

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

KVN Prediktor Větvení

**Dominik Liberda
Moravskoslezský kraj**

Sedliště 2020

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

KVN Prediktor Větvení

KVN Branch Predictor

Autoři: Dominik Liberda

Škola: Střední průmyslová škola elektrotechniky a informatiky,
Kratochvílova 7/1490, 702 00 Ostrava – Moravská Ostrava

Kraj: Moravskoslezský kraj

Konzultant: Ing. Pavlína Pavlová

Sedliště 2020

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Sedlístích dne 24. 3. 2020

Dominik Liberda

Poděkování

Děkuji celé mé rodině, která mě v mé práci podporovala. Dále bych chtěl poděkovat moji třídní učitelce Pavlíně Pavlové, která mě v práci neustále podporovala a zároveň se svým popoháněním zasloužila o to, že jsem tuto práci vůbec zvládl dokončit včas. Zmínit si zaslouží i mí spolužáci a přátelé, na které jsem se vždy mohl obrátit.

Velké díky patří i Pierrovi Michaudovi, který mi dovolil použít jeden z jeho grafů a pomohl mi objasnit nesrovnalosti mezi mými a jeho výsledky.

Anotace

Tato práce se zabývá vývojem prediktoru větvení KVN a jeho simulací. Prediktor větvení je digitální obvod přítomný ve všech moderních procesorech. Jeho prací je predikovat, zda budou podmínky (větve) přijaty nebo odmítnuty. Tímto umožňuje procesorovému jádru pracovat „napřed“.

Cílem práce je vyvinout prediktor větvení, který bude schopen vyrovnat se prediktorům z období okolo roku 2001.

Klíčová slova

Predikce větvení; simulace; CPU; větev; podmínka; instrukce;

Annotation

This project is focused on development of the KVN branch predictor and it's simulation. Branch predictor is a digital circuit present in all modern processors. It's work is to predicate whether a condition (branch) will be taken or not taken. Thanks to this prediction is CPU core able to work „ahead“.

Goal of this project is to develop a branch predictor which will be able to compete with predictors from around year 2001.

Keywords

Branch prediction; simulation; CPU; branch; condition; instruction;

Obsah

1. Úvod.....	7
2. Úvod do predikce větvení	9
2.1 Jak to vypadá v praxi	10
3. Metodika.....	13
3.1 Jak vypadají simulace.....	14
3.2 Bity a Byty.....	15
4. Seznámení se základními prediktory	16
4.1 Statická predikce	16
4.2 SW metody predikce	17
4.3 Dynamická predikce	18
4.4 Bimodal predictor.....	19
4.5 Two level adaptive branch predictor	20
4.6 Gshare.....	24
5. De-aliased prediktory	28
6. Moderní prediktory.....	32
7. KVN Prediktor.....	35
7.1 Confidence level.....	37
7.2 META prediktor	38
7.3 Nastavení historie	39
7.4 Aktualizační politika.....	40
7.5 Výsledky	40
8. Jak prediktor plánuji vylepšit v budoucnu.....	43
8.1 Jak plánuji v práci pokračovat	43
9. Závěr.....	44
10. Použitá literatura	46
11. Seznam obrázků, tabulek, schémat a grafů.....	48
11.1 Seznam obrázků.....	48
11.2 Seznam Tabulek.....	48
11.3 Seznam Grafů.....	48
11.4 Seznam schémat.....	48
12. Seznam pojmů a zkratk.....	49
13. Seznam příloh	51
14. Dodatek A – legenda k přílohám č. 2 a č. 3	52

1. Úvod

Snad skoro každý má doma počítač nebo mobilní telefon, používáme je dnes a denně. Mozkem každého počítače nebo telefonu je procesor, to ví asi každý, ale už jen málo lidí ví, jak procesory skutečně fungují. Dnešní procesory jsou neuvěřitelně komplikované, skládají se z miliard tranzistorů – v současné době má nejpokročilejší serverový procesor 39.54 miliardy tranzistorů¹, a každým rokem toto číslo roste.

Možná jste někdy slyšeli pojem „Von Neumannova architektura“², jedná se o model fungování počítače, který v roce 1945 představil matematik John von Neumann. Ačkoliv z velké části je jeho model stále aktuální, procesory ho porušují již někdy od roku 1970. Jeden ze základních kamenů Von Neumannovy architektury je, že v jednu chvíli může procesor pracovat pouze na jedné instrukci. Systém může začít zpracovávat další instrukci až potom, co bylo dokončeno zpracování současné instrukce. Stále mě překvapuje, kolik lidí si myslí, že toto je stále platné.

Inženýři si už dávno uvědomili, že pokud chtějí postavit rychlé procesory, musí zpracovávat několik instrukcí najednou, neboli paralelně. Každé jádro moderního procesoru je schopné najednou pracovat na 100-300 instrukcích najednou, to je hodně vzdálené od Von Neumannovy jediné instrukce.

Aby byly jádra procesoru schopna pracovat na tolika instrukcích zároveň, obsahují mnoho komponent, které „krmí“ procesor instrukcemi. Jedná se o různé buffery, dekodéry, paměti cache, prefetchery a v neposlední řadě také o prediktor větvení.

Prediktor větvení je speciální obvod, který se snaží odhadnout (predikovat), zda budou podmínky/větve přijaty (anglicky Taken), nebo odmítnuty (anglicky Not Taken). Nejmodernější prediktory větvení dosahují přesnosti 99+ %.

Jak už asi můžete tušit, zamiloval jsem se do digitálních obvodů. Dlouhou dobu jsem hledal způsob, jak bych si mohl vyzkoušet vývoj vlastního procesoru. Pokoušel jsem se navrhnout vlastní MIPS procesor, ale nakonec jsem to vzdal, protože jsem neměl trpělivost psát pro něj testovací kód v assembly a ručně jej pak překládat do binární a následně do hexadecimální soustavy. Nakonec jsem narazil na predikci větvení [21, 22, 23]. Toto téma se mi okamžitě zalíbilo a začal jsem číst vědecké práce [1, 2, 3] zabývající se vývojem prediktorů větvení. Bohužel tyto práce měly jeden nešvar – každý testoval svůj vlastní prediktor, ale už se nikdo neobtěžoval porovnat své výsledky s cizími prediktory. Rozhodl jsem se, že místo toho, abych se spoléhal na to, že někdo za mě tyto prediktory otestuje, tak si je postavím a otestuji sám.

¹ AMD EPYC 2, https://en.wikipedia.org/wiki/Transistor_count

² https://en.wikipedia.org/wiki/Von_Neumann_architecture

Následující měsíc jsem strávil hledáním způsobu, jak otestovat jednotlivé prediktory. Zvažoval jsem postavit prediktory z logických hradel a pak provést simulaci, jenže jsem si brzo uvědomil, že abych mohl simulaci provést, potřebuju vstupní data – chování reálného programu, který má běžet na procesoru.

Nespočet nocí jsem proseděl u počítače a „Googloval“ mé možnosti. Když už jsem to chtěl vzdát, objevil jsem soutěž jménem CBP 2016³ – Championship Branch Prediction (Šampionát v predikci větvení). Nejednalo se o žádnou středoškolskou soutěž, to ani zdaleka. Jedná se o malou soutěž pořádanou společností Samsung (předchozí ročníky pořádala společnost Intel), kde se hlásí ti nejlepší inženýři světa – mnozí z nich psali vědecké práce o prediktorech větvení v době, kdy jsem nebyl ani na světě. V době, kdy jsem na tuto soutěž narazil, byla už dva roky ukončena, ale i tak mi nic nebránilo stáhnout si soutěžní infrastrukturu pro testování prediktorů. Bylo to přesně to, co jsem hledal.

Jelikož nikdo nepočítal s tím, že se o soutěž bude zajímat takový začátečník, dokumentace k infrastruktuře byla na můj vkus dost strohá. Doteď nevím, jakým zázrakem se mi ji podařilo zprovoznit. Byla to pro mě noc plná zoufalství.

Infrastruktura soutěže CBP je dělána tak, že prediktory píšete v programovacím jazyce C++. To mi naštěstí nedělalo problém. Když jsem tedy měl konečně způsob, jak testovat prediktory, pustil jsem se do jejich stavby. Nejdříve jsem nasimuloval prediktory, které jsem si načetl z různých vědeckých prací (kapitola č. 4).

Brzy jsem ale dostal chuť postavit si vlastní prediktor větvení na základě znalostí načtených z různých vědeckých prací a následnou simulací prediktorů (kapitola č. 7). Samozřejmě jsem neměl šanci soutěžit s účastníky soutěže CBP 2016. Jako svého soupeře jsem si vybral prediktor 2bc-Gskew použitý v nedokončeném procesoru Alpha 21464 (někdy také znám pod kódovým označením Alpha EV8), jehož vývoj byl zrušen v roce 2001. Tento prediktor jsem si vybral z toho důvodu, že podle mě reprezentuje to nejlepší, co bylo v dané době k dispozici. Je k němu dostupná poměrně rozsáhlá dokumentace a je považován za pomyslný vrchol dané éry před nástupem perceptronových a TAGE prediktorů.

Prediktor v tomto procesoru dosahuje MPKI 5.485 (MPKI – zjednodušeně počet chyb, čím nižší, tím lepší. Viz kapitola Metodika pro vysvětlení pojmu MPKI), já si tedy stanovil jako svůj cíl dosáhnout 5.400 MPKI.

V následujících kapitolách bych vám chtěl představit proces vývoje mého prediktoru KVN.

³ <https://www.jilp.org/cbp2016/>

2. ÚVOD DO PREDIKCE VĚTVENÍ

Proč je predikce větvení vlastně tak důležitá?

Představte si koleje v 18. století, na nichž se nachází výhybka. Jelikož míra automatizace je v této době ještě dost omezená, výhybku ovládá manuálně výhybkář.



Obrázek č. 1 – Výhybka, zdroj:

https://commons.wikimedia.org/wiki/File:Entroncamento_do_Transpraia.JPG

Když k této výhybce přijede vlak, musí před ní zastavit (což vzhledem k jeho hmotnosti není jen tak), strojvůdce vykukne z okna a řekne výhybkáři „Přepni mi to doleva“, výhybkář přepne výhybku a vlak se začne pomalu rozjíždět.

Asi si dokážete představit, že tento systém má daleko k dokonalosti. Vlak musí brzdit a hned se znovu rozjíždět, což představuje značné plýtvání energií. Zároveň vlak na každé výhybce ztrácí čas.

Ale co kdyby si výhybkář začal značit, který vlak odbočuje vlevo, a který vpravo, a na základě získaných dat následně „tipoval“ a předčasně přepínal výhybky tak, aby vlak nemusel zastavovat?

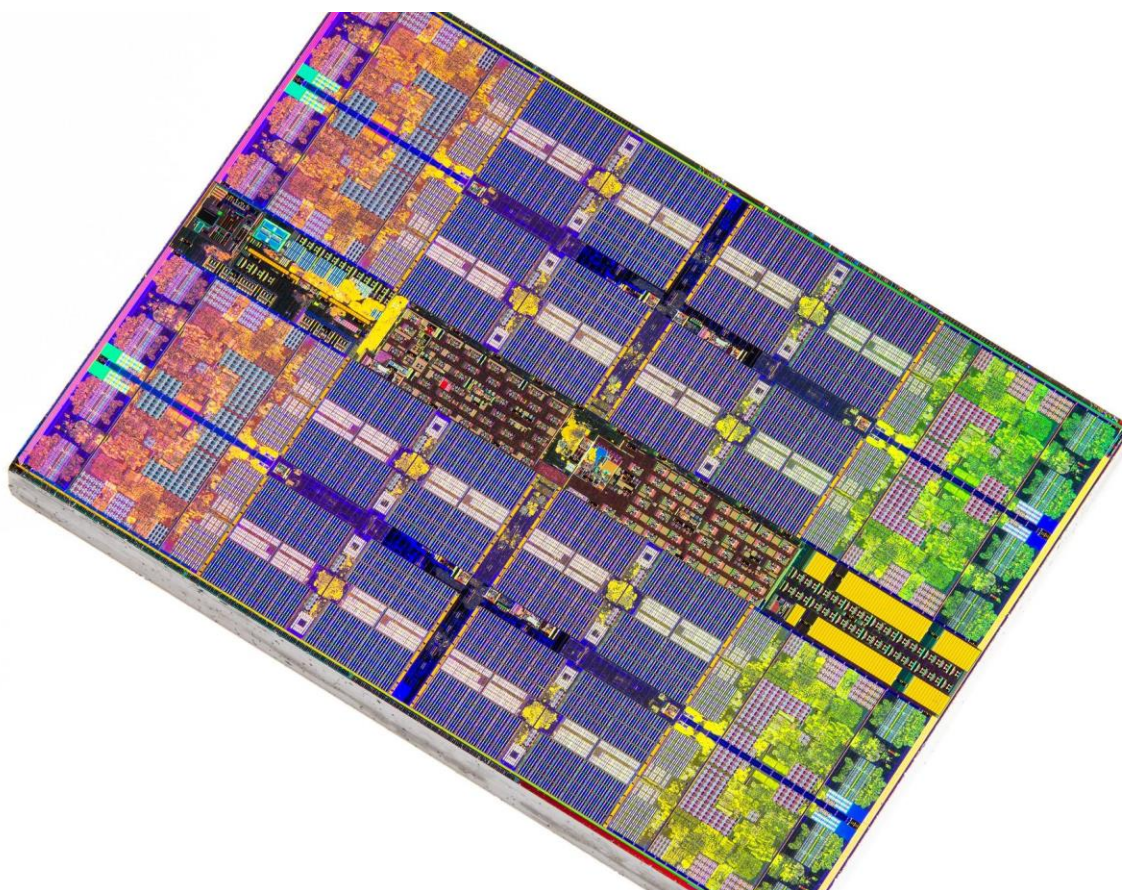
Tato analogie dokonale popisuje chování prediktorů větvení v procesorech. Pokaždé, když procesor narazí na větev (instrukce reprezentující programátorovu podmínku, naše pomyslná výhybka), musí zastavit a čekat, dokud nezjistí, zda bude větev přijata nebo odmítnuta. Ztracený čas čekáním je přímo úměrný ztracenému výkonu, a navíc je čekáním proplýtváno velké množství energie.

Prediktor větvení je schopen toto čekání obejít. Pokaždé, když procesor narazí na větev, prediktor větvení udělá „tip“ (predikci) zda bude větev přijata. Pokud je tato predikce správná, procesor nemusí zastavovat a pokračuje ve vykonávání programu. Pokud je tato predikce špatná (mispredikce), procesor musí zastavit a provést tzv. pipeline flush – zahodit dosavadní výsledky a vrátit se zpět až ke špatně predikované větvi.

2.1 JAK TO VYPADÁ V PRAXI

V úvodu jsem poukázal na existenci procesorového jádra. Jádro je digitální obvod schopný načítat a zpracovávat data a instrukce. Mnoho dnešních procesorů obsahuje více než 1 jádro (spotřebitelské procesory mívají 2-18 jader, serverové až 80 jader), přičemž každé jádro je schopné pracovat nezávisle na ostatních.

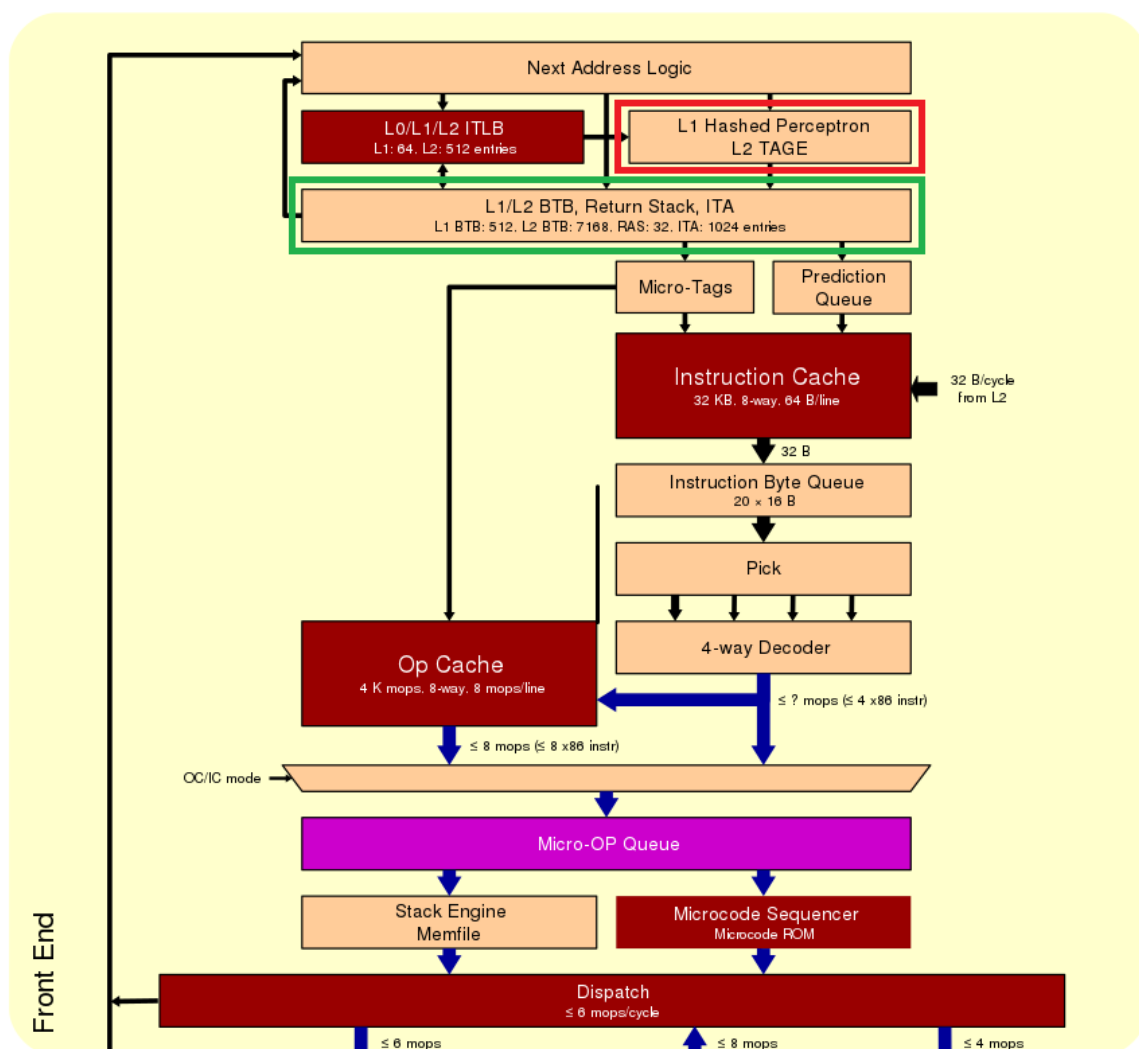
Pro ilustraci, jak takový procesor vypadá, pokud ho „rozeberete“ a následně odleptáte vodiče, lze vidět na obrázku č. 2:



Obrázek č. 2 – Die shot procesoru Ryzen 3xxx, zdroj: <https://flic.kr/p/2hShx9G>

Na obrázku je procesor AMD Ryzen 3xxx obsahující 8 jader Zen 2 složený z 3.9 miliard tranzistorů umístěných na ploše pouhých 74 mm². Bohužel integrace postoupila natolik, že jak pohledem, tak ani mikroskopem nejsme s určitostí schopni rozeznat jednotlivé bloky uvnitř procesorového jádra, a výrobci tyto informace často nezveřejňují

O to užitečnější je blokové schéma samotného jádra (obrázek č. 3), z kterého lze leccos vyčíst:



Obrázek č. 3 – Blokové schéma jádra AMD Zen 2, zdroj:
https://en.wikichip.org/wiki/amd/microarchitectures/zen_2

Jedná se o část blokového schéma jádra AMD Zen 2 (tzv. „Front End“, část jádra zodpovědná za načítání instrukcí. Tak zvaný „Back End“ pak obsahuje všechny výpočetní jednotky ALU, AGU, FPU, SIMD a jeho prací je zpracování instrukcí).

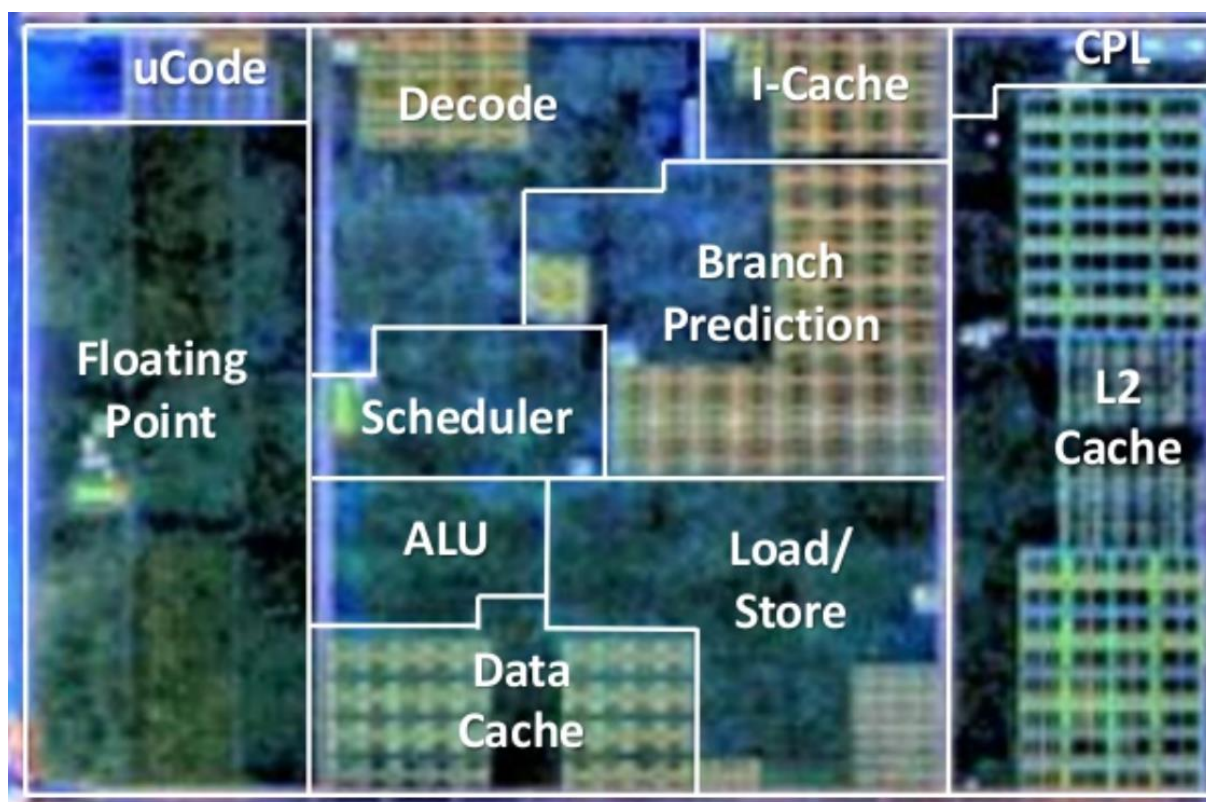
Červeným čtvercem jsem označil prediktor větvení – „L1 Hashed Perceptron/L2 TAGE“.

Toto jádro používá dvouúrovňový prediktor větvení – L1 Perceptronový prediktor je méně přesný, ale za to je rychlý a je schopný poskytnout predikci během jednoho cyklu. L2 TAGE prediktor je mnohem přesnější, ale poskytnout predikci mu trvá mnohem déle.

Tyto prediktory se tedy pěkně doplňují – predikce je nejdříve založena na rychlém a málo přesném (řádově se bavíme o přesnosti 96-99 %) L1 prediktoru, o několik cyklů později predikci opraví přesný (přesnost 99+ %), ale pomalý L2 prediktor.

Zeleným čtvercem jsem označil BTB – Branch Target Buffer (paměť cílových adres). Tato komponenta je úzce svázána s prediktorem větvení – prediktor větvení predikuje **jestli** větev „skočí“ (zda bude přijata), BTB pak predikuje **kam** větev skočí (predikuje cílovou adresu skoku).

Na obrázku č. 4 můžete vidět reálný snímek jádra Zen 2 přiblíženého pod mikroskopem – celá fotografie představuje plochu pouhých několika milimetrů čtverečních. Jedná se o oficiální snímek, a tak můžeme věřit, že všechny bloky jsou zaznačeny správně. Zhruba uprostřed je velký blok „Branch Prediction“ – zde spadá jak prediktor větvení, tak i BTB. Jde vidět, že prediktor větvení je jeden z největších bloků v celém jádře.



Obrázek č. 4 – Jádro Zen 2 s jednotlivými vyznačenými bloky, Zdroj:

Prezentace na konferenci ISSCC 2020 s názvem: „Zen 2: The AMD 7nm Energy-Efficient High-Performance x86-64 Microprocessor Core“

3. METODIKA

Jak už bylo řečeno, všechny prediktory byly simulovány v infrastruktuře soutěže CBP 2016. Každá simulace se skládá ze 440 testů (nazývané „Trace“) reprezentující chování reálných programů. Velikost jednotlivých testů se pohybuje přibližně od milionu až po 20 miliard instrukcí.

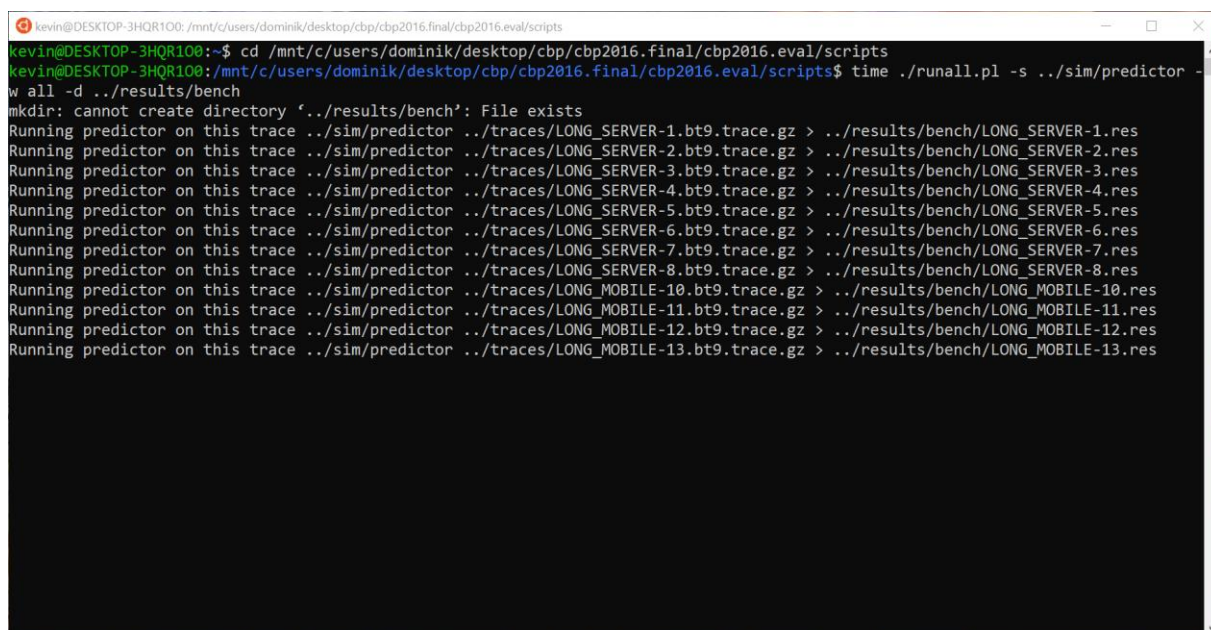
Vypočítat jednu simulaci trvá přibližně hodinu, přičemž jsem celkově provedl asi 600 simulací.

Výsledky simulací jsou možná trochu neintuitivně udávány v MPKI – misspredictions per thousand instructions (počet mispredikcí na 1000 instrukcí). Proč nejsou použita procenta? Ačkoliv se může zdát, že udávat přesnost v procentech je jednoznačně nejlepší možnost, v praxi tomu tak není.

Pokud budeme mít 1000 instrukcí, z toho 50 je větví a my 5 z nich špatně predikujeme, naše přesnost bude 90 %. Jelikož máme 5 mispredikcí na 1000 instrukcí, naše MPKI bude 5 - procesor bude muset 5x za každých tisíc instrukcí zastavit kvůli špatné predikci.

Pokud budeme mít 1000 instrukcí, z toho 200 větví a my z nich 20 špatně predikujeme, naše přesnost sice zůstane na 90 %, ale MPKI povyskočí na 20 - procesor bude muset 20x za každých 1000 instrukcí zastavit kvůli špatné predikci.

Ačkoliv přesnost v procentech zůstala stejná v obou případech, MPKI nám ukázalo, že ve skutečnosti bude procesor muset zastavovat 4x častěji. Z tohoto důvodu budu přesnost prediktorů uvádět v MPKI.



```
kevin@DESKTOP-3HQ100: /mnt/c/users/dominik/desktop/cbp/cbp2016.final/cbp2016.eval/scripts
kevin@DESKTOP-3HQ100:~$ cd /mnt/c/users/dominik/desktop/cbp/cbp2016.final/cbp2016.eval/scripts
kevin@DESKTOP-3HQ100:/mnt/c/users/dominik/desktop/cbp/cbp2016.final/cbp2016.eval/scripts$ time ./runall.pl -s ../sim/predictor -
w all -d ../results/bench
mkdir: cannot create directory '../results/bench': File exists
Running predictor on this trace ../sim/predictor ../traces/LONG_SERVER-1.bt9.trace.gz > ../results/bench/LONG_SERVER-1.res
Running predictor on this trace ../sim/predictor ../traces/LONG_SERVER-2.bt9.trace.gz > ../results/bench/LONG_SERVER-2.res
Running predictor on this trace ../sim/predictor ../traces/LONG_SERVER-3.bt9.trace.gz > ../results/bench/LONG_SERVER-3.res
Running predictor on this trace ../sim/predictor ../traces/LONG_SERVER-4.bt9.trace.gz > ../results/bench/LONG_SERVER-4.res
Running predictor on this trace ../sim/predictor ../traces/LONG_SERVER-5.bt9.trace.gz > ../results/bench/LONG_SERVER-5.res
Running predictor on this trace ../sim/predictor ../traces/LONG_SERVER-6.bt9.trace.gz > ../results/bench/LONG_SERVER-6.res
Running predictor on this trace ../sim/predictor ../traces/LONG_SERVER-7.bt9.trace.gz > ../results/bench/LONG_SERVER-7.res
Running predictor on this trace ../sim/predictor ../traces/LONG_SERVER-8.bt9.trace.gz > ../results/bench/LONG_SERVER-8.res
Running predictor on this trace ../sim/predictor ../traces/LONG_MOBILE-10.bt9.trace.gz > ../results/bench/LONG_MOBILE-10.res
Running predictor on this trace ../sim/predictor ../traces/LONG_MOBILE-11.bt9.trace.gz > ../results/bench/LONG_MOBILE-11.res
Running predictor on this trace ../sim/predictor ../traces/LONG_MOBILE-12.bt9.trace.gz > ../results/bench/LONG_MOBILE-12.res
Running predictor on this trace ../sim/predictor ../traces/LONG_MOBILE-13.bt9.trace.gz > ../results/bench/LONG_MOBILE-13.res
```

Obrázek č. 5 – Spuštění simulace

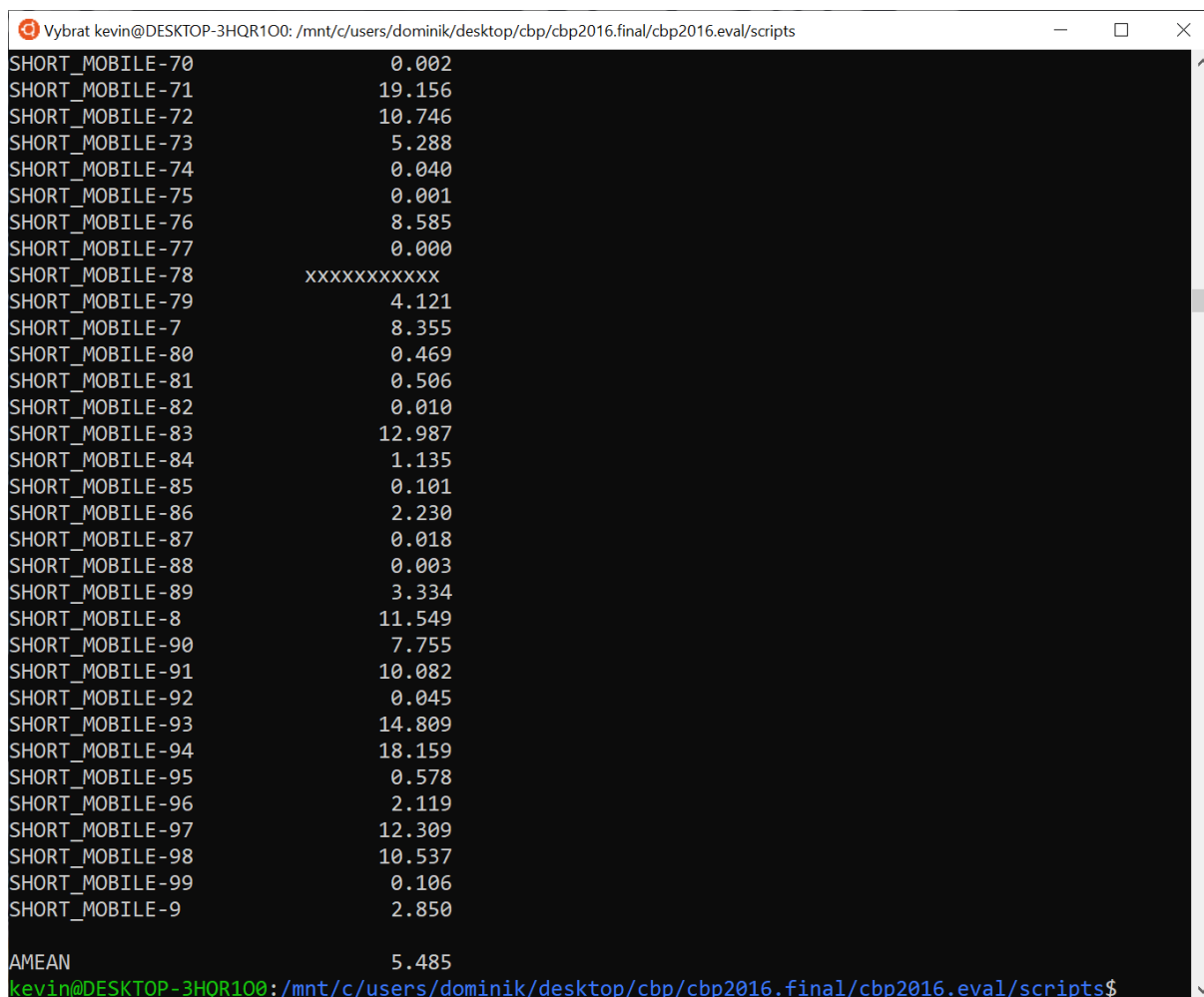
Jelikož prediktory jsou digitální obvody, jejich velikost/cena odpovídá počtu použitých tranzistorů na jejich postavení. V praxi se toto nepoužívá, protože zjistit přesný počet tranzistorů je nemožné – záleží na konkrétní implementaci v konkrétním procesoru. Proto se velikost (cena) prediktorů udává v KB – množství paměti, které prediktor používá.

Například v soutěži CBP 2016 se soutěžilo v kategoriích 8 KB, 64 KB a Unlimited (neomezené množství paměti). Já jsem si pro svůj prediktor KVN vybral rozpočet 32 KB.

3.1 JAK VYPADAJÍ SIMULACE

Samotné simulace se spouští z Linuxové příkazové řádky zadáním příkazu **./runall.pl** – obrázek č. 5. Po zadání tohoto příkazu se prediktor postupně otestuje na všech 440 testech – to trvá zhruba hodinu.

Po dokončení simulace můžeme příkazem **./getdata.pl** získat její výsledky – obrázek č. 6.



```
Vybrat kevin@DESKTOP-3HQR100: /mnt/c/users/dominik/desktop/cbp/cbp2016.final/cbp2016.eval/scripts
SHORT_MOBILE-70      0.002
SHORT_MOBILE-71     19.156
SHORT_MOBILE-72     10.746
SHORT_MOBILE-73      5.288
SHORT_MOBILE-74      0.040
SHORT_MOBILE-75      0.001
SHORT_MOBILE-76      8.585
SHORT_MOBILE-77      0.000
SHORT_MOBILE-78      xxxxxxxxxxxx
SHORT_MOBILE-79      4.121
SHORT_MOBILE-7      8.355
SHORT_MOBILE-80      0.469
SHORT_MOBILE-81      0.506
SHORT_MOBILE-82      0.010
SHORT_MOBILE-83     12.987
SHORT_MOBILE-84      1.135
SHORT_MOBILE-85      0.101
SHORT_MOBILE-86      2.230
SHORT_MOBILE-87      0.018
SHORT_MOBILE-88      0.003
SHORT_MOBILE-89      3.334
SHORT_MOBILE-8      11.549
SHORT_MOBILE-90      7.755
SHORT_MOBILE-91     10.082
SHORT_MOBILE-92      0.045
SHORT_MOBILE-93     14.809
SHORT_MOBILE-94     18.159
SHORT_MOBILE-95      0.578
SHORT_MOBILE-96      2.119
SHORT_MOBILE-97     12.309
SHORT_MOBILE-98     10.537
SHORT_MOBILE-99      0.106
SHORT_MOBILE-9      2.850
AMEAN                5.485
kevin@DESKTOP-3HQR100:/mnt/c/users/dominik/desktop/cbp/cbp2016.final/cbp2016.eval/scripts$
```

Obrázek č. 6 – Výsledek simulací

V levém sloupci na obrázku č. 6 jsou vypsané názvy jednotlivých testů, v pravém sloupci jsou výsledky 2bc-gskew-EV8 prediktoru v MPKI – pod sebou je vypsané všech 440 výsledků pro jednotlivé testy. Dole je pak spočítán „AMEAN“ – průměrné MPKI ze všech 440 testů, tento údaj představuje finální výsledek prediktoru. Pokud tedy někde v práci uvedu, že některý prediktor dosáhl MPKI x.xxx, mluvím právě o tomto údaji.

Zajímavé je, že např. test SHORT_MOBILE-77 dosahuje MPKI 0.000 - to znamená 0 mispredikcí (ono jich není přesně nula, pokud se podíváme do souborů s výsledky, zjistíme, že se jedná přesně o 3 mispredikce na 5 miliónů větví/ 100 miliónů instrukcí, což lze zaokrouhlit na nulu) a tudíž dosahuje přesnosti 100 %.

Ještě zajímavější je výsledek testu SHORT_MOBILE-78 – „xxxxxxxxxxxx“. Na první pohled se může zdát, že se jedná o chybný výsledek, ale pokud se podíváme do souborů s výsledky, zjistíme, že prediktor udělal přesně 0 mispredikcí na 5 miliónů větví/145 miliónů instrukcí. A jelikož dělení nulou je zakázaná operace, výsledek je „xxxxxxxxxxxx“ – čili taktéž přesnost 100%.⁴

Oba výsledky jsem zkontroloval a nejedná se o chybu. Prediktory větvení jsou skutečně schopné dostat se v některých případech na 100 % přesnost. Tyto výsledky nejsou ojedinělé, podobné výsledky lze nalézt minimálně u dalších 10 testů.

3.2 BITY A BYTY

V průběhu práce budou používány tyto jednotky označující velikost prediktoru:

1 b – jeden bit (logická 1 nebo 0)

1 B – jeden Byte (8 bitů)

1 Kb – jeden Kilobit (1024 bitů)

1 KB – jeden KiloByte (1024 Bytů)

Pozor! - předpona K (Kilo) v počítačové architektuře představuje 1024násobek základní jednotky, nikoli 1000násobek, jak jsme zvyklí ze soustavy SI, respektive z předpon soustavy SI. Podobně předpona M (Mega) představuje 1024násobek předpony Kilo.

Rozpočet 32 KB tedy představuje $32 * 8 * 1024 = 262\,144$ bitů, nikoli $32 * 8 * 1000 = 256\,000$ bitů

⁴ „xxxxxxxxxxxx“ nemusí znamenat pouze 0 mispredikcí. Někdy (opravdu vzácně) to může skutečně znamenat chybný výsledek – to se může stát (a reálně se mi to stalo), například když jeden ze souborů s výsledky kompletně chybí protože při simulaci nastala nějaká chyba.

4. SEZNÁMENÍ SE ZÁKLADNÍMI PREDIKTORY

Predikce větvení se dělí na dva základní druhy:

- Statická – nepoužívá historii
- Dynamická – používá historii

4.1 STATICKÁ PREDIKCE

Statickou predikci lze nalézt u těch nejjednodušších procesorů, nevyužívá žádnou formu historie. Jedná se o „levnou“ metodu predikce, jejíž implementace je kolikrát prakticky zdarma.

Základní dva způsoby jsou Always Taken a Always Not Taken. Jak už jejich název naznačuje, první jmenovaný způsob – Always Taken – predikuje, že všechny větve budou vždy přijaty. Druhý způsob – Always Not Taken – predikuje, že všechny větve budou vždy **nepřijaty**.

Pokud zvolíme metodu Always Taken, dosáhneme MPKI 80.383. Metoda Always Not Taken pak dosáhne MPKI 51.256 (připomínám – nižší MPKI = lepší).⁵

Pro porovnání, základní dynamické prediktory se pohybují okolo MPKI 10, čili jsou pětikrát lepší. Ale i přes to je statická predikce i dnes používána v různých jednoduchých mikrokontrolerech.

Zajímavou alternativou je metoda Backward Taken Forward Not Taken (zkráceně BTFN). Tato metoda je o poznání důmyslnější. Je založena na předpokladu, že cykly skáčou v programu **dozadu**, kdežto podmínky skáčou v programu spíše **dopředu**. Jedna ze základních vlastností cyklů je, že bývají mnohokrát opakovány, kdežto podmínky bývají přijaty spíše zřídka.

⁵ Ačkoliv mé výsledky naznačují, že pokud použijeme Always Not Taken, dosáhneme vyšší přesnosti, mnohé starší vědecké práce dosahují opačných výsledků - např. často se uvádí, že při použití Always Taken je přesnost 60 % a při použití Always Not Taken je přesnost 40 % (čili Always Taken je lepší volba).

Tato nesrovnalost je způsobena tím, že moderní kompilátory jsou schopny provádět velké množství optimalizací a jednou z nich je organizování větví právě tak, aby byly většinou nepřijaty. Starší kompilátory neprováděly tuto optimalizaci, a tak byly větve většinou přijaty (to vyplývá z toho, že pro programátora je přirozenější psát kód, kde jsou podmínky často přijaty). Proto se opravdu často ve starší literatuře dočtete, že většina větví bývá přijata – tuto informaci mnoho lidí i dnes přejímá, a tak se stále objevuje i v mnoha aktuálních článcích a literatuře.

Mně samotnému tento fakt dost zamotal hlavu – pokud si přečtete 10 postarších vědeckých prací, a 10 z nich tvrdí, že větve bývají většinou přijaty, a pak jeden z vašich prvních výsledků, který nasimulujete, tvrdí úplný opak, tak to ve vás vzbudí velké pochyby o správnosti/kvalitě/relevantnosti vašich výsledků – nicméně mé výsledky jsem ověřil a byl jsem si jejich správností jistý. Trvalo mi několik měsíců, než jsem přišel na to, co tuto nesrovnalost mezi postarší literaturou a mými výsledky způsobuje.

Když tedy procesor narazí na větev, která skáče **dozadu**, předpokládá, že se jedná o cyklus, a tak predikuje že tato větev bude přijata. Ale když narazí na větev, která skáče **dopředu**, předpokládá, že se jedná o obyčejnou podmínku, a predikuje, že nebude přijata.

Tato metoda dosáhla MPKI 26.191 – čili dvakrát lepší přesnost než metoda Always Not Taken.

4.2 SW METODY PREDIKCE

Predikci lze řešit nejen hardwarově, ale také i softwarově – nejznámější metody jsou Hint bit a Delay slot.

Delay slot byl použit v instrukční sadě MIPS (představena roku 1985 a i dnes se používá v různých mikrokontrolerech). Delay slot zajišťuje, že instrukce umístěná hned za větvi bude vždy vykonána, nezáleží na tom, jestli bude podmínka přijata či odmítnuta – díky tomu je penalizace za mispredikci snížena o jeden cyklus.

V době, kdy byla architektura MIPS představena, se Delay slot zdál jako dobrý nápad, jak nahradit drahé dynamické prediktory větvení, bohužel tato metoda má ale mnoho neduhů – největší z nich je potřeba nějak zaplnit Delay slot. Programátor (respektive kompilátor) musí najít instrukci, která je společná pro obě strany větve – ne vždy lze takovou instrukci najít.

Dnes je Delay slot považován spíše za přítěž – komplikuje načítání instrukcí, přidává práci programátorovi/kompilátoru a jeho efekt na výkon je nízký (v momentě, kdy penalizace za mispredikci představuje 10-20 cyklů – což není vůbec nerealistické – má i zaplněný Delay slot mizivý efekt na výkon).

Metoda Hint bit (nápo vědný bit) spočívá v tom, že každá větev má v sobě jeden bit, který pro procesor slouží jako nápo věda, zda bude větev přijata (pokud je bit nastaven na logickou 1, tak napo vídá procesoru, že větev bude přijata).

Problém této metody je podobný jako u Delay slotu – někdo musí nastavit Hint bity. Většina programátorů ani netuší, že Hint bit existuje, takže nastavení Hint bitu zůstává na kompilátoru – ten může nastavit Hint bity buď staticky dle různých předpokladů (např. že cykly jsou mnohokrát přijaty), nebo dynamicky pomocí tzv. profilování – kompilátor sleduje běh programu (kolikrát jsou které větve přijaty) a ze získaných dat pak vytváří Hint bity. V praxi se toto používá velmi málo často a kvalita nastavených Hint bitů bývá dost nízká.

Přesnost metody Hint bit nelze jednoduše otestovat, jelikož je úzce svázána s kvalitou nastavených Hint bitů – a ta se drasticky liší program od programu. Moderní procesory Hint bity buď nemají implementovány vůbec, nebo je ignorují, protože jejich dynamické prediktory jsou tak silné, že ve všech případech překonají i dobře nastavené Hint bity.

4.3 DYNAMICKÁ PREDIKCE

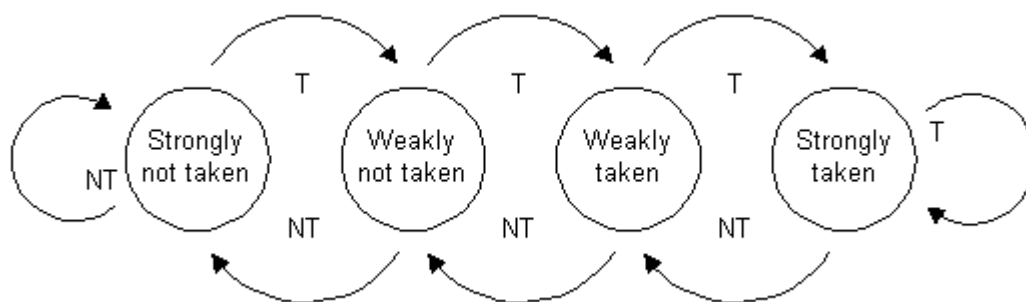
Základní vlastností dynamických prediktorů je schopnost uchovávat si historii o chování programu/větví.

Drtivá většina prediktorů používá tzv. „2-bit Saturating Counter“ (doslovně přeloženo „dvou-bitové saturující počítadlo“, v práci bude dále uváděn jako „2-bitový counter“), který můžete vidět na obrázku č. 7. Jedná se o sekvenční logický obvod, který byl poprvé představen v práci „A STUDY OF BRANCH PREDICTION STRATEGIES“^[1]. Tento sekvenční logický obvod má čtyři stavy – Strongly not taken (silně nepřijatá), Weakly not taken (slabě nepřijatá), Weakly taken (slabě přijatá) a Strongly taken (silně přijatá).

Pokud se počítadlo nachází ve stavech „Strongly not taken“ nebo „Weakly not taken“, budeme predikovat Not Taken (nepřijatá větev).

Pokud se počítadlo nachází ve stavech „Strongly taken“ nebo „Weakly taken“, budeme predikovat Taken (přijatá větev).

Princip aktualizace 2-bitového counteru lze krásně vysvětlit na našem přirovnání s vlaky – pokud budou vlaky jezdit doprava, bude se stav našeho counteru posouvat také doprava. Pokud budou vlaky jezdit doleva, bude se stav counteru posouvat doleva.



Obrázek č. 7 – Přejchodový diagram 2bit Saturating Counteru, zdroj:
<https://www.agner.org/optimize/microarchitecture.pdf>

Když to převedeme na větve – pokud bude větev přijata, posouvá se stav counteru po šipkách T (Taken) doprava. Pokud bude větev nepřijata, posouvá se stav counteru po šipkách NT (Not Taken) doleva.

Podobně jako je na obrázku zobrazen 2-bitový counter, lze sestavit i 3-bitový counter, který má 8 stavů, teoreticky lze použít jakýkoliv počet bitů. Prakticky se u starších prediktorů setkáváme hlavně s 2-bitovými a výjimečně se 3-bitovými country, u novějších prediktorů se často vyskytují 3- a 4-bitové country.

Všechny prediktory (až na výjimky) jsou sestaveny z x-bitových Saturating Counterů, používají jich opravdu velké množství (řádově tisíce až statisíce). Jednotlivé prediktory se pak odlišují organizací těchto Counterů do polí (nejjednodušší prediktory používají jedno velké pole Counterů, moderní TAGE prediktory obsahují velké množství polí, např. 24), indexováním těchto polí, politikou aktualizování jednotlivých polí, množstvím použité historie, tagováním.

4.4 BIMODAL PREDICTOR

Bimodal predictor/bimodální prediktor (zobrazen na schéma č. 1) je nejjednodušší prediktor ze všech, používá jediné pole 2-bit Saturating Counterů, které je indexované adresou větve (PC – Program Counter) - adresu větve si lze představit jako její jméno.

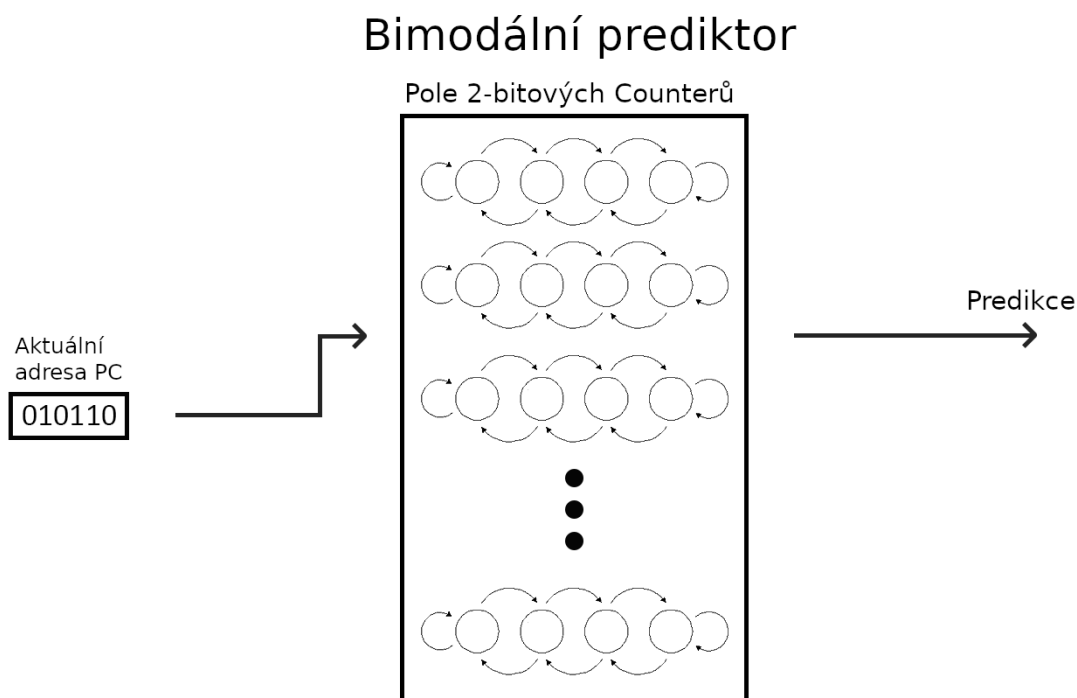


Schéma č. 1 – Bimodální prediktor

Pole je indexováno pomocí N spodních bitů z aktuální adresy PC, velikost pole je pak rovna 2^N . Např. pokud $N = 4$, tak je pole indexováno čtyřmi spodními bity adresy PC a jeho velikost je $2^4 = 16$ záznamů – každý záznam obsahuje jeden 2-bitový counter, takže velikost takového prediktoru je 32 bitů.

Ačkoliv každá větev má svou vlastní 32/64 bitovou adresu (své vlastní unikátní jméno), jelikož je použito nedokonalé indexování (z 32/64 bitové adresy používáme pouze N spodních bitů), často dochází ke konfliktům – větev A a větev B se „perou“ o jeden záznam v poli (jeden Counter).

Vezměme si příklad, kdy $N = 4$, větev „A“ má adresu „10010“ a větev „B“ má adresu „00010“. Jelikož prediktor vidí jen spodní 4 bity, které jsou pro obě větve shodné, obě větve se namapují do counteru s adresou „0010“ - dojde tedy ke konfliktu. Pokud tomuto chování chceme zabránit, musíme zvýšit N na 5 (aby prediktor viděl i pátý bit adresy) a tím zvýšit i cenu prediktoru (jeho velikost se zvedne z 32 bitů na 64 bitů).

Tento prediktor jsem nasimuloval ve velikostech od 4 bitů až do 8 KB. Z grafu č. 1 můžeme vidět, že i při použití pouhých 4 bitů je tento prediktor schopný překonat statickou predikci – metoda Always Not Taken dosahovala přesnosti MPKI 51.256, 4-bitový bimodální prediktor dosahuje přesnosti MPKI 35.404.

Ze začátku tento prediktor škáluje velmi pěkně (škáluje = se zvyšující velikostí dostáváme odpovídající nárůst v přesnosti), ale když se začneme přibližovat k hranici MPKI 10, prediktor začne se zvyšující kapacitou škálovat opravdu špatně. Zdvojnásobení velikosti ze 4 KB na 8 KB způsobí nárůst přesnosti o pouhých 0.069 MPKI. Simulovat vyšší velikost nemá smysl, protože jde vidět, že tento prediktor se škálováním končí někde okolo velikosti 1 KB.

4.5 TWO LEVEL ADAPTIVE BRANCH PREDICTOR

Roku 1991 vydal Tse-Yu Yeh a Yale Nance Patt práci s názvem „Two-level adaptive training branch prediction“ [2]. V této práci představili prediktor s dvěma úrovněmi historie – první úroveň zaznamenává do speciálního registru (GHR – Global History Register, registr globální historie), zda bylo posledních N větví přijato (logická 1) nebo odmítnuto (logická 0).

Např. pokud je v GHR uložena posloupnost „0001“, znamená to, že nejdříve procesor zpracoval tři nepřijaté větve a po nich následovala jedna přijatá větev. Pokud by další větev byla přijatá, změní se obsah GHR na „0011“.

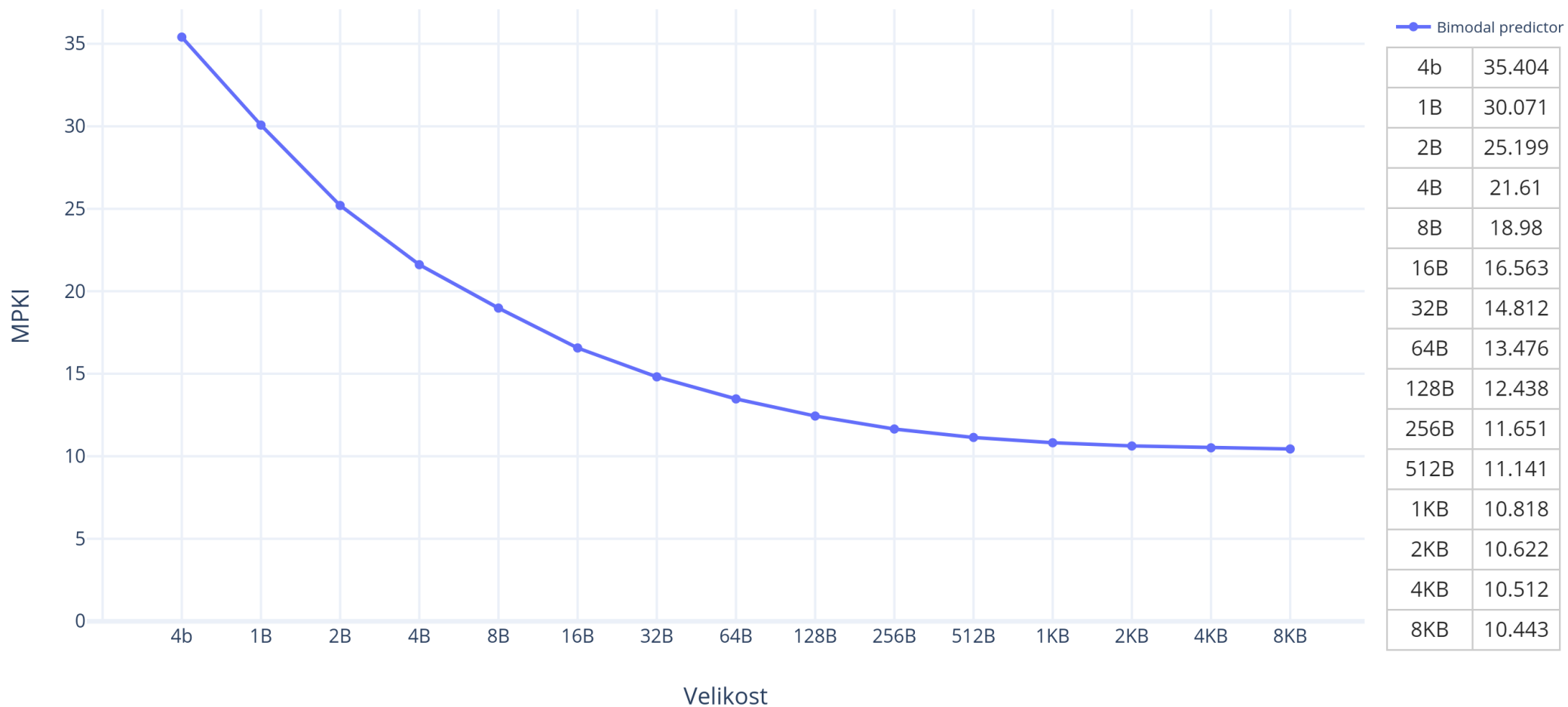
GHR je následně využit pro adresování pole counterů (druhá úroveň historie), toto pole se nazývá PHT – Pattern History Table. Tato metoda predikce se nazývá Global prediction, globální predikce.⁶ Tento prediktor můžete vidět na schéma č. 2

⁶ Yeh a Patt původně uvažovali schéma, kdy každá větev má svůj vlastní speciální registr, kde se ukládá historie pouze dané větve – později se tomuto schématu začalo říkat Local Prediction, lokální predikce. Oproti tomu schéma s jediným registrem pro ukládání historie všech větví se nazývá Global prediction, Globální predikce.

Lokální predikce má nevýhodu v podobě potřeby velkého množství (např. 512) N -bitových registrů – pokud je naše $N = 14$, potřebuje lokální prediktor 7 168 bitů jen na uložení první úrovně historie, zatímco globální prediktor si vystačí s pouhými 14-ti bity (protože má jen jeden registr ukládající první úroveň historie).

Další nevýhoda lokální predikce spočívá v potřebě spekulativně aktualizovat velké množství registrů. Z těchto důvodů jsou lokální prediktory mnohem méně populární než globální. Kdysi se lokální prediktory prediktory používaly zřídka, příkladem může být procesor **Alpha 21264**, dnes se lokální prediktory v praxi nepoužívají vůbec. Proto jsem ve své práci lokální prediktory úplně vynechal a věnuji se pouze globálním prediktorům.

Bimodal predictor



Graf č. 1 – Přesnost Bimodálního prediktoru, osa Y – přesnost v MPKI, osa X – velikost prediktoru v bitech/Bytech/Kilobytech

Velká výhoda dvouúrovňových adaptivních prediktorů je jejich schopnost rozeznat opakující se vzorce a predikovat je s dokonalou přesností – prediktor je schopný se 100 % přesností predikovat všechny opakující se vzorce o délce $N+1$, kde N je šířka GHR registru.

Pokud má GHR 4 bity, prediktor je se 100 % přesností schopen predikovat opakující se vzorce „0101“, „1010“ nebo „10001“ a podobné

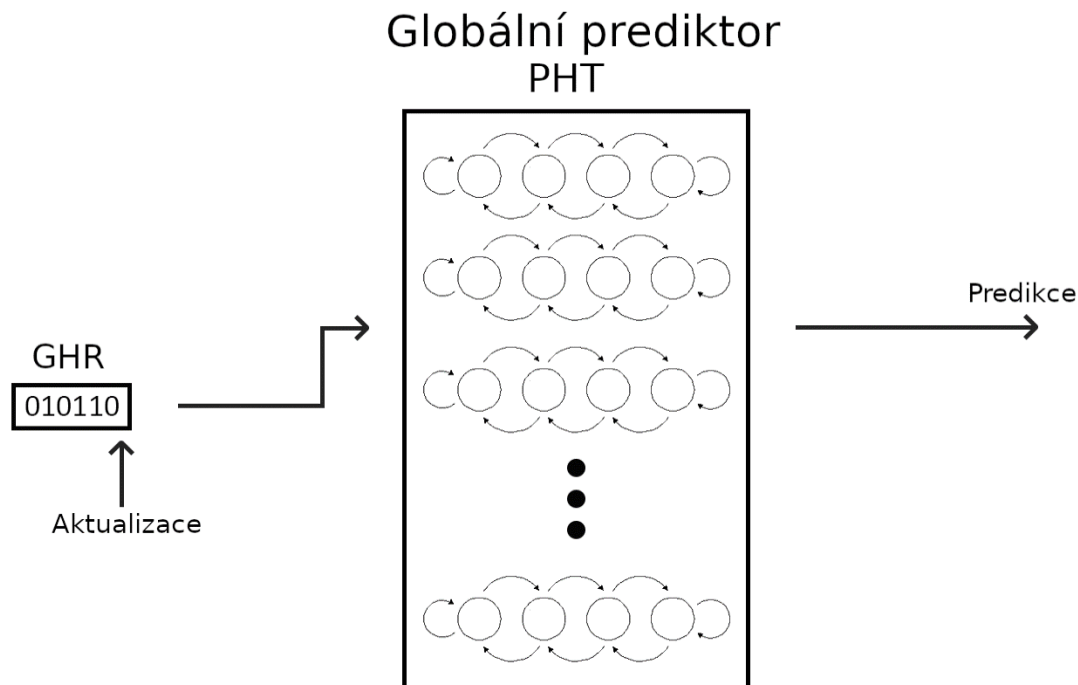


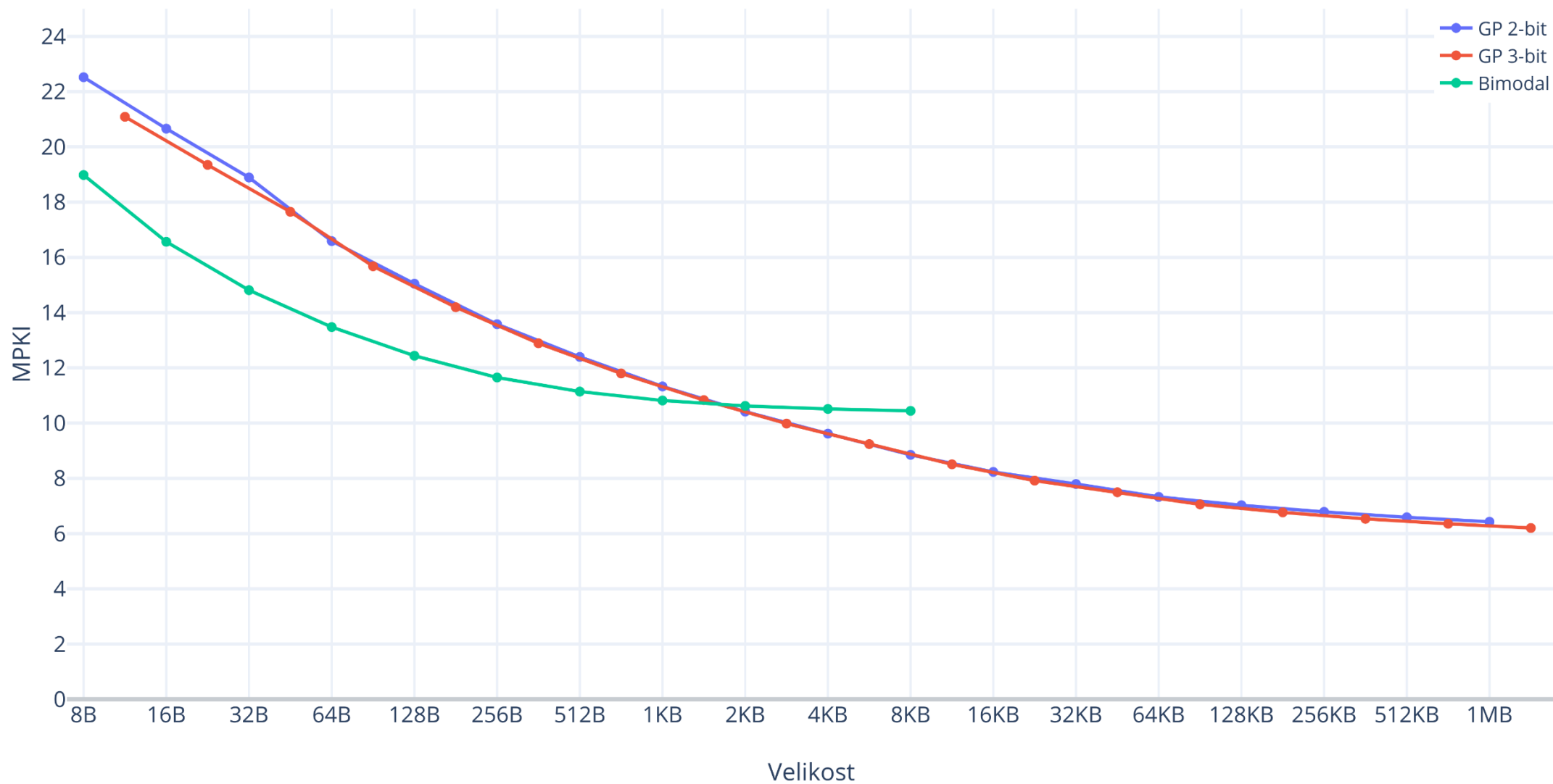
Schéma č. 2 – Globální prediktor

Globální prediktor jsem nasimuloval rovnou ve dvou variantách – první varianta s 2-bitovými country, druhá varianta s 3-bitovými country. Data jsem zanesl do grafu č. 2 (konkrétní výsledky jsou v tabulce č. 1). Obě varianty podávají porovnatelnou přesnost, možná lze říct, že varianta s 3-bitovými country podává jemně lepší výsledky při extrémně malé a extrémně vysoké velikosti, ale v nejzajímavějším středu grafu jsou si obě varianty vyrovnané.

Bimodální prediktor při nízkých kapacitách podává nejlepší přesnost, ale někdy okolo 1 KB přestává na rozdíl od Globálního prediktoru škálovat a obě varianty globálního prediktoru ho předbíhají.

Pro nás je nejdůležitější přesnost při velikosti 32 KB (tuto velikost jsem si vybral jako rozpočet pro svůj prediktor). Varianta s 2-bitovými country dosahuje MPKI 7.795, varianta s 3-bitovými country neexistuje ve velikosti 32 KB, v nejbližší nižší velikosti (24 KB) dosahuje přesnosti 7,918. Přibližně se tedy bavíme o přesnosti 7.8 až 8 MPKI.

Global prediction



Graf č. 2 – Přesnost Globálního prediktoru, osa Y – přesnost v MPKI, osa X – velikost prediktoru v bitech/Bytech/KiloBytech/MegaBytech

4.6 GSHARE

Roku 1993 Scott McFarling ve své práci „Combining Branch Predictors“ [3] objevil rovnou několik zajímavých prediktorů.

Prvním z nich je Gshare. Gshare obsahuje stále jen jedno pole counterů, ale index tohoto pole je vytvořen logickou funkcí XOR mezi registrem GHR a adresou větve PC – Gshare prediktor tedy kombinuje indexování bimodálního prediktoru a globálního prediktoru.⁷ Tento prediktor je zobrazen na schéma č. 3.

Dále McFarling prezentoval úplně nový koncept kombinování prediktorů – jak jsme mohli vidět v grafu č. 2, bimodální a globální prediktor mají odlišné chování, možná že jejich kombinací lze získat vyšší přesnost. Proto McFarling představil novou komponentu – tehdy ji nijak nepojmenoval, ale dnes ji známe pod pojmem META prediktor. Myšlenka za META prediktorem je, že některé větve jdou přesněji predikovat pomocí bimodálního prediktoru a jiné pomocí globálního prediktoru (nebo jiné kombinace dvou prediktorů). META prediktor pak dynamicky vybírá, který ze dvojice prediktorů se použije pro predikci.

Prediktory založené na tomto principu se často označují jako Hybridní prediktory. META prediktor je implementován pomocí pole 2-bitových counterů, které je většinou indexováno adresou PC.

V grafu č. 3 jsem zobrazil data získaná simulací bimodálního, globálního a Gshare prediktoru (globální a Gshare prediktor ve variantě s 2 i 3-bitovými counterami), jedná se o více jak 80 hodin simulací.

Jde vidět, že pro jakýkoliv rozpočet je Gshare vždy o dost přesnější než Globální prediktor. Znovu jsme si potvrdili, že mezi variantou s 2 a 3-bitovými counterami není moc velký rozdíl.

Pokud se zaměříme na náš rozpočet 32 KB (pro připomenutí, globální prediktor dosáhl MPKI 7.795), tak 2-bitová varianta Gshare dosahuje MPKI 6.338 – zlepšení o více jak jedno MPKI. 3-bitová varianta Gshare s velikostí 24 KB dosáhla MPKI 6.453.

⁷ McFarling ve stejné práci představil i Gselect prediktor, který funguje na podobném principu jako Gshare. Ačkoliv co se týče přesnosti byly na tom Gshare a Gselect dost podobně (většinou byl Gshare o fous lepší), v neprospěch prediktoru Gselect hraje nutnost zdlouhavého nastavení parametrů (vyžaduje mnoho simulací). Tyto parametry se navíc mění pro každou velikost prediktoru, testování Gselect je opravdu zdlouhavé a ve výsledku je toto schéma při nejlepším stejně dobré jako Gshare. Proto nebyl Gselect tak „populární“ jako jeho sourozenec Gshare a ani já ho v této práci nebudu rozebírat.

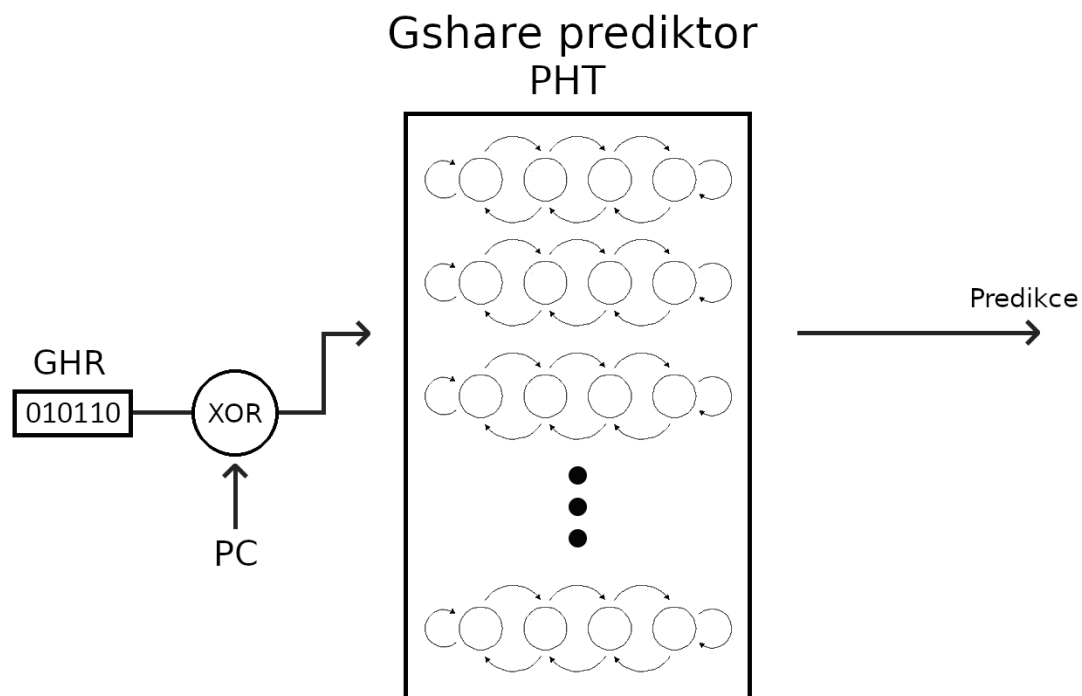
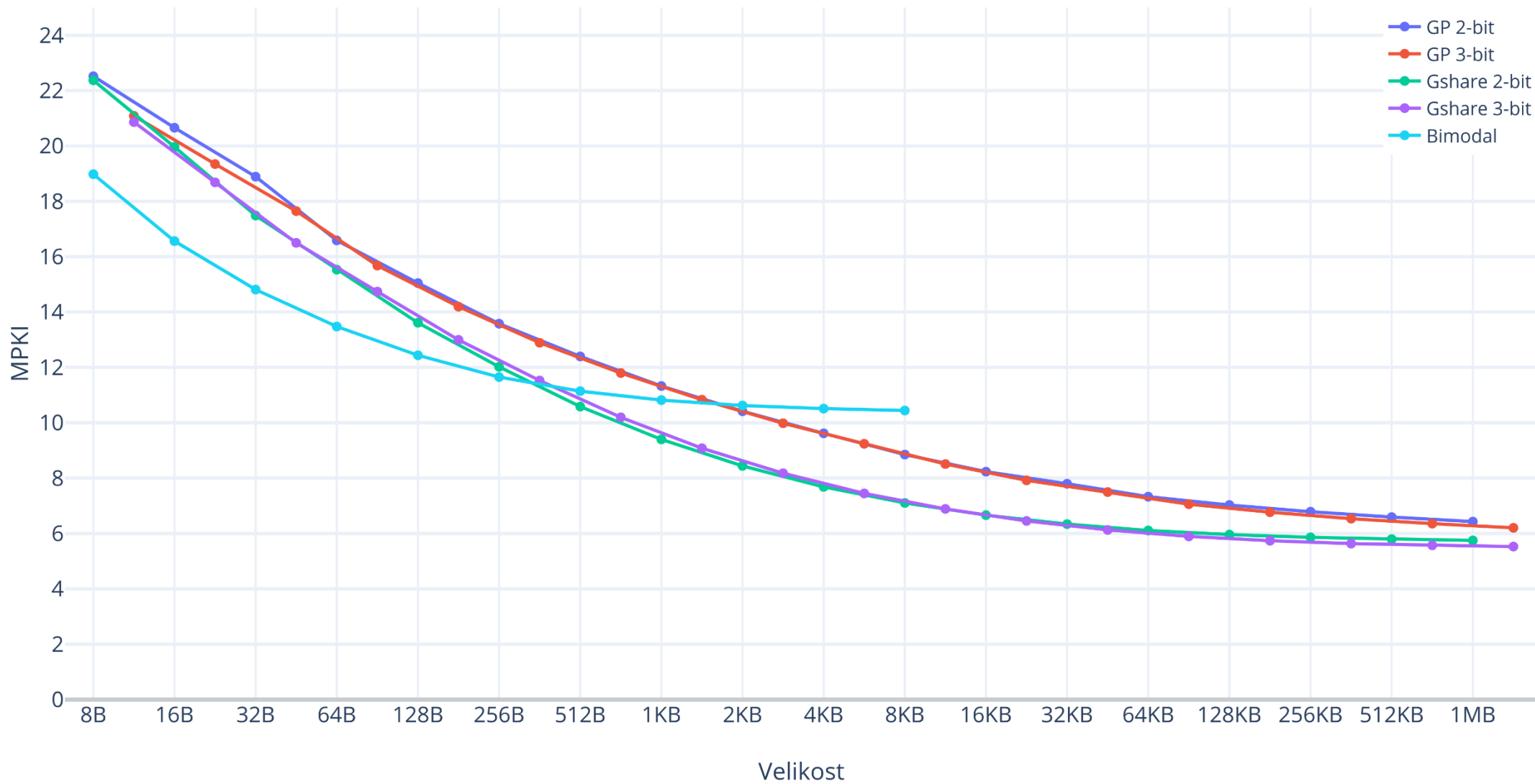


Schéma č. 3 – Gshare prediktor

Gshare prediktor je asi nejznámější ze všech prediktorů co byly kdy představeny. I přes svou jednoduchost je schopný dosáhnout dobré přesnosti a po dlouhou dobu byl považován za „zlatý standard“ a všichni své prediktory porovnávali právě s Gshare prediktorem. Tento prediktor byl použit v mnoha procesorech a domnívám se, že i dnes je stále populární (bohužel výrobci se většinou neobtěžují publikovat informace o použitém prediktoru větvení) a našli bychom ho v různých jednoduchých procesorech (možná i v levných procesorech pro telefony, ale to je jen můj odhad).

Taktéž META prediktor a na něm založené Hybridní prediktory zaznamenaly úspěch. Nejčastěji byly Hybridní prediktory složeny z META prediktoru a kombinace prediktorů Gshare a Bimodal nebo Gshare a Lokální prediktor. Lze říct, že McFarlingova práce byla první impulz pro stavbu prediktorů složených z více komponent.

Gshare prediktor



Graf č. 3 – Přesnost Gshare prediktoru

Velikost	2-Bitová varianta		
	Globální prediktor	Gshare prediktor	Bimodální prediktor
8 B	22.520	22.373	18.98
16 B	20.659	19.966	16.563
32 B	18.891	17.486	14.812
64 B	16.588	15.533	13.476
128 B	15.046	13.613	12.438
256 B	13.579	12.020	11.651
512 B	12.396	10.585	11.141
1 KB	11.327	9.398	10.818
2 KB	10.416	8.440	10.622
4 KB	9.618	7.683	10.512
8 KB	8.851	7.102	10.443
16 KB	8.232	6.662	
32 KB	7.795	6.338	
64 KB	7.329	6.108	
128 KB	7.027	5.959	
256 KB	6.786	5.859	
512 KB	6.591	5.795	
1 MB	6.429	5.748	

Velikost	3-Bitová varianta	
	Globální prediktor	Gshare prediktor
12 B	21.088	20.864
24 B	19.347	18.686
48 B	17.648	16.499
96 B	15.678	14.739
192 B	14.195	12.995
384 B	12.890	11.525
768 B	11.797	10.196
1,5 KB	10.832	9.080
3 KB	9.982	8.176
6 KB	9.240	7.447
12 KB	8.508	6.887
24 KB	7.918	6.453
48 KB	7.496	6.128
96 KB	7.058	5.893
192 KB	6.768	5.738
384 KB	6.535	5.633
768 KB	6.355	5.571
1,5 MB	6.203	5.526

Tabulka č. 1 – Hodnoty MPKI pro globální, Gshare a Bimodální prediktor o různých velikostech

5. DE-ALIASED PREDIKTORY

Ačkoliv Gshare prediktor přinesl malou revoluci v oblasti predikce větvení, stále zde byl velký prostor pro zlepšení.

Graf č. 4 pochází od Pierra Michauda z jeho práce „An Alternative TAGE-like Conditional Branch Predictor“ [4] z roku 2018. V této práci představil BATAGE prediktor, nejpokročilejší prediktor do dnešního dne. Zároveň v této práci krásně shrnul historii výzkumu v oblasti predikce větvení a jednotlivé prediktory zanesl do grafu rovnou ve třech velikostech – 8 KB, 32 KB a 64 KB. Jelikož můj rozpočet je 32 KB, vybral jsem graf právě pro tuto velikost.

Zeleným čtvercem jsem označil takzvané období „De-aliased“ prediktorů – inženýři se v tomto období snažili snížit počet konfliktů, ke kterému dochází v PHT. Pro porovnání je přítomen i Gshare prediktor nebo i hybridní prediktor složený z bimodálního a Gshare prediktoru (v grafu zaznačen bim/gshare).⁸

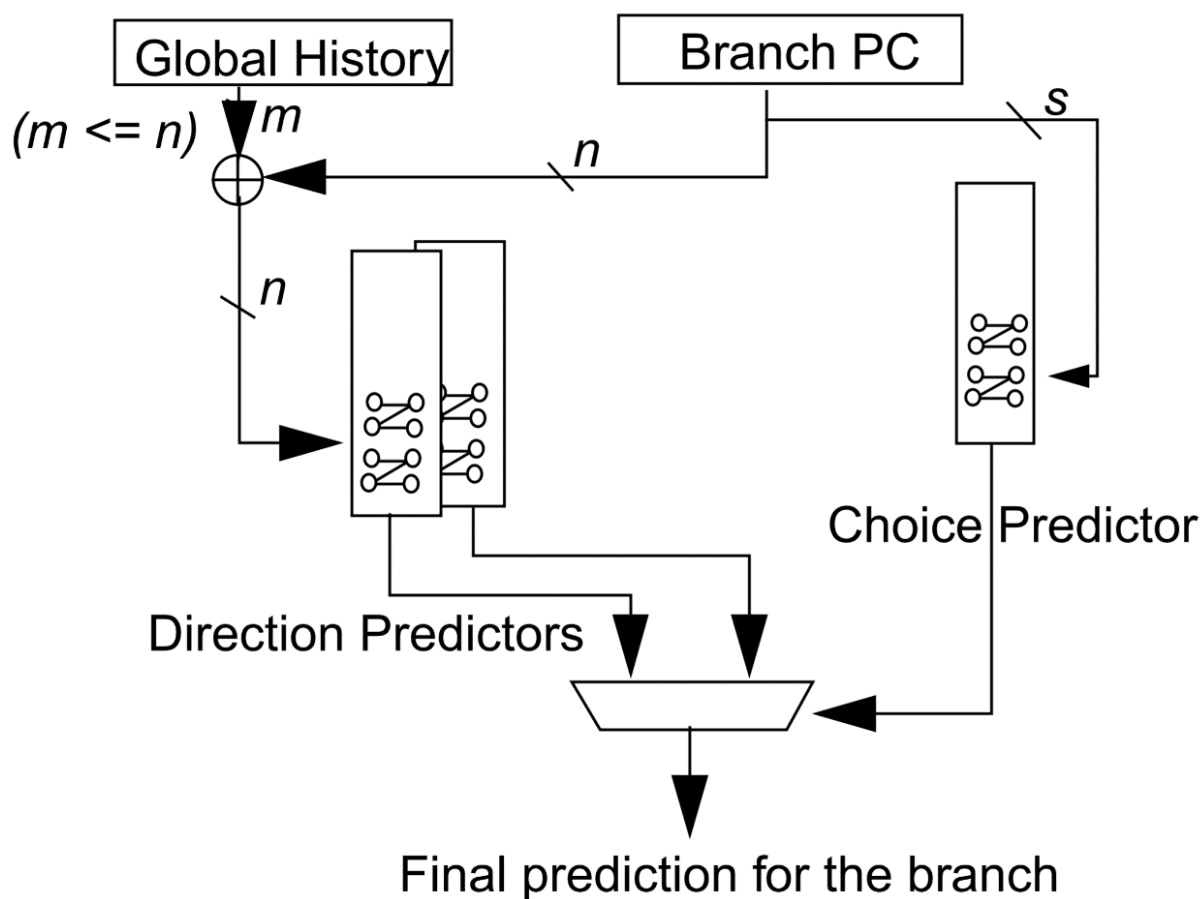


Schéma č. 4 – Bimode prediktor

⁸ Z grafu jde vidět, že MPKI pro Gshare prediktor je +-6.1, kdežto já dosáhl 6.338 – Pierre Michaud pravděpodobně použil pokročilejší indexovací funkci, proto tento rozdíl.

Bimode prediktor (zobrazen na schéma č. 4) funguje na principu rozdělení PHT – jedno PHT pro větve co bývají často přijaty (Taken PHT), druhé PHT je pro větve, které bývají často nepřijaty (Not Taken PHT).

Bimodální prediktor (na schématu pojmenován Choice predictor) pak vybírá, jestli bude pro predikci použit Taken nebo Not Taken PHT. Pokud Choice prediktor predikuje Taken, pro predikci se použije Taken PHT, v opačném případě se použije Not Taken PHT [5,6,7].

Rodina Gskew prediktorů má základní myšlenku podobnou – PHT je rozděleno na tři menší PHT, přičemž každé PHT je indexováno jinou indexovací funkcí. Pokud tedy dojde v jednom z PHT ke konfliktu, díky rozdílným indexovacím funkcím [13] nedojde ke konfliktu v ostatních PHT. Predikce je pak získána pomocí většinového hlasu (Majority vote) – pokud dva nebo více PHT hlasuje pro Taken, prediktor predikuje Taken. Naopak pokud dva nebo tři PHT hlasují pro Not Taken, prediktor predikuje Not Taken. [8]

Lze si to představit jako hlasování poroty v soutěži, porota má 3 členy (3 PHT), pokud dva nebo tři porotci hlasují pro postup soutěžícího, soutěžící postoupí. Naopak pokud většina poroty hlasuje pro konec soutěžícího, soutěžící ze soutěže odejde.

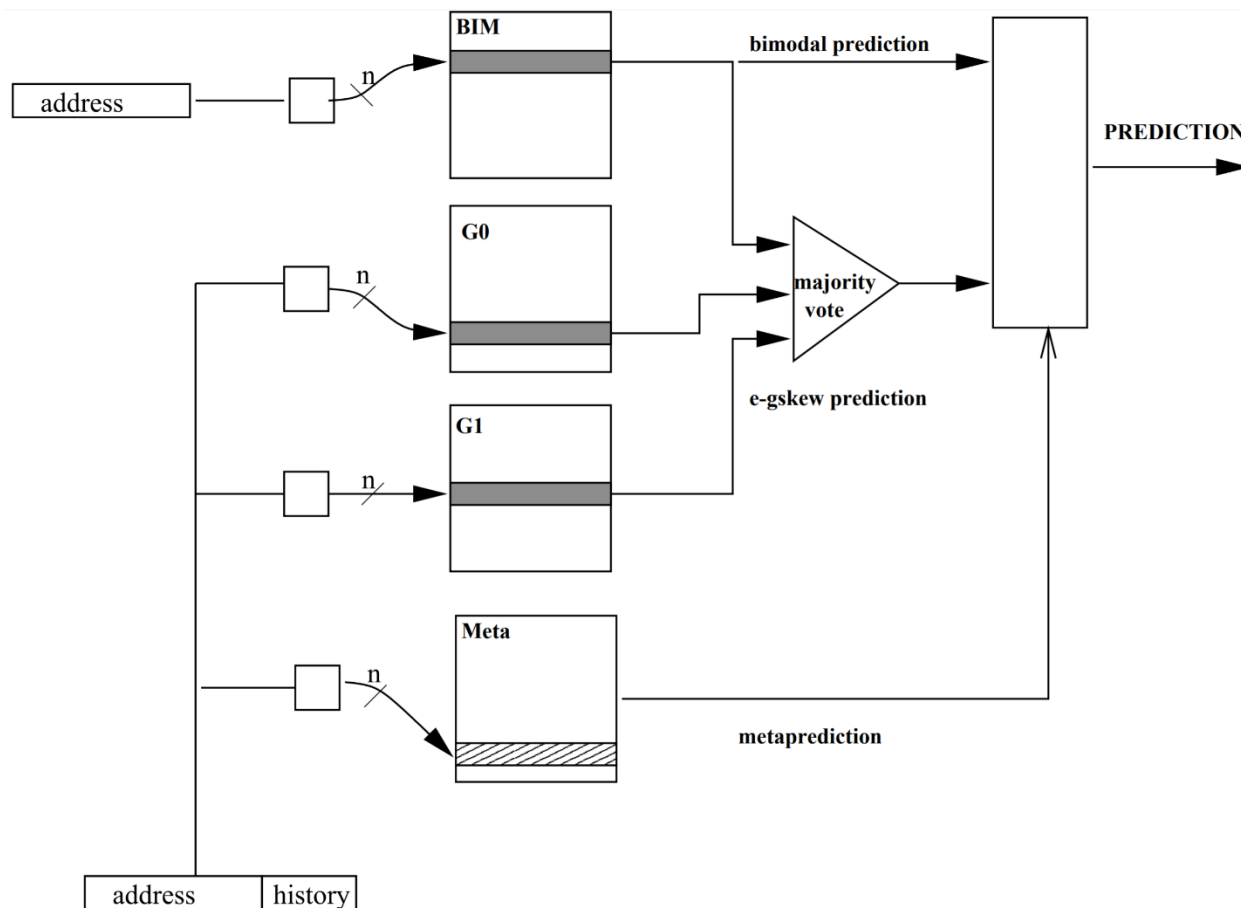


Schéma č. 5 – 2bc-gskew prediktor

Tento prediktor byl rozvíjen v mnoha výzkumných pracích. Nejdříve bylo jedno z PHT vyměněno za bimodální prediktor (e-gskew prediktor, zobrazen v grafu č. 4). Následovala integrace META prediktoru (2bc-gskew prediktor, zobrazen na schéma č. 5 a taktéž i v grafu), který vybíral mezi predikcí bimodálního prediktoru nebo predikcí přicházející z Majority vote. Vývoj tohoto prediktoru vyvrcholil prediktorem 2bc-gskew EV8 – o tomto prediktoru jsem mluvil v úvodu jako o mém cíli k překonání. [9, 10, 11]

Bohužel u simulace 2bc-gskew EV8 prediktoru jsem narazil na menší problém – jeho autor (André Seznec) zpřístupnil kód tohoto prediktoru na svých webových stránkách. Tento kód jsem předělal tak, aby bylo možné prediktor nasimulovat s infrastrukturou soutěže CBP 2016.

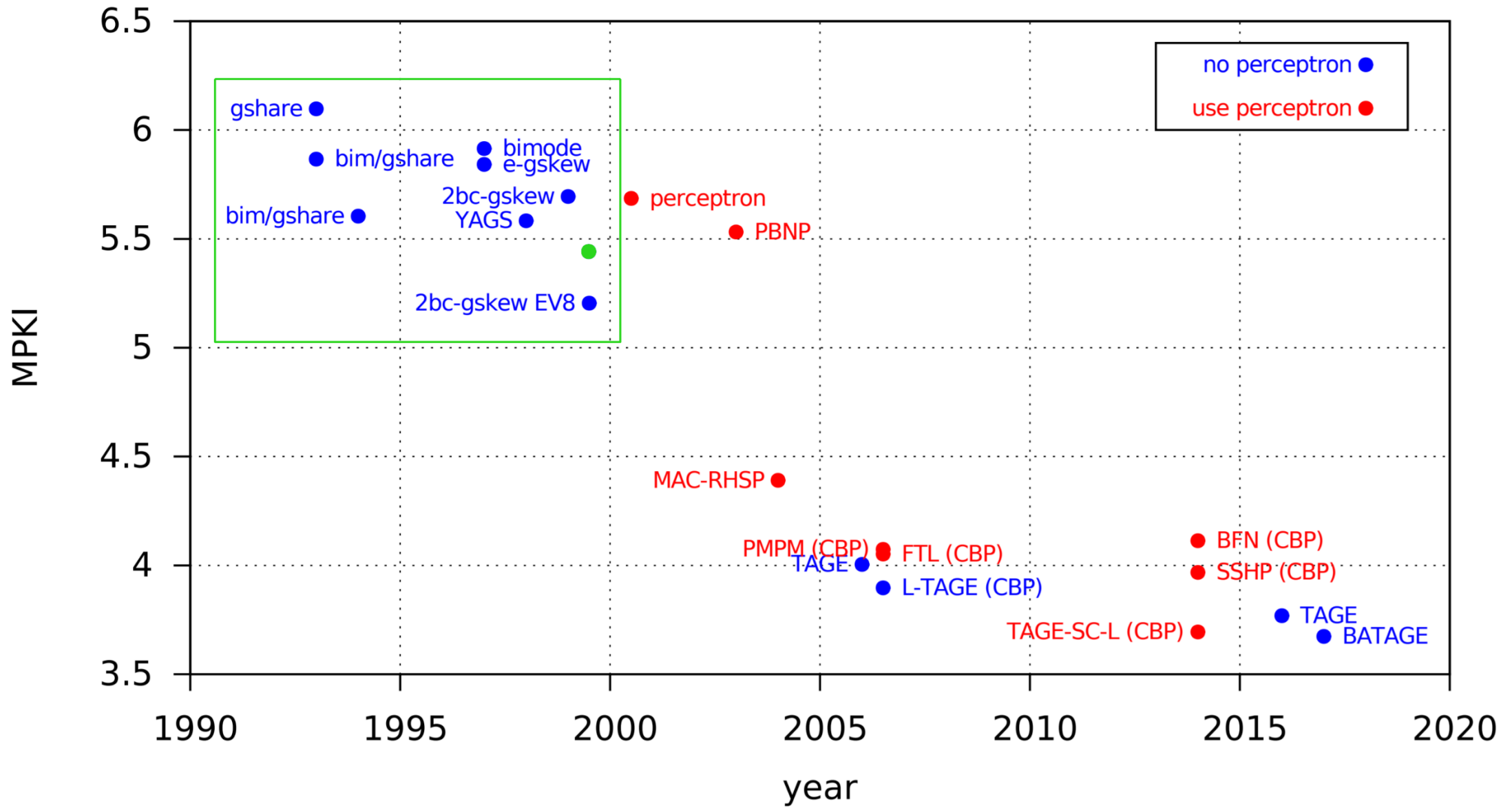
Dosáhl jsem MPKI 5.485, ale jak lze vidět z grafu, Pierre Michaud dosáhl cca MPKI 5.200.

Napadly mě dvě možnosti, jak mohl tento rozdíl vzniknout – buď jsem při předělání kódu udělal chybu, nebo Pierre Michaud použil upravenou verzi tohoto prediktoru.

Napsal jsem proto Pierru Michaudovi a ten potvrdil mou domněnku – v grafu jsou výsledky pro upravený 2bc-gskew EV8 prediktor. Zároveň mi i zaslal jeho upravený kód – po otestování jsem dosáhl shodného výsledku jako Pierre Michaud, a to 5.205 MPKI. Tímto mu velmi děkuji za jeho ochotu.

Do grafu jsem zanesl zelený puntík – přesnost originálního prediktoru 2bc-gskew EV8, 5.485 MPKI. Jako svůj cíl jsem si dal dosáhnout MPKI 5.400.

32KB



Graf č. 4 – Vývoj prediktorů

6. MODERNÍ PREDIKTORY

Tato kapitola je plně přeskočitelná, netýká se mojí práce a je určena pro ty, koho zajímá, jak to s predikcí větvení vypadá dnes.

Okolo roku 1999-2000 byl objeven Perceptronový prediktor – místo 2-bitových saturujících counterů byly použity perceptrony. Perceptron je nejjednodušší forma umělé inteligence. Ačkoliv perceptron nezpůsobil revoluci, co se týče dosažené přesnosti, byla to jediná práce, která se nesnažila vylepšit prediktory založené na 2-bitových counter, ale představila kompletně odlišné schéma predikce.

Nicméně perceptrony se ze začátku kvůli své složitosti nepoužívaly, poprvé se v reálných procesorech objevily okolo roku 2010. Prediktor založený na perceptronech obsahují jádra AMD Bulldozer (a na něm založená jádra Steamroller, Jaguar, Puma, Vishera, Excavator...) – tyto ne moc úspěšné procesory znáte pod jménem AMD FX. Perceptronový prediktor používá i jádro AMD Zen 1 – procesory Ryzen 1000 a 2000. Perceptronový prediktor používá i Samsung ve svých procesorech pro telefony.

YAGS byl první prediktor, který použil tagování (princip známý z pamětí Cache). Následoval ho PPM prediktor. Ten položil základ pro TAGE prediktor, který byl představen v roce 2006 a udělal velký průlom. TAGE prediktor vyhrál následující 4 ročníky soutěže CBP (2006-2016) a dlouho byl považován za to nejlepší.

Okolo roku 2017-2018 se na pultech obchodů začaly objevovat první procesory používající TAGE prediktor (všimněte si 10-ti leté mezery mezi představením a použitím v procesoru). Tento prediktor je přítomný v jádru AMD Zen 2 – poměrně nově představené a úspěšné procesory Ryzen 3000. Dále je použit v serverových procesorech IBM Power 9.⁹

V soutěži CBP 2016 v kategorii 8 KB zvítězil TAGE-SC-L prediktor, jehož autorem je André Seznec a dosáhl MPKI 4.191. V kategorii 64 KB zvítězil taktéž TAGE-SC-L prediktor od André Sezneca s MPKI 3.344. V kategorii Unlimited (nelimitovaný rozpočet), která je čistě akademická a zkoumá limity predikce větvení, zvítězil znovu André Seznec, ale tentokrát s prediktorem MTAGE-SC. Dosáhl MPKI 2.284.

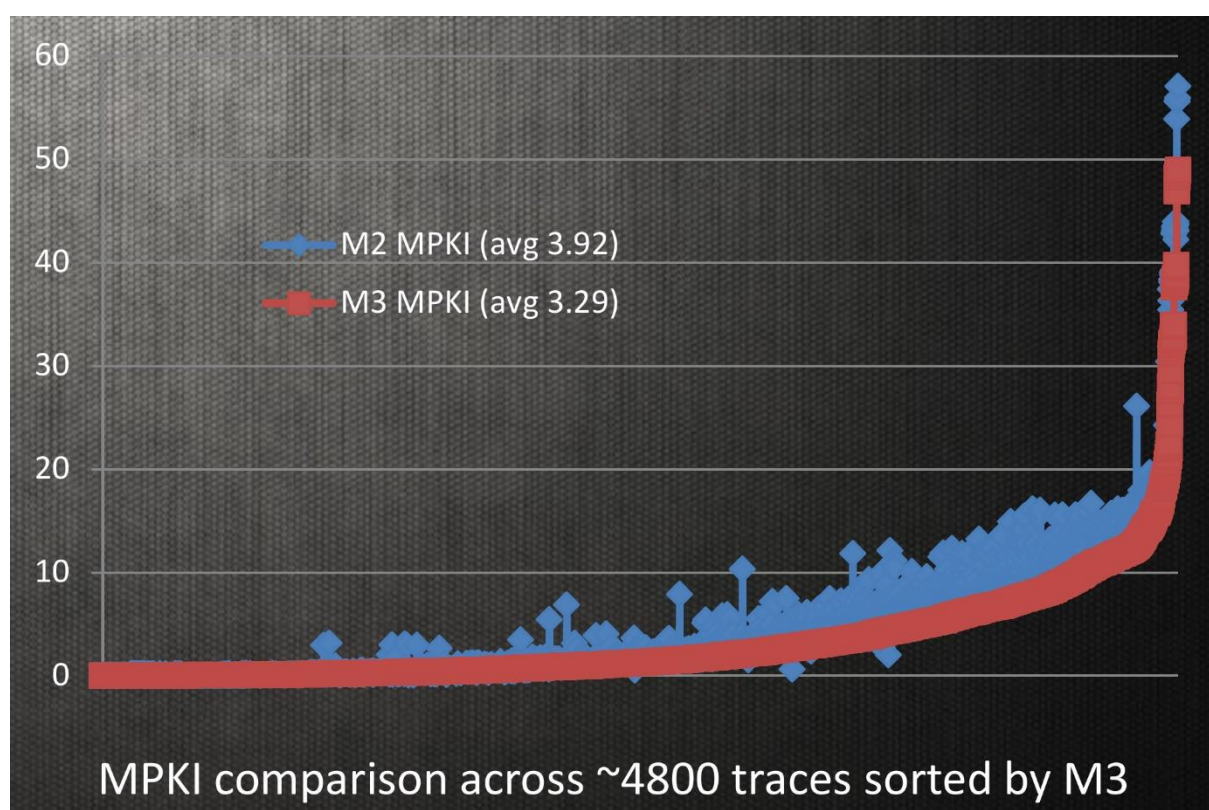
Poslední novinka na poli prediktorů větvení byla představena roku 2018 – BATAGE prediktor ^[4]. BATAGE prediktor (mimo jiné) řeší problém TAGE prediktoru s přiřazováním záznamů – TAGE prediktor neví, které záznamy byly nově přiřazeny a které naopak jsou přiřazeny delší dobu.

⁹ Pokud někomu vadí, že neuvádím informace o procesorech Intel, tak věřte, že bych je uvedl rád, ale Intel za posledních 10 let nepublikoval ani slovo o použitém prediktoru. Tradovalo se, že Intel použil TAGE prediktor již v jádrech Haswell, ale tato informace je neoficiální, a ještě k tomu podle mě nepravděpodobná.

To má za následek to, že se nově přiřazené záznamy nestihnou „zahřát“ (nějakou dobu trvá, než začnou podávat dobrou predikci) a kontrolní logika TAGE prediktoru tyto záznamy vyhodnotí za nepotřebné – a proto na jejich místo přiřadí nové záznamy.

BATAGE prediktor umí poznat, zda jsou záznamy „zahřáté“ nebo „studené“, a proto vždy počká, než se záznam zahřeje/zatrénuje a až poté vyhodnotí, zda se jedná o užitečný či neužitečný záznam a rozhodne o jeho ponechání nebo nahrazení.

Ačkoliv to tak nemusí z grafu č. 4 vypadat, tento prediktor má nad ostatními prediktory poměrně velký náskok – jeho nejbližší konkurent je TAGE-SC-L. Toto je vylepšená verze TAGE prediktoru, která používá statistickou korekci, loop counter, globální historii, lokální historii i path historii. Dalo by se říct, že se jedná o TAGE na steroidech. Zatímco BATAGE je schopný tomuto prediktoru konkurovat jen s použitím globální a path historie.



Graf č. 5 – přesnost jader M2 a M3 v MPKI

Zdroj: Prezentace „Samsung M3 Processor“ na konferenci Hot Chips 2018

Graf č. 5 pochází z konference Hot Chips, kde inženýři Samsungu představili jejich nové jádro – M3. Jádro M2 bylo použito v procesoru Exynos 8895, který byl srdcem telefonu Samsung Galaxy S8. Jádro M3 bylo použito v procesoru Exynos 9810, který poháněl Samsung Galaxy S9.

Prediktor v jádře M2 dosáhl v průměru MPKI 3.92, novější jádro M3 dosáhlo MPKI 3.29

Bohužel údaje z grafu č. 5 nejsou přímo porovnatelné s mými výsledky, protože inženýři Samsungu použili jiná testovací data (uvádějí použití ~4 800 traces, tj. 4800 testů). Tento graf tu uvádím pro představu, jakých výsledků dosahují prediktory nasazené v reálných produktech.

Moderní procesory neobsahují jen prediktor větvení – obsahují také prediktory cílových adres (BTB), prediktory nepřímých větví (toto jsou speciální větve, které nemají jen jednu cílovou adresu, ale mají jich hned několik) nebo prediktory návratových adres.

Možná si říkáte, zda jsou vůbec silnější prediktory větvení (nebo jiné prediktory) stále potřeba, když už nyní jsme schopni se dostat pod tři mispredikce na tisíc instrukcí – ve skutečnosti „hlad“ po silnějších prediktorech stále roste. Procesorová jádra jsou výkonnější a výkonnější, jsou schopná pracovat na více instrukcích najednou, a tudíž potřebují i silnější prediktory, aby stíhaly „krmit“ procesorová jádra.

7. KVN PREDIKTOR

Základní myšlenka mého KVN prediktoru je spojení Bimode a 2bc-Gskew prediktoru – nejdříve jsem PHT rozdělil na Taken a Not Taken, stejně jako v Bimode prediktoru. Potom jsem každý z těchto PHT rozdělil ještě jednou – po vzoru Gskew prediktoru používám dvojici PHT, kde každý je indexovaný jinou funkcí (funkce F0 a F1). Dále je přítomen Bimodální prediktor (BIM), který je indexovaný adresou PC. Výsledný prediktor lze vidět na schéma č. 6.

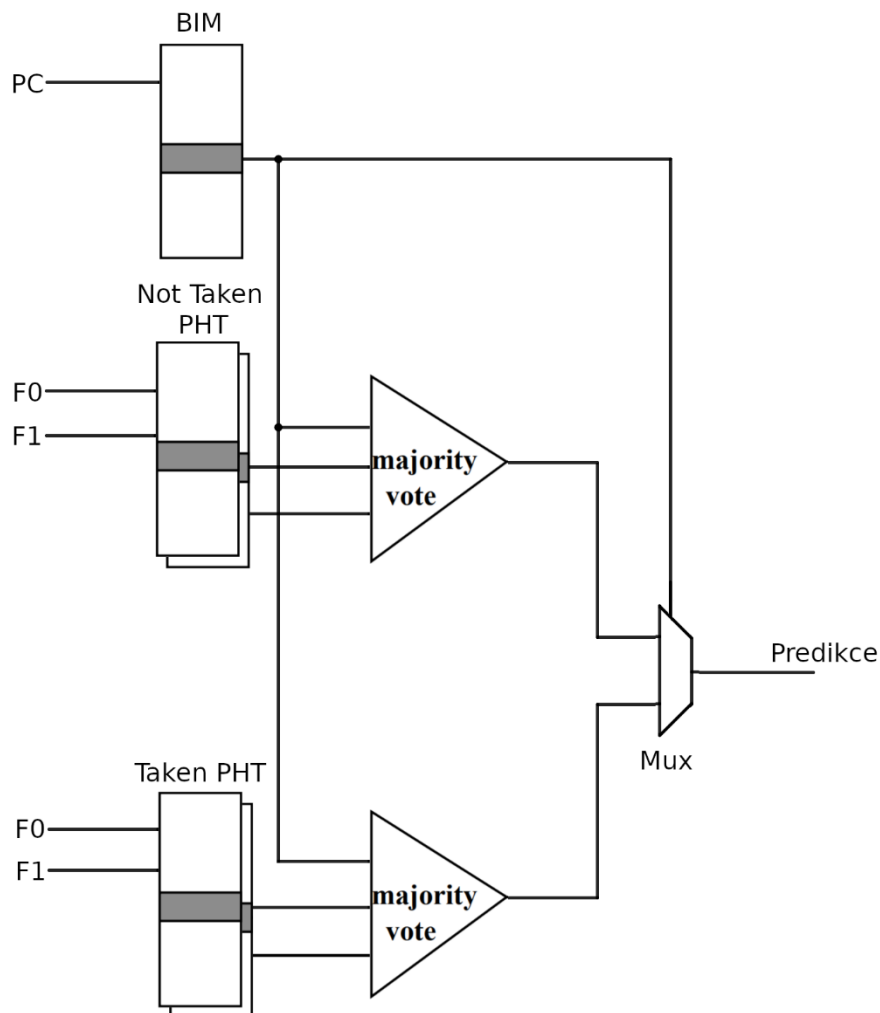


Schéma č. 6 – KVN prediktor

Pro zjednodušení vývoje jsem ze začátku vynechal META prediktor a soustředil se na vývoj KVN prediktoru bez něj. MPKI tohoto prediktoru se pohybovalo někde okolo MPKI 6.

Velký problém tohoto prediktoru spočívá v malé velikosti PHT – zatímco 2bc-gskew EV8 prediktor používá jedno PHT o velikosti 2^{17} bitů, KVN má rovnou čtyři PHT o velikosti 2^{14} – osmkrát menší velikost PHT.

Proto jsem se rozhodl sloučit obě Taken PHT a zároveň sloučit i obě Not Taken PHT – schéma č. 7. Tento krok má tři efekty:

1. Zdvojnásobení velikosti PHT na 2^{15} bitů přináší zlepšení přesnosti
2. Sloučení dvou PHT znamená zdvojnásobení počtu konfliktů uvnitř nového PHT (sloučené komponenty se bijí o záznamy). Tento efekt měl naštěstí zanedbatelný vliv na přesnost díky tomu, že KVN prediktor je vůči konfliktům dost odolný – díky tomu, že se predikce počítá ze 3, respektive 5 komponent, nevadí, když v jedné nebo dokonce ve dvou komponentách dojde ke konfliktu. Dalo by se říct, že se jedná o určitou formu redundance.
3. Zjednodušení prediktoru.

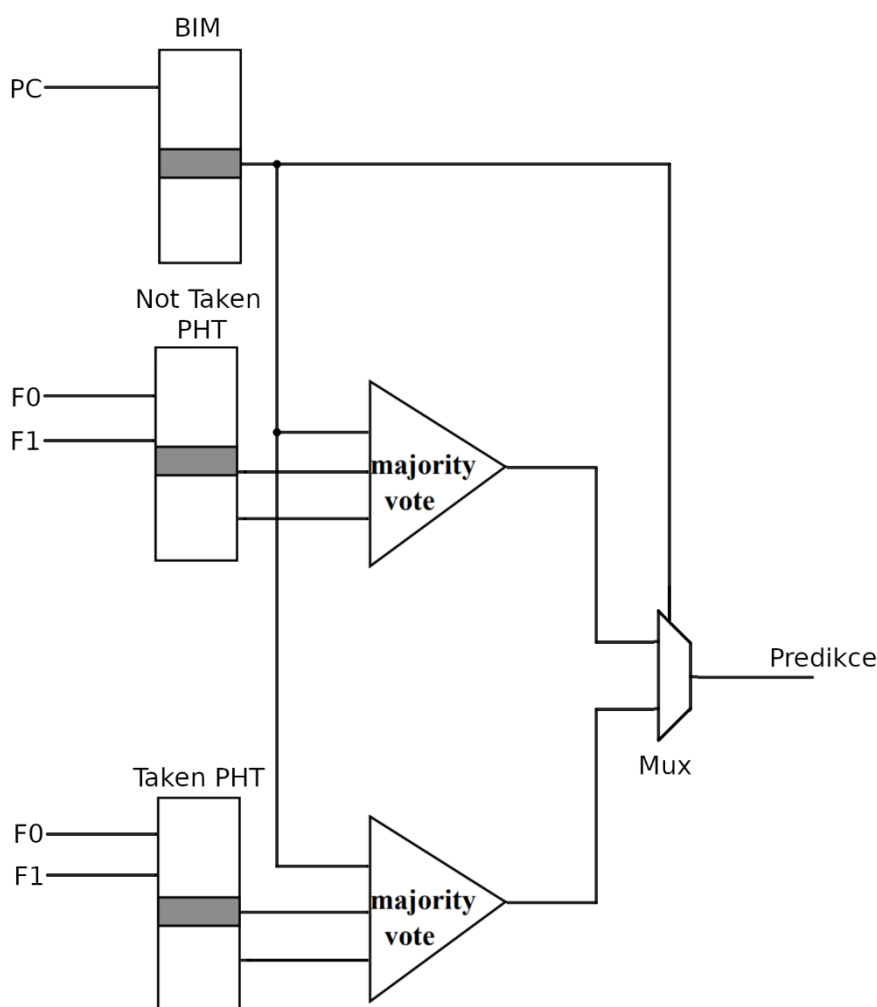


Schéma č. 7 – Prediktor KVN po sloučení PHT

Zvažoval jsem i sloučení Taken PHT a Not Taken PHT – po dlouhém testování jsem usoudil, že tento krok se nevyplatí – penalizace za dvojnásobný počet konfliktů vždy překonala získanou přesnost.

Dalším krokem bylo použití Hystereze ^[19]. Princip hystereze spočívá ve sdílení bitů mezi více záznamy – díky tomu dojde k ušetření velkého množství bitů, které můžu investovat jinak. Se sdílením bitů mezi záznamy jde ruku v ruce riziko vzniku konfliktů mezi záznamy. Pokud je hystereze implementována správně, dojde k minimálnímu snížení přesnosti (záleží na konkrétní implementaci, ale bavíme se o zhoršení cca 0.060 MPKI). Cena za jeden 2-bitový counter ale může klesnout ze 2 bitů například na 1.25 bitu. Díky ušetřené kapacitě jsem byl schopný zdvojnásobit velikost všech PHT na 2^{16} – což více než vynahradí přesnost ztracenou kvůli hysterezi.

7.1 CONFIDENCE LEVEL

V průběhu vývoje prediktoru mě napadlo využít tzv. confidence level (míru jistoty).

Má myšlenka vypadala nějak takto – zjistit, jak moc si je prediktor jistý svou predikcí, a na základě toho:

1. Pokud si prediktor je hodně jistý, využije standardní majority vote z 3 komponent (Bimodální prediktor a dvě predikce z Taken/Not Taken PHT).
2. Pokud si prediktor není moc jistý, využije se 5 komponentový majority vote z Bimodálního prediktoru a všech čtyřech PHT.

Jediná otázka byla, jak zjistit jistotu prediktoru? Vyzkoušel jsem hned několik možností, dokonce jsem zvažoval i využití perceptronu. Nakonec se jako nejlepší řešení ukázala nejjednodušší možnost – využít informaci, kterou nám vlastně sám od sebe poskytuje 2-bitový counter.

Pokud se znovu podíváme na přechodové schéma 2-bitového counteru (obrázek č. 7), je vidět, že dva prostřední stavy jsou označeny „Weakly“, což má označovat slabou jistotu. Zatímco dva krajní stavy jsou označeny „Strongly“, což znamená vysokou jistotu. Pokud se tedy counter nachází v jedné ze dvou prostředních pozic, víme, že si svou predikcí není moc jistý.

Takto vylepšený prediktor lze vidět na schéma číslo 8. Pokusil jsem se graficky znázornit, který majority vote je kdy použit pro predikci. Mohou nastat tyto tři stavy:

1. Pokud se bimodální prediktor nachází ve stavu „Strongly not Taken“ (červený puntík), je si velmi jistý, že větev nebude přijata, použije se predikce přicházející z majority vote označeného taktéž červeným puntíkem.
2. Když se bimodální prediktor nachází ve stavu „Strongly Taken“ (zelený puntík), je si velmi jistý, že větev bude přijata, použije se predikce z majority vote označeného zeleným puntíkem
3. Pokud se bimodální prediktor nachází v jednom z prostředních stavů (modrý puntík), tak si prediktor svou predikcí není moc jistý, a proto bude použit pěti-komponentový majority vote označený modrým puntíkem

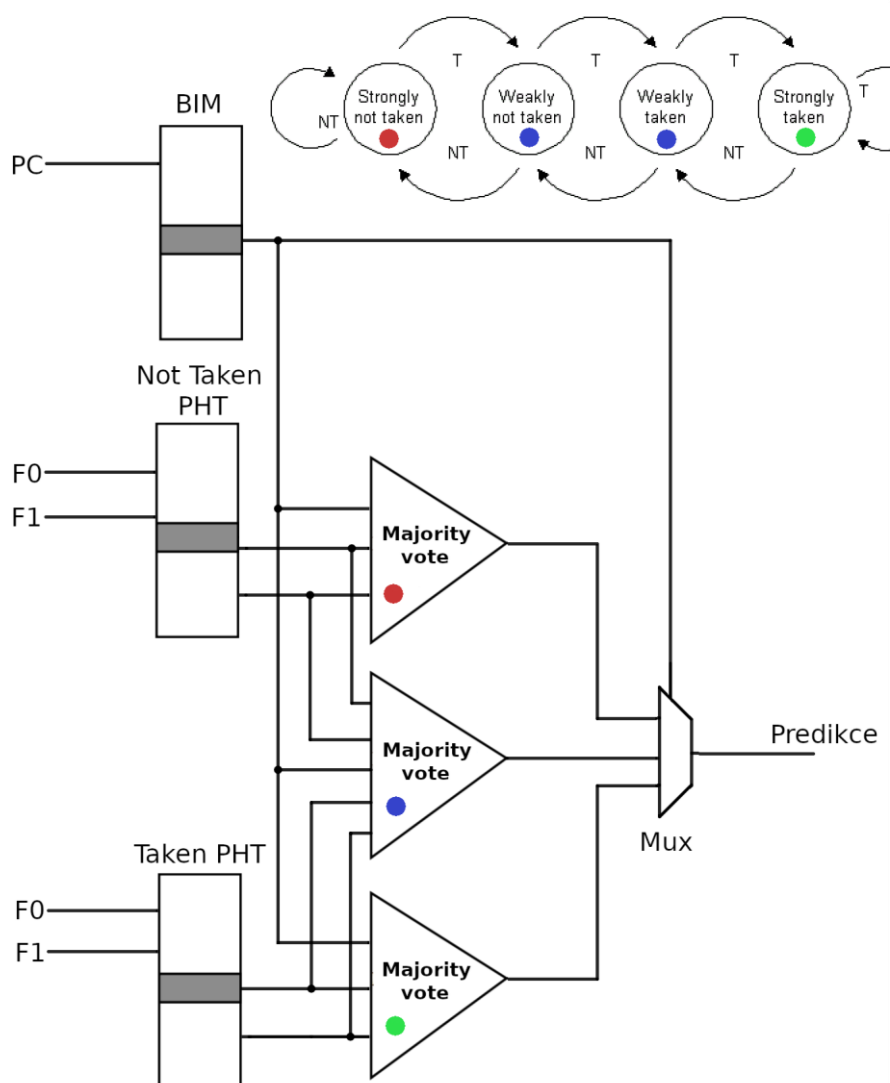


Schéma č. 8 – Prediktor KVN po implementaci 5 komponentového Majority Vote

A jak toto vylepšení fungovalo? Byl jsem sám překvapen, když jsem hned po první simulaci bez jakéhokoliv ladění dosáhl zlepšení o 0.1 MPKI. Jedná se o opravdu krásný výsledek, protože toto vylepšení mě nestálo jediný bit paměti, dá se říct, že bylo „zadarmo“.

7.2 META PREDIKTOR

Ačkoliv nerad, nakonec jsem se rozhodl pro přidání META prediktoru – i přes to, že sama o sobě tato komponenta nezvýšila přesnost (ale za to stála hromadu bitů a nabourala rozpočet), tak díky jeho použití KVN prediktor lépe škáloval s délkou historie (použití delší historie se najednou stalo výhodnější), a tak META prediktor nepřímým způsobem zlepšil přesnost.

META prediktor vždy vybírá, jestli predikce bude pocházet z Bimodálního prediktoru (BIM na schéma č. 8) nebo jestli bude vypočítána z většinových hlasů.

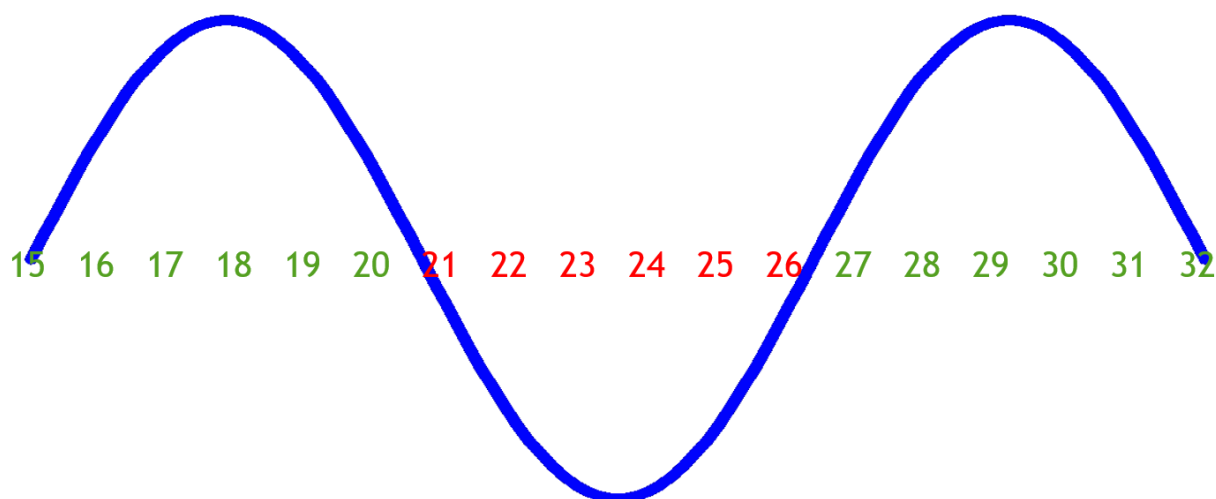
7.3 NASTAVENÍ HISTORIE

KVN prediktor má čtyři PHT a pro všechny čtyři lze separátně nastavit množství historie, které používají. Původně jsem si myslel, že se bude jednat pouze o výhodu, ale později se ukázalo, že se jedná o dvojsečnou zbraň.

Nastavení historie si totiž žádá obrovské množství simulací. Množství historie se může pohybovat kdekoliv od 15 do 64 (teoreticky i méně než 15, ale to jsou nepraktické hodnoty). Jelikož mám 4 nastavitelné parametry, dělá to 5 764 801 variací. Otestovat tolik variací je nemyslitelné, proto jsem dlouho přemýšlel, jak tento problém zjednodušit.

Ačkoliv se na první pohled při testování historie zdá, že je snadné nalézt ideální kombinaci, opak je pravdou. Možná Vás napadá myšlenka, že budete zvedat jeden parametr, najdete jeho ideální hodnotu, zapíšete a pustíte se do testování druhého parametru – podobně najdete ideální hodnotu pro třetí a čtvrtý parametr. Jenže když pohnete s druhým, třetím a čtvrtým parametrem, změní se ideální hodnota prvního parametru. Mám velké množství nasimulovaných dat (zhruba 100 hodin simulací), kde jsem podobnou taktiku zkoušel, a můžu vám říct, že to vede k minimálním ziskům v přesnosti a je to strašně neefektivní postup.

Později jsem ještě k tomu zjistil, že jednotlivé parametry se nechovají lineárně, ale spíše jako sinusovka. Co tím myslím? První parametr například může podávat dobré výsledky v rozmezí hodnot 15-20, v rozmezí 21-26 pak podává špatné výsledky, ale v rozmezí 27-32 zase záhadně začne nabírat na přesnosti. Toto chování je ilustrováno na obrázku č. 8.



Obrázek č. 8 – Znázornění chování jednotlivých parametrů

Řešení (ačkoliv ne úplně dokonalé) nakonec bylo mnohem jednodušší – držet se mocnin dvojky (např. 16, 32, 64) nebo mezikroků mezi těmito mocninami (24, 48). Toto masivně zredukuje počet kombinací a zároveň dosahuje až překvapivě uspokojivých výsledků.

Použité indexovací funkce umožňují měnit množství historie i pro Bimodální a META prediktor. Finální hodnoty historie jsou:

- Taken PHT 1–24 bitů
- Taken PHT 2–64 bitů
- Not Taken PHT 1–20 bitů
- Not Taken PHT 2–48 bitů
- META – 11 bitů
- BIM – 6 bitů

7.4 AKTUALIZAČNÍ POLITIKA

Pokud si přečtete výzkumné práce týkající se 2bc-gskew prediktoru, v každé z nich je použita aktualizací politika tzv. Partial Update – částečné aktualizování, pomáhá zmenšit počet konfliktů tím, že definuje pravidla, kdy se určité komponenty nebudou aktualizovat. Například pokud se všechny komponenty shodnou na predikci, tak tyto komponenty nemusí být aktualizovány. Nebo pokud se 2 ze 3 komponent shodnou na predikci, tak třetí komponenta nemusí být aktualizována.

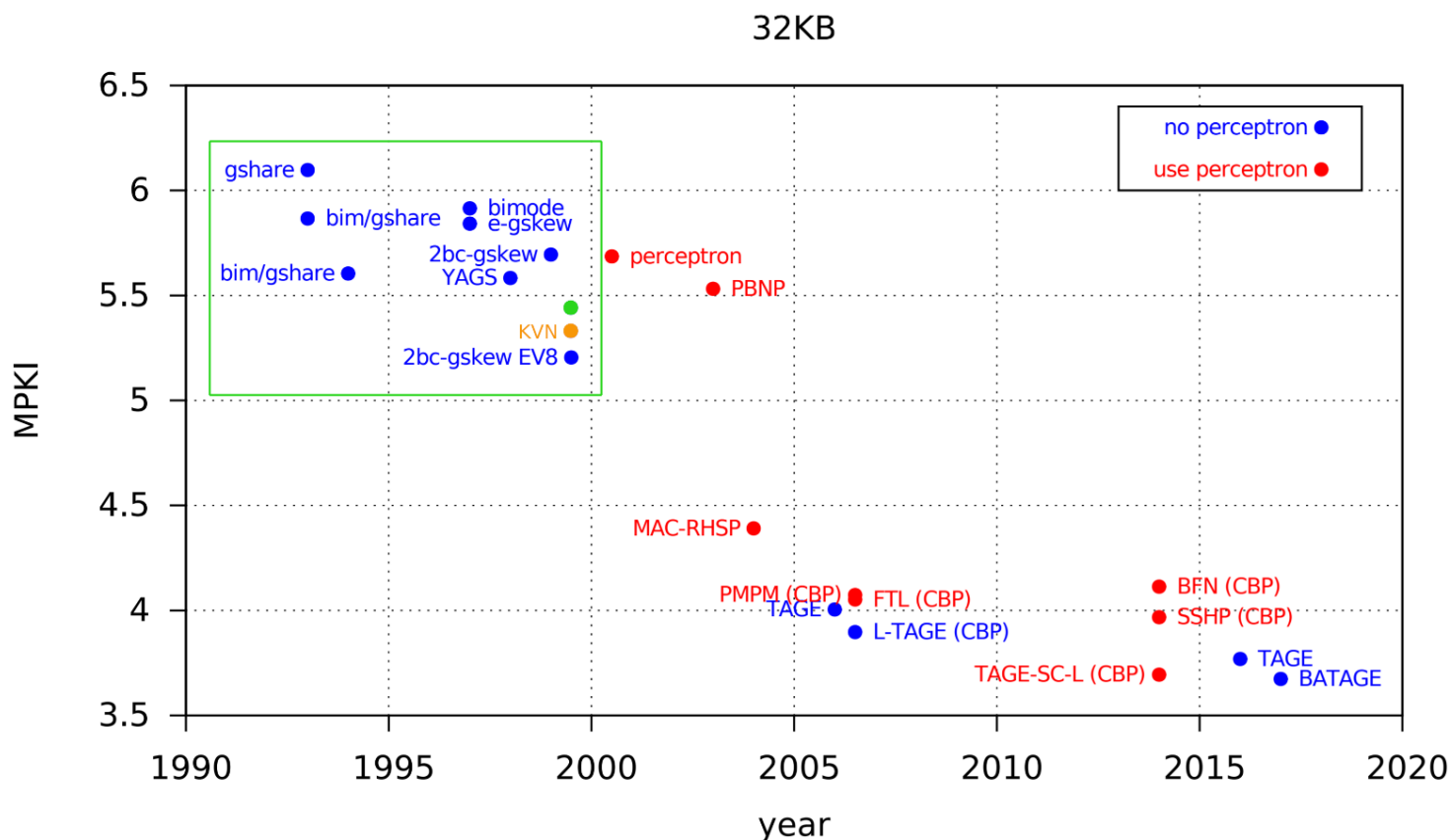
Experimentoval jsem s různými formami partial update, ale ve výsledku ani jedna z nich nepředstavovala pro můj KVN prediktor velký přínos (často spíš negativní) – domnívám se, že je to proto, že KVN prediktor je vůči konfliktům v PHT dost imunní – díky tomu, že se predikce počítá ze 3 respektive 5 komponent, nevadí, když v jedné nebo dokonce ve dvou komponentách dojde ke konfliktu. Díky této odolnosti není potřeba využívat částečné aktualizování.

7.5 VÝSLEDKY

Než vám konečně představím přesnost KVN prediktoru, sluší se uvést velikost jednotlivých komponent:

- Sdílené Not Taken PHT má 131 072 bitů
- Sdílení Taken PHT má 65 536 bitů
- Hystereze pro Taken PHT a Not Taken PHT má 32 768 bitů
- Sdílené pole pro META a Bimodální prediktor má 32 768 bitů

V součtu to dělá 262 144 bitů neboli 32 KB.



Graf č. 6 – Porovnání finálního KVN prediktoru s ostatními prediktory

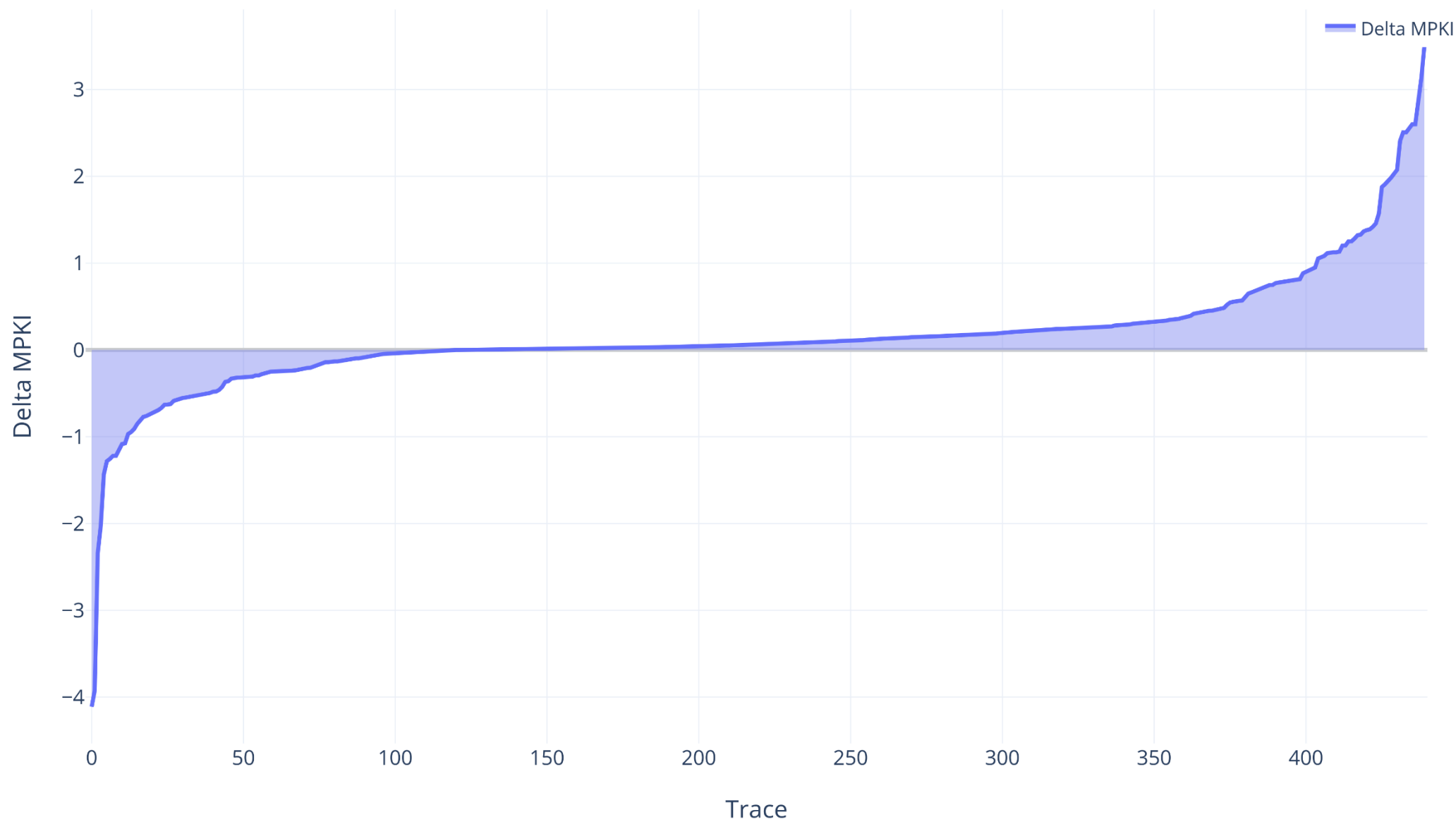
V této konfiguraci dosáhl KVN prediktor **5.337 MPKI** (zanesen do grafu č. 6 jako oranžová tečka). Tato hodnota je zhruba mezi originálním 2bc-gskew EV8 prediktorem a upravenou verzí od Pierra Michauda.

V grafu č. 7 pak můžete vidět rozdíl v dosaženém MPKI mezi KVN a 2bc-gskew EV8 v jednotlivých 440 testech (tak zvané Trace).

Osa Y představuje rozdíl MPKI mezi KVN a 2bc-gskew EV8, na ose X pak jsou výsledky jednotlivých testů – seřazeny od nejhoršího výsledku KVN až po nejlepší.

Můžete vidět, že zhruba v prvních 100 testech vítězí 2bc-gskew EV8, pak následují testy 100 až 200, kdy se jedná o remízu. V testech 200 až 440 (240 testů) pak vítězí KVN prediktor.

KVN x 2bc-gskew EV8



Graf č. 7 – Rozdíl v MPKI mezi KVN a 2-bcgskew EV8 v jednotlivých testech

8. JAK PREDIKTOR PLÁNUJI VYLEPŠIT V BUDOUCNU

Jak jste asi už pochopili, toto ještě není finální verze prediktoru. Ačkoliv jsem dosáhl svého cíle 5.4 MPKI, stále je co zlepšovat.

Jako první bych chtěl kompletně předělat indexovací funkce – momentálně používám indexovací funkce z 2bc-gskew EV8 prediktoru. Chtěl bych si vytvořit své vlastní indexovací funkce, ideálně aby využily jak globální historii, tak i Path historii ^[14] (alternativní druh historie, který pracuje s cílovými adresami).

Dále bych chtěl otestovat použití vlastních 3-bitových counterů v PHT – navrhnul jsem si dvě alternativy k standardním counterům, jejichž účelem je odstranit tzv. Ping Pong efekt ^[11], jak ho nazval André Seznec. Zároveň díky většímu počtu stavů ($2^3 = 8$ stavů) by z nich měl jít vypočítat přesnější confidence level.

Použití Confidence levelu se více než osvědčilo, chtěl bych na tomto konceptu zapracovat a počítat confidence level nejen z bimodálního prediktoru, ale zároveň i ze všech 4 PHT – tím získám mnohem více informací o tom, jak moc si je prediktor jistý predikcí. Eventuálně zvažuji možnost počítat confidence level pro každý majority vote zvlášť a pak zvolit nejjistější majority vote. Ideálně bych chtěl, aby tento princip úplně nahradil META prediktor – to by znamenalo ušetřený rozpočet (který se dá investovat jinak).

8.1 JAK PLÁNUJI V PRÁCI POKRAČOVAT

Mým dlouhodobým cílem je navrhnout vlastní procesor a následně ho nahrát na FPGA (Field Programmable Gate Array, programovatelné hradlové pole).

Pro svůj procesor jsem si zvolil instrukční sadu RISC-V (konkrétně její 32 bitovou verzi), a to hned z několika důvodů:

- Jedná se o perspektivní instrukční sadu, která se používá v reálných produktech.
- Tato instrukční sada je plně otevřená – její využití není svázáno žádnými poplatky a do jejího vývoje může promluvit kdokoli.
- Chtěl bych postavit plnohodnotný procesor, na kterém půjde spustit skoro jakýkoliv kód napsaný v jazyce C. Pro instrukční sadu RISC-V existuje hned několik kompilátorů, a to tento cíl značně zjednodušuje.
- Některé distribuce operačního systému Linux mají již nyní podporu RISC-V.

Momentálně jsem ve fázi, kdy už mám skoro hotovou představu o mém procesoru (co by měl splňovat) a pomalu se přesouvám k jeho vývoji.

9. ZÁVĚR

Splnil jsem svůj cíl a postavil prediktor větvení, který překonal MPKI 5.400 (konkrétně dosáhl MPKI 5.337, připomínám nižší MPKI = lepší) a zařadil se mezi jedny z nejlepších prediktorů dostupných v roce 2000. Ačkoliv tento prediktor nemá ambice dostat se do špičkových procesorů v počítačích nebo serverech, myslím si, že v oblasti mikrokontrolerů nebo jiných jednoduchých procesorů (např. do levných telefonů, televizí) se díky jeho jednoduchosti jedná o velmi zajímavou volbu.

Jak už jsem nastínil v kapitole 8, vývoj tohoto prediktoru pro mě určitě ještě není u konce. Až dokončím vývoj, chtěl bych si buď navrhnout vlastní procesor (viz. kapitola 8.1), nebo bych rád přistoupil na spolupráci s někým znalejším, kdo navrhuje vlastní procesorové jádro – na GitHubu se nachází poměrně velké množství procesorových jader, které často obsahují obyčejný Gshare prediktor. Mohl bych tedy aplikovat své zkušenosti s vývojem prediktorů a pokusit se implementovat KVN prediktor do procesoru.

Co mi tato práce přinesla? Vyzkoušel jsem si, jaké to je pracovat na složitém projektu a jak probíhá reálná vědecká práce – otestovat nějaký nový nápad, zanést data do grafu/tabulky a na základě získaných dat pak posoudit další postup – má smysl tento nápad rozvíjet? Není to jen slepá ulička? Pokud nový nápad přinesl zhoršení v MPKI, musel jsem zjišťovat proč – je to proto, protože se skutečně jedná o špatný nápad, nebo jsem ten nápad jen špatně implementoval?

Pokud jsem testoval koncept z některé starší vědecké práce (20-30 let), a moje výsledky se neshodovaly s výsledky z dané vědecké práce, musel jsem navíc zohlednit fakt, že výsledky z dané vědecké práce už nemusí být aktuální – nemuseli z dnešního pohledu použít relevantní testovací data. Pokud se podíváte na výzkumné práce z let 1995-2000, často hlásí například 20 % nárůst přesnosti oproti Gshare prediktoru, ale pokud stejný prediktor otestujete v infrastruktuře CBP 2016, dočkáte se 10 % zlepšení, ne-li menšího (tuto zkušenost potvrdil i Pierre Michaud ^[4]).

V průběhu jsem se zdokonalil i ve spoustě oblastech nespojených s prediktory větvení:

- Analýza získaných dat – přehledné zobrazení velkého množství dat v grafu
- Kritické uvažování – z mého pohledu naprosto klíčová vlastnost nejen pro vědeckou práci, ale i pro každodenní život. Vždy se na všechno nejdříve „dívat přes prsty“ a hledat možné mezery a chyby – ať už v mých datech získaných simulací, nebo v datech a tvrzeních uvedených v cizích vědeckých pracích.
- Programování – když jsem někdy před rokem a půl začínal s testováním, ve škole jsme stále neměli probrány funkce, makra, pole. Pro psaní prediktorů byly tyto znalosti klíčové, a tak jsem se je sám musel naučit.

- Optimalizace kódu – pomalost simulací mě donutila získat vědomosti z oblasti optimalizace kódu, nastavování optimalizačních vlajek pro kompilátor, dokonce jsem úspěšně vyzkoušel PGO (Profile Guided Optimization. PGO přineslo cca 10 % zrychlení simulací, ale bohužel ho nelze využít vždy). V budoucnu bych se chtěl nadále věnovat i tomuto tématu – vyzkoušet rozdíly v optimalizaci mezi různými kompilátory, otestovat vliv jednotlivých optimalizačních vlajek, naučit se provádět optimalizace přímo v assembly kódu a ručně vektorizovat kód.
- Zkušenosti – zde spadají různé nepopsatelné znalosti, které člověk získá pouze praxí.

Dva roky zpět jsem hltal různé výzkumné práce o predikci větvení z let 1990 a obdivoval práci vědců a inženýrů, kteří je psali. Dnes jsem schopný navrhnout a otestovat vlastní prediktor. To je podle mě obrovský úspěch, a proto jsem se svou prací spokojený.

10. POUŽITÁ LITERATURA

- [1] SMITH, J. E. *A STUDY OF BRANCH PREDICTION STRATEGIES*. Arden Hills, Minnesota, 1981.
- [2] YEH, T. Y. a Y. N. PATT. *Two-Level Adaptive Training Branch Prediction*. Ann Arbor, Michigan 48109-2122, 1991.
- [3] MCFARLING, Scott. *Combining Branch Predictors*. Palo Alto, Kalifornie, 1993.
- [4] MICHAUD, Pierre. *An Alternative TAGE-like Conditional Branch Predictor*. Inria, Univ Rennes, CNRS, IRISA, 2018.
- [5] I-CHENG K. CHEN, CHIH-CHIEH LEE a Trevor N. MUDGE. *The Bi-Mode Branch Predictor*. EECS Department, University of Michigan, 1997.
- [6] KISE, Kenji, Takahiro KATAGIRI, Hiroki HONDA a Toshitsugu YUBA. *The Bimode++ Branch Predictor*. PRESTO, Japan Science and Technology Agency (JST), 2005.
- [7] KISE, Kenji, Takahiro KATAGIRI, Hiroki HONDA a Toshitsugu YUBA. *Bimode-Plus Branch Predictor*. PRESTO, Japan Science and Technology Agency (JST), 2003.
- [8] MICHAUD, Pierre, Andre SEZNEC a Richard UHLIG. *Trading Conflict and Capacity Aliasing in Conditional Branch Predictors*. 35042 Rennes, Franc, 1997.
- [9] MICHAUD, Pierre a Andre SEZNEC. *De-aliased Hybrid Branch Predictors*. 35042 Rennes, Franc, 1999.
- [10] FELIX, Stephen, Andre SEZNEC, Venkata KRISHNA a Yiannakis SAZEIDE. *Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor*. 2002.
- [11] SEZNEC, Andre. *An optimized 2bcgskew branch predictor*. 35042 Rennes Cedex, France, 2003.
- [12] SEZNEC, Andre. *A case for two-way skewed-associative cache*. 1993.
- [13] SEZNEC, Andre a F. BODIN. *Skewed associative caches*. 1993.
- [14] NAIR, R. *Dynamic path-based branch correlation*. 1995.
- [15] JIMINEZ, D. A. a C. LIN. *Dynamic Branch Prediction with Perceptrons*. 2001.
- [16] JIMINEZ, D. A. a C. LIN. *Neural Methods for Dynamic Branch Prediction*. 2002.
- [17] EDEN, A. N. a T. MUDGE. *The YAGS Branch Prediction Scheme*. 1998.
- [18] KELLER, Jim. *Digital 21264 Sets New Standard*. 1996.
- [19] CHEN, I-Cheng K., John T. COFFEY a Trevor N. MUDGE. *Analysis of Branch Prediction via Data Compression*. 1996.
- [20] SPRANGLE, E., R. S. CHAPPELL, M. ALSUP a Y. N. PATT. *The Agree Predictor: Mechanism for Reducing Negative Branch History Interference*. 1997.

[21] *Agner Fog: Software optimization resources* [online]. [cit. 2020-03-24]. Dostupné z: <https://www.agner.org/optimize/>

[22] *Digital design and computer architecture*. 2nd ed. Waltham: Morgan Kaufmann, c2013. ISBN 978-0123944245.

[23] *Onur Mutlu Lectures* [online]. [cit. 2020-03-24]. Dostupné z: <https://www.youtube.com/channel/UCIwQ8uOeRFgOEvBLYc3kc3g>

[24] SCHOR, David. *WikiChip* [online]. [cit. 2020-04-18]. Dostupné z: <https://en.wikichip.org/wiki/WikiChip>

[25] *Wikipedia: Transistor count* [online]. [cit. 2020-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Transistor_count

11. SEZNAM OBRÁZKŮ, TABULEK, SCHÉMAT A GRAFŮ

11.1 SEZNAM OBRÁZKŮ

Obrázek č. 1 – Výhybka.....	9
Obrázek č. 2 – Die shot procesoru Ryzen 3xxx.....	10
Obrázek č. 3 – Blokové schéma jádra AMD Zen 2	11
Obrázek č. 4 – Jádro Zen 2 s jednotlivými vyznačenými bloky	12
Obrázek č. 5 – Spuštění simulace	13
Obrázek č. 6 – Výsledek simulací	14
Obrázek č. 7 – Přejchodový diagram 2bit Saturating Counteru	18
Obrázek č. 8 – Znázornění chování jednotlivých parametrů.....	39
Obrázek č. 9 – Soubor s výsledky KVN prediktoru.....	53

11.2 SEZNAM TABULEK

Tabulka č. 1 – Hodnoty MPKI pro globální, Gshare a Bimodální prediktor	27
--	----

11.3 SEZNAM GRAFŮ

Graf č. 1 – Přesnost Bimodálního prediktoru	21
Graf č. 2 – Přesnost Globálního prediktoru, osa Y – přesnost v MPKI	23
Graf č. 3 – Přesnost Gshare prediktoru	26
Graf č. 4 – Vývoj prediktorů.....	31
Graf č. 5 – přesnost jader M2 a M3 v MPKI	33
Graf č. 6 – Porovnání finálního KVN prediktoru s ostatními prediktory.....	41
Graf č. 7 – Rozdíl v MPKI mezi KVN a 2-bcgs skew EV8 v jednotlivých testech	42

11.4 SEZNAM SCHÉMAT

Schéma č. 1 – Bimodální prediktor	19
Schéma č. 2 – Globální prediktor	22
Schéma č. 3 – Gshare prediktor	25
Schéma č. 4 – Bimode prediktor	28
Schéma č. 5 – 2bc-gskew prediktor	29
Schéma č. 6 – KVN prediktor	35
Schéma č. 7 – Prediktor KVN po sloučení PHT.....	36
Schéma č. 8 – Prediktor KVN po implementaci 5 komponentového Majority Vote.....	38

12. SEZNAM POJMŮ A ZKRATEK

2bc-gskew EV8 – prediktor použitý v nedokončeném procesoru Alpha 21464 (někdy také znám pod kódovým označením Alpha EV8)

AMD, Intel, IBM – výrobci procesorů

Assembly – strojový jazyk

Binární soustava – číselná soustava se základem 2. Používá pouze číslice 1 a 0.

Bit – nejmenší jednotka paměti, nabývá hodnoty logická 1 nebo 0

Branch prediction – predikce větvení

BTB – paměť cílových adres

Byte – 8 bitů

CPU – Central Processing Unit, procesor

FPGA – Field Programmable Gate Array, programovatelné hradlové pole – čip, který lze nakonfigurovat tak, že realizuje jakoukoliv logickou funkci (někdy i celý procesor)

Indexovací funkce – funkce, která vezme určitý vektor vstupních informací (adresa PC, historie) a vytvoří z něj adresu pro pole counterů

Instrukce – příkaz pro procesor, např. „sečti hodnoty v registrech R5 a R8 a výsledek ulož do R12“ nebo „pokud je hodnota uložená v registru R3 větší než 0, skoč na adresu xyz“

Instrukční sada (ISA) – standard určující základní vlastnosti procesoru – počet registrů a jejich šířka v bitech, definuje jednotlivé instrukce. Příklady ISA jsou např. x86, MIPS, RISC-V, ARM

Hexadecimální soustava – číselná soustava se základem 16. Používá číslice 0 až 9 a písmena A-F

Kompilátor – program sloužící pro překlad kódu do strojového jazyka (assembly)

Logické hradlo – několik tranzistorů realizujících logickou funkci, např. AND, OR...

Majority Vote – většinový hlas, viz kapitola 5.

Mispredikce – chybně predikovaná větev

MPKI – Misspredictions Per Kilo Instruction – počet mispredikcí na tisíc instrukcí

Perceptron – algoritmus používaný ve strojovém učení, zjednodušený model neuronu

Prediktor větvení – digitální obvod uvnitř procesorového jádra

Registr – rychlá paměť uvnitř procesorového jádra

Tranzistor – elektronická součástka, založena na polovodičích

Větev – větev je programátorova podmínka (např. if, else nebo také cykly while a for)

x-bitový counter – viz kapitola 4.2

13. SEZNAM PŘÍLOH

Příloha č. 1 – Kód KVN prediktoru – soubor „predictor.h“

Příloha č. 2 – Výsledky simulací pro KVN prediktor – složka „KVN“

Příloha č. 3 – Výsledky simulací pro 2bc-gskew EV8 prediktor, složka „2bc-gskew_EV8“

14. DODATEK A – LEGENDA K PŘÍLOHÁM Č. 2 A Č. 3

Jednou z příloh mé práce (viz kapitola „Seznam příloh“) jsou soubory s výsledky simulací – tato kapitola slouží jako legenda pro ty, kteří si chtějí výsledky projít (jinak je tato kapitola jen doplňující), protože samotné soubory jsou špatně čitelné a nachází se v nich velké množství informací.

Ačkoliv mají soubory příponu „res“, lze je otevřít v normálním poznámkovém bloku – takový otevřený soubor je zobrazen na obrázku č. 9.

Standardně tyto soubory obsahují pouze informace, které se nachází na prvním řádku, já jsem si infrastrukturu trochu modifikoval, takže mi tiskne mnohem více informací (jsou to informace specifické pro můj KVN prediktor, proto výsledky 2bc-gskew EV8 prediktoru tyto informace neobsahují).

Bohužel na prvním řádku se nachází opravdu velké množství dat, a tak se všechna nevejdou na obrázek č. 9, proto Vám doporučuji otevřít si některý ze souborů s výsledky a procházet je během čtení této kapitoly.

Na prvním řádku se nachází tyto informace:

- „MPKBr_xx“ – počet mispredikcí na 1000 větví. Tento údaj se generuje průběžně během simulace – proto je vždy za podtržítkem uvedeno „kdy“ byl vygenerován. Například pokud je za podtržítkem „100K“, znamená to, že tento údaj byl vygenerován po zpracování 100 000 větví. Tyto údaje vypovídají o tzv. Warm up time (zahřívací čas, za jak dlouho se prediktor „zahřeje“ a začne podávat dobrý výkon). Doporučuji tyto informace ignorovat, mají pro nás malý význam.
- „TRACE“ – název a umístění testu (např. „.../traces/LONG_MOBILE-6.bt9.trace.gz“ znamená test „LONG_MOBILE-6“)
- „NUM_INSTRUCTIONS“ – počet instrukcí, které byly zpracovány
- „NUM_BR“ – počet větví, které byly zpracovány, tento údaj se dále dělí na podmíněné a nepodmíněné větve
- „NUM_UNCOND_BR“ – počet nepodmíněných větví, tzv. skoků (skok se chová jako větev, která je vždy přijata), které byly zpracovány
- „NUM_CONDITIONAL_BR“ – počet podmíněných větví (ty nás zajímají nejvíce, ty predikuje prediktor), které byly zpracovány
- „NUM_MISPREDICTIONS“ – počet mispredikcí (kolik chyb udělal náš prediktor)
- „MISPRED_PER_1K_INST“ – MPKI

Na zbylých řádcích (které jsou již všechny vidět na obrázku č. 9) jsou informace, které si tisknu speciálně pro můj KVN prediktor:

- „Predictor_accuracy“ – přesnost prediktoru v procentech
- „Meta dont care“ – META prediktor nemá jak ovlivnit predikci, protože predikce bimodálního prediktoru a predikce většinového hlasu se shodují
- „Meta choosed right“ – META prediktor zvolil dobře
- „Meta choosed wrong“ – META prediktor zvolil chybně
- „Meta choosed BIM“ – META prediktor zvolil predikci z Bimodálního prediktoru

- „Meta choosed Bigskew“ – META prediktor zvolil predikci z většinového hlasu
- „5 Component majorityVote was used“ – jak často byl zvolen 5 komponentový většinový hlas
- „5 Component majorityVote accuracy“ – jaká byla přesnost 5 komponentového většinového hlasu (přesnost většinou bývá nižší než u 3 komponentového většinového hlasu, protože 5 komponentový většinový hlas je většinou použit na ty těžší větve)
- „3 Component majorityVote was used“ – jak často byl použit 3 komponentový většinový hlas
- „3 Component majorityVote accuracy“ – přesnost 3 komponentového většinového hlasu
- „Taken PHT was used“ – jak často byly pro predikci použity Taken PHT – tento údaj doporučuji ignorovat
- „Not Taken PHT was used“ - jak často byly pro predikci použity Not Taken PHT – tento údaj doporučuji také ignorovat
- „Taken PHT accuracy“ – přesnost Taken PHT, taktéž doporučuji ignorovat
- „Not Taken PHT accuracy“ - přesnost Not Taken PHT, taktéž doporučuji ignorovat
- „Bimodal accuracy“ – přesnost Bimodálního prediktoru
- „Accuracy of xxxxx“ – přesnost (Not) Taken PHT $\frac{1}{2}$
- „Size“ – velikost prediktoru v bitech

```

1  MPKBr_1K          :    62.0000  MPKBr_10K          :    44.6000  MPKBr_100K          :    43.0100
2
3
4  Predictor_accuracy: 96.33719 %
5
6  META
7  Meta dont care   61.55615%
8  Meta choosed right 36.44900%
9  Meta choosed wrong 1.99485%
10
11 Meta choosed BIM  76.81080%
12 Meta choosed Bigskew 23.18920%
13 Mode
14 5 Component majorityVote was used 3.57168%
15 5 Component majorityVote accuracy 86.07319%
16 3 Component majorityVote was used 19.61752%
17 3 Component majorityVote accuracy 94.04961%
18
19 Taken PHT was used 11.61466%
20 Not Taken PHT was used 8.00286%
21 Taken PHT accuracy 91.84908%
22 Not Taken PHT accuracy 97.24326%
23
24 Bimodal accuracy 95.01028%
25 Accuracy of Taken PHT 1 99.00474%
26 Accuracy of Taken PHT 2 98.88673%
27 Accuracy of Not Taken PHT 1 99.43686%
28 Accuracy of Not Taken PHT 2 99.48990%
29 Size 262144
30
31
32
33

```

Obrázek č. 9 – Soubor s výsledky KVN prediktoru