branch 명령이 들어올 경우 용량 초과로 capacity miss가 발생하는 것으로 여겨진다. D = 2^5, 2^6의 경우 branch 개수 N의 값이 조금 커지면 바로 resteer 확률이 급속도로 하락하는 모습을 보여 branch 간의 간격이 클수록 BTB의 예측 가능성이 낮아지는 것을 알 수 있었다. 정렬 간격이 좁을 때에는 어드레스 하위 비트에 위치한 인덱스 비트 branch들이 BTB 내의 다수의 셋에 저장되었지만 간격이 넓어지면서 BTB 내의 같은 셋을 놓고 경쟁하는 conflict miss가 발생했기 때문인 것으로 보인다.
스카이레이크의 경우 대체로 하스웰과 비슷한 모습을 보이지만 D = 2 구간은 N의 크기와 상관없이 일관적으로 50%가량의 resteer확률을 유지했으며, D = 2^5 구간에서는 N = 2560부터 4096까지 3%로, D = 2^6 구간에서는 N = 1536에서 2048까지 6%가량의 확률로 낮아져 하스웰보다 BTB 성능의 향상이 있었음을 알 수 있다.

3.3 BTB Way

만일 branch 명령들이 BTB의 동일한 세트 내에 계속 배치된다면, 세트의 크기를 초과하여 배정된 branch는 conflict miss를 유발할 것이다. branch의 어드레스에서 인덱스 위치를 결정하는 구간 밖의 비트만 계속 변화한다면

branch들은 모두 BTB의 같은 세트를 놓고 경쟁하게 된다. 만일 4-way라면 5번째의 branch 명령이 들어오는 순간 eviction이 발생할 것이고, evict된 branch가 다음에 사용된다면 prediction miss가 발생할 것이다. 이러한 조건을 만들기 위해 첫 번째 실험보다 branch 명령의 개수 N은 1~12로 적게 잡은 대신 align 간격 D는 어드레스의 상위 비트까지 영향을 미치도록 2^21까지 넓게 설정하여 같은 셋에 배정될 확률을 높였다. 이는 인덱스 비트들이 어드레스의 하위 비트에 위치할 것이라는 가정에 근거하였다.

하스웰의 경우 N = 4까지는 모든 D 값에 대해 0%의 resteer 확률을, N=5에서 20% 정도로 낮은 예측 miss 확률을 보였다. 기본적으로 4-웨이로 구성되었고 5-웨이에 가까운 성능을 낼 정도로 효율적인 eviction 방식을 사용하는 것으로 보인다.

D = 2^11 이하에서는 모든 D값에 대해 resteer가 거의 발생하지 않았는데 이 경우 branch 명령들이 서로 다른 셋에 배정되었기 때문인 것으로 추측할 수 있고, 따라서 BTB의 셋을 결정하는 인덱스 비트들은 어드레스의 [11:2] 구간 내에서 결정될 것이다.

스카이레이크의 경우 하스웰과 마찬가지로 N = 4까지 항상 0%의 확률을 나타냈다. N = 5일 시엔 33.4%로 다소 증가하였지만 N = 6의 경우 100%의 resteer 확률을 보인 하스웰과 달리 약 34.2 %로 N = 5일 때와 비슷한 확률로 더 우수한 성능을 보였다. 또한 모든 N 값에 대해 0%의 resteer확률을 보이는 구간도 D = 2^13까지로 다소 증가하였다.

4. 결론 및 향후 연구 계획

본 연구에서는 performance monitor counter를 이용해 인텔 하스웰, 스카이레이크에서 사용하는 branch target buffer의 구조를 탐구하는 실험을 수행하였다. 실험 결과 두 프로세서 모두 4-way의 총 4096개 entries로 구성된 BTB를 가지고 있다는 것을 확인할 수 있었다. 현재 BTB의 index와 tag를 결정하는 방식을 알아내기 위한 연구를 진행 중이며 향후 전체 branch predictor unit의 구조 파악 및 구조적 취약점을 찾기 위한 추가 연구를 진행할 예정이다.

참고문헌

[1] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., M., MANGARD, S.,PRESCHER, T., SCHWARZ, M., & Yarom, Y. (2018). Spectre attacks: Exploiting speculative execution. arXiv preprint arXiv:1801.01203.

[2] O. Acıı"Ccmez, "C C. Kaya Ko"Cc, and J.P. Seifert. Predicting secret keys via branch prediction. RSA Conference Cryptographers Track - CT-RSA '07, LNCS vol. 4377, pp. 225 - 242, Springer, 2007.

[3] O. Acıic¸mez, S. Gueron, and J.-P. Seifert, "New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures," in International Conference on Cryptography and Coding (IMA), 2007.

[4] D. Evtyushkin, D. V. Ponomarev, and N. B. Abu-Ghazaleh, "Jump over ASLR: Attacking branch predictors to bypass ASLR," in MICRO, 2016.

[5] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing," in USENIX Security Symposium, 2017.

[6] M. Milenkovic, et al., "Microbenchmarks for determining branch predictor organization," in Software Practice and Experience. vol. 34, 2004, pp. 465-487.

[7] B. D. Hoyt, et al., "Method and Apparatus for Implementing a Set-Associative Branch Target Buffer," U.S. Patent 5574871, Intel Corporation, 1996.

[8] INTEL. Intel 64 and ia-32 architectures software developer's manual combined volumes: 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c and 3d, October. 2016.

[9] The BTB in contemporary Intel chips—Matt Godbolt's blog. http://xania.org/201602/bpu-part-three.

[10] UZELAC, V., AND MILENKOVIC, A. Experiment flows and microbenchmarks for reverse engineering of branch predictor structures. In Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on (2009), IEEE, pp. 207 - 217.

[11] https://nasm.us/doc/nasmdoc4.html#section-4.11.13