Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Vliv zranitelností procesorů na výkon systému

Bc. Tomáš Havlík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Tomáš Havlík

Systémové inženýrství a informatika
Informatika

Název práce

Vliv zranitelností procesorů na výkon systému

Název anglicky

Influence of processors vulnerability at computing performance

---

Cíle práce

Cílem diplomové práce je analýza výkonu procesorů, při opravách zranitelností typu útok postranním kanálem (např. Spectre a Meltdown). Výkon bude porovnán pomocí testů na fyzických a virtualizovaných systémech.

Metodika

Teoretická část bude zpracována pomocí rešerše odborné literatury a publikací článků. Vysvětleny budou základní pojmy, jednotlivé chyby procesorů, jejich odstranění a boj proti nim. Dále bude provedeno odstranění chyb ve virtualizovaném systému.

V praktické části budou vytvořeny scénáře pro testování ztráty výkonu. Budou provedeny testy, které omezují vstupní a výstupní výkon ve fyzickém prostředí a vliv chyb na hyper-threading ve virtualizovaném prostředí. Dále budou vypínány jednotlivé služby počítače a testovány jaký mají vliv na výkon systému. Nakonec budou provedena jednotlivá měření a vyhodnoceny výsledky pro komparaci vlivu na ztrátu výkonu virtualizace.

Doporučený rozsah práce

50 – 60 stran

Klíčová slova

Procesory, Intel, bezpečnostní hrozby, virtualizace, Meltdown, Spectre

---

Doporučené zdroje informací

GIL DE LAMADRID, James. Computer organization: basic processor structure. Boca Raton: CRC Press,Taylor & Francis Group, [2018]. ISBN 978-1-4987-9951-5.

LIČEV, Lačezar a David MORKES. Procesory. Brno: Computer Press, 1999. Hardware (Computer Press). ISBN 80-7226-172-x.

RUEST, Danielle a Nelson RUEST. Virtualizace: podrobný průvodce. Brno: Computer Press, 2010. ISBN 9788025126769.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Marek Pícka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 9. 3. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 9. 3. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 29. 03. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Vliv zranitelností procesorů na výkon systému" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 6. 4. 2020 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) Ing. Marku Píckovi, Ph.D. za vstřícnost a připomínky na konzultacích při psaní této diplomové práce a za jeho cenné rady.
V neposlední řadě patří obrovské poděkování mé rodině, která mě vždy v mých studiích podporovala a bez které bych nemohl tuto práci dokončit.

Vliv zranitelností procesorů na výkon systému

Abstrakt

Diplomová práce analyzuje ztrátu výkonu procesorů při opravách zranitelností typu útok postranním kanálem ve fyzickém i virtualizovaném prostředí. V úvodní části autor uvádí základní pojmy a principy zranitelností procesorů, jednotlivé bezpečnostní hrozby a jejich vliv ve virtualizovaném prostředí. Následně představuje jejich odstranění a ochranu před těmito hrozbami.

V další části autor vytváří scénáře pro testování ztráty výkonu. Definuje jednotlivé testy, které omezují vstupní a výstupní výkon procesoru ve fyzickém prostředí a vliv chyb na hyper-threading ve virtualizovaném prostředí. Následně po vypíná jednotlivé služby počítače a otestuje jejich vliv na výkon systému. Nakonec provádí jednotlivá měření a vyhodnocuje výsledky pro komparaci vlivu na ztrátu výkonu virtualizace. Součástí této práce je také statistické měření jednoduché analýzy rozptylu a mnohonásobného porovnávání za účelem potvrzení prokazatelné ztráty výkonu.

Klíčová slova: Procesory, Bezpečnostní hrozby, Virtualizace, Hyper-threading, Meltdown, Spectre, Foreshadow, Ztráta výkonu, Zranitelnosti, Výkonové testy, Útok postranním kanálem, Jednoduchá analýza rozptylu, Mnohonásobné porovnávání, Databázové testy, Predikce skoků, Spekulativní vykonávání, Retpolíny, TLB Cache, Pipelining

Influence of processors vulnerability at computing performance

Abstract

The main goal of the Master's thesis is analysing the performance loss of processors in side channel attack in physical and virtualized environments. In the first part, the author introduces the basic concepts and principles of processors vulnerabilities, individual security threats and their impact in a virtualized environment. Further, author explaining their removal and protection against these threats.

In the next part author creates scenarios for performance loss testing. Author defines individual tests that limit processors input and output performance in a physical environment and the impact of errors on hyper-threading in a virtualized environment. Then author shuts down individual computers services and tests their impact on system performance. Finally, performs individual measurements and evaluates the results to compare the impact on virtualization performance loss. Master's thesis contains also statistical measurement of simple analysis of variance and multiple comparison in order to confirm demonstrable performance loss.

Keywords: Processors, Security Threats, Virtualization, Hyper-threading, Meltdown, Spectre, Foreshadow, Performance Loss, Vulnerability, Performance Tests, Side Channel Attack, Simple analysis of variance, Multiple comparisons, Database tests, Branch Prediction, Speculative Execution, Retpolines, TLB Cache, Pipelining

Obsah

1 Úvod ... **13**

2 Cíl práce a metodika ... **14**

2.1 Cíl práce ... 14

2.2 Metodika ... 14

3 Teoretická východiska ... **15**

3.1 Procesory ... 15

3.1.1 Výroba čipu procesoru ... 15

3.1.2 Stavba procesoru ... 16

3.2 Techniky zvyšování výkonu procesorů ... 18

3.2.1 Rozšíření bitové šířky zpracovávaných dat ... 19

3.2.2 Zvýšení počtu pracovních registrů ... 19

3.2.3 Hierarchické uspořádání paměti ... 19

3.2.4 Fronta instrukcí ... 20

3.2.5 Pipelining ... 21

3.2.6 Speculative execution ... 23

3.2.7 Branch prediction ... 24

3.2.8 Out of order execution ... 26

3.2.9 Indirect branch prediction ... 28

3.2.10 Return target predictor ... 29

3.2.11 TLB Cache a tabulka stránek ... 30

3.2.12 Simultánní Multi-threading a Hyper-threading ... 33

3.3 Bezpečnostní hrozby ... 33

3.3.1 Spectre ... 33

3.3.2 Meltdown ... 39

3.3.3 Foreshadow ... 41

4 Vlastní práce ... **43**

4.1 Výběr operačního systému a virtualizačního prostředí ... 43

4.2 Výběr komponent ... 43

4.2.1 Testovací komponenty ... 44

4.3 Příprava scénářů ... 45

4.4 Testy a naměřené hodnoty ... 47

4.4.1 Vstupní a výstupní výkon ... 48

4.4.2 Výkon procesoru ... 51

4.4.3 Výkon systému ... 56

4.5 Jednoduchá analýza rozptylu s mnohonásobným porovnáváním ... 62

4.5.1 Statistická testování 64

5 Výsledky a diskuse 74

5.1 Výsledky ztrát na výkon dle naměřených hodnot 74

5.2 Výsledky jednoduché analýzy rozptylu 77

6 Závěr 79

7 Seznam použitých zdrojů 81

8 Přílohy 84

8.1 Příloha A – Statistická testování Hackbench 84

8.2 Příloha B – Statistická testování Stress-NG – Stress CPU 93

8.3 Příloha C – Statistická testování Stress-NG – Context Switching 101

8.4 Příloha D – Přehled změn výkonu procesorů při zapnutých mitigacích 110

Seznam obrázků

Obrázek č. 1 - Blokové schéma mikroprocesoru [4] 20
Obrázek č. 2 - Jednotlivé kroky při zpracování instrukce [5] 21
Obrázek č. 3 - Tok instrukcí v procesoru pomocí pipeliningu [5] 22
Obrázek č. 4 - Stavový diagram dvou bitového prediktoru [6] 26
Obrázek č. 5 – Čtení dat z virtuálního adresního prostoru [20] 31
Obrázek č. 6 – Ukázka příkazů pro jednoduchou analýzu rozptylu v programu SAS [autor] 63
Obrázek č. 7 – Ukázka výstupu procedury Boxplot u procesoru i7-740Q [autor] 65
Obrázek č. 8 – Rozdělení četností ztrát výkonu u všech měřených testů, režimů a procesorů [autor] 75
Obrázek č. 9 – Procedura GLM pro fyzickou část testu Hackbench i7-740Q [autor] 84
Obrázek č. 10 – Procedura Boxlpot pro fyzickou část testu Hackbench i7-740Q [autor] 85
Obrázek č. 11 – Leveneův test pro fyzickou část testu Hackbench i7-740Q [autor] 85
Obrázek č. 12 – Procedura GLM pro fyzickou část testu Hackbench i7-4790K [autor] 85
Obrázek č. 13 – Procedura Boxlpot pro fyzickou část testu Hackbench i7-4790K [autor] 86
Obrázek č. 14 – Leveneův test pro fyzickou část testu Hackbench i7-4790K [autor] 86
Obrázek č. 15 – Neparametrický Kruskal-Wallisuv test pro fyzickou část Hackbench i7-4790K[autor] 86
Obrázek č. 16 – Párové porovnání procedury npar1way pro fyzickou část Hackbench i7-4790K [autor] 87
Obrázek č. 17 – Procedura npar1way pro fyzickou část testu Hackbench i7-4790K [autor] 87
Obrázek č. 18 – Procedura GLM pro fyzickou část testu Hackbench i5-7267U [autor] 88
Obrázek č. 19 – Procedura Boxlpot pro fyzickou část testu Hackbench i5-7267U [autor] 88
Obrázek č. 20 – Leveneův test pro fyzickou část testu Hackbench i5-7267U [autor] 88
Obrázek č. 21 – Procedura Boxlpot pro virtualizovanou část testu Hackbench i7-740Q [autor] 90
Obrázek č. 22 – Procedura GLM pro virtualizovanou část testu Hackbench i7-740Q [autor] 90
Obrázek č. 23 – Leveneův test pro virtualizovanou část testu Hackbench i7-740Q [autor] 90

Obrázek č. 24 – Procedura Boxlpot pro virtualizovanou část testu Hackbench i7-4790K [autor]....91
Obrázek č. 25 – Procedura GLM pro virtualizovanou část testu Hackbench i7-4790K [autor]....91
Obrázek č. 26 – Leveneův test pro virtualizovanou část testu Hackbench i7-4790K [autor] 91
Obrázek č. 27 – Procedura Boxlpot pro virtualizovanou část testu Hackbench i5-7267U [autor]....92
Obrázek č. 28 – Procedura GLM pro virtualizovanou část testu Hackbench i5-7267U [autor]....92
Obrázek č. 29 – Leveneův test pro virtualizovanou část testu Hackbench i5-7267U [autor] 92
Obrázek č. 30 – Procedura Boxlpot pro fyzickou část testu Stress CPU i7-740Q [autor] ..94
Obrázek č. 31 – Procedura GLM pro fyzickou část testu Stress CPU i7-740Q [autor] 94
Obrázek č. 32 – Leveneův test pro fyzickou část testu Stress CPU i7-740Q [autor] 94
Obrázek č. 33 – Procedura Boxlpot pro fyzickou část testu Stress CPU i7-4790K [autor] 95
Obrázek č. 34 – Procedura GLM pro fyzickou část testu Stress CPU i7-4790K [autor] 95
Obrázek č. 35 – Leveneův test pro fyzickou část testu Stress CPU i7-4790K [autor] 95
Obrázek č. 36 – Procedura Boxlpot pro fyzickou část testu Stress CPU i5-7267U [autor] 96
Obrázek č. 37 – Procedura GLM pro fyzickou část testu Stress CPU i5-7267U [autor] 96
Obrázek č. 38 – Leveneův test pro fyzickou část testu Stress CPU i5-7267U [autor] 96
Obrázek č. 39 – Procedura Boxlpot pro virtualizovanou část testu Stress CPU i7-740Q [autor]....97
Obrázek č. 40 – Procedura GLM pro virtualizovanou část testu Stress CPU i7-740Q [autor] 97
Obrázek č. 41 – Leveneův test pro virtualizovanou část testu Stress CPU i7-740Q [autor]98
Obrázek č. 42 – Procedura Boxlpot pro virtualizovanou část testu Stress CPU i7-4790K [autor]....98
Obrázek č. 43 – Procedura GLM pro virtualizovanou část testu Stress CPU i7-4790K [autor]....98
Obrázek č. 44 – Leveneův test pro virtualizovanou část testu Stress CPU i7-4790K [autor] 99
Obrázek č. 45 – Procedura Boxlpot pro virtualizovanou část testu Stress CPU i5-7267U [autor]....99
Obrázek č. 46 – Procedura GLM pro virtualizovanou část testu Stress CPU i5-7267U [autor]....99
Obrázek č. 47 – Leveneův test pro virtualizovanou část testu Stress CPU i5-7267U [autor] 100
Obrázek č. 48 – Procedura GLM pro fyzickou část testu Context Switching i7-740Q [autor]....102
Obrázek č. 49 – Procedura Boxlpot pro fyzickou část testu Context Switching i7-740Q [autor]....103
Obrázek č. 50 – Leveneův test pro fyzickou část testu Context Switching i7-740Q [autor] 103
Obrázek č. 51 – Procedura GLM pro fyzickou část testu Context Switching i7-4790K [autor]....103
Obrázek č. 52 – Procedura Boxlpot pro fyzickou část testu Context Switching i7-4790K [autor]....104

Obrázek č. 53 – Leveneův test pro fyzickou část testu Context Switching i7-4790K [autor] 104
Obrázek č. 54 – Procedura GLM pro fyzickou část testu Context Switching i5-7267U [autor] 104
Obrázek č. 55 – Procedura Boxlpot pro fyzickou část testu Context Switching i5-7267U [autor] 105
Obrázek č. 56 – Leveneův test pro fyzickou část testu Context Switching i5-7267U [autor] 105
Obrázek č. 57 – Procedura Boxlpot pro virtualizovanou část testu Context Switching i7-740Q [autor] 106
Obrázek č. 58 – Procedura GLM pro virtualizovanou část testu Context Switching i7-740Q [autor] 107
Obrázek č. 59 – Leveneův test pro virtualizovanou část testu Context Switching i7-740Q [autor] 107
Obrázek č. 60 – Procedura Boxlpot pro virtualizovanou část testu Context Switching i7-4790K [autor] 107
Obrázek č. 61 – Procedura GLM pro virtualizovanou část testu Context Switching i7-4790K [autor] 108
Obrázek č. 62 – Leveneův test pro virtualizovanou část testu Context Switching i7-4790K [autor] 108
Obrázek č. 63 – Procedura Boxlpot pro virtualizovanou část testu Context Switching i5-7267U [autor] 108
Obrázek č. 64 – Procedura GLM pro virtualizovanou část testu Context Switching i5-7267U [autor] 109
Obrázek č. 65 – Leveneův test pro virtualizovanou část testu Context Switching i5-7267U [autor] 109

Seznam tabulek

Tabulka č. 1 – Natavení parametrů jádra pro fyzickou část testování [autor] 47
Tabulka č. 2 – Natavení parametrů jádra pro virtualizovanou část testování [autor] 47
Tabulka č. 3 – Naměřené hodnoty testu Compile Bench [autor] 49
Tabulka č. 4 – Naměřené hodnoty testu Flexible IO Tester [autor] 51
Tabulka č. 5 – Naměřené hodnoty testu PostMark [autor] 51
Tabulka č. 6 – Naměřené hodnoty testu Timed Linux Kernel Compilation [autor] 52
Tabulka č. 7 – Naměřené hodnoty testu Glibc bench [autor] 53
Tabulka č. 8 – Naměřené hodnoty testu Hackbench [autor] 54
Tabulka č. 9 – Naměřené hodnoty testu OpenSSL [autor] 55
Tabulka č. 10 – Naměřené hodnoty testu TTSIOD 3D Renderer [autor] 56
Tabulka č. 11 – Naměřené hodnoty testu Apache Benchmark [autor] 57
Tabulka č. 12 – Naměřené hodnoty testu NGINX Benchmark [autor] 57
Tabulka č. 13 – Naměřené hodnoty testu PostgreSQL pgbench [autor] 59
Tabulka č. 14 – Naměřené hodnoty testu Redis [autor] 60
Tabulka č. 15 – Naměřené hodnoty testu Stress-NG [autor] 62
Tabulka č. 16 – Souhrnný pokusný plán pro Flexible IO Tester fyzické části [autor] 65
Tabulka č. 17 – Souhrnný pokusný plán pro Flexible IO Tester virtualizované části [autor] 66
Tabulka č. 18 – Výstupy procedury GLM a Leveneova testu pro Flexible OI Tester [autor] 67

Tabulka č. 19 – Výstup mnohonásobného porovnávání pro Flexible IO Tester fyzické části [autor] 68
Tabulka č. 20 – Výstup mnohonásobného porovnávání pro Flexible IO Tester virtualizované části [autor] 69
Tabulka č. 21 – Zkrácený výstup procedury GLM pro testování analýzy rozptylu u testu Hackbench [autor] 69
Tabulka č. 22 – Hackbench mnohonásobné porovnání fyzické části [autor] 70
Tabulka č. 23 -Hackbench mnohonásobné porovnání virtualizované části [autor] 70
Tabulka č. 24 – Stress CPU zkrácený výstup procedury GLM [autor] 71
Tabulka č. 25 – Stress CPU mnohonásobné porovnávání fyzické části [autor] 71
Tabulka č. 26 – Stress CPU mnohonásobné porovnávání virtualizované části [autor] 72
Tabulka č. 27 – Context Switching zkrácený výstup procedury GLM [autor] 72
Tabulka č. 28 – Context Switching mnohonásobné porovnávání fyzické části [autor] 72
Tabulka č. 29 – Context Switching mnohonásobné porovnávání virtualizované části [autor] 73
Tabulka č. 30 – Četnost výskytu procentní ztráty výkonu u různých procesorů všech testů [autor] 74
Tabulka č. 31 –Nejčastější výskyt intervalu ztráty výkonu u různých typů měření [autor] 76
Tabulka č. 32 – Výsledky analýzy rozptylu – statisticky významná změna při poklesu výkonu [autor] 77
Tabulka č. 33 – Pokusný plán pro fyzickou část testu Hackbench [autor] 84
Tabulka č. 34 – Pokusný plán pro virtualizovanou část testu Hackbench [autor] 89
Tabulka č. 35 – Pokusný plán pro fyzickou část testu Stress CPU [autor] 93
Tabulka č. 36 – Pokusný plán pro virtualizovanou část testu Stress CPU [autor] 97
Tabulka č. 37 – Pokusný plán pro fyzickou část testu Context Switching [autor] 102
Tabulka č. 38 – Pokusný plán pro virtualizovanou část testu Context Switching [autor] . 106
Tabulka č. 39 – Přehled změn výkonu procesorů při zapnutých mitigacích i7-740Q [autor] 111
Tabulka č. 40 – Přehled změn výkonu procesorů při zapnutých mitigacích i7-4790K [autor] 112
Tabulka č. 41 – Přehled změn výkonu procesorů při zapnutých mitigacích i5-7267U [autor] 113

1 Úvod

Nebývale rychlým vývojem v oblasti IT se také zvětšují prostory pro potenciální napadení ať už sítí, virtuálních prostředí či fyzických zařízení. Hrozby jsou inteligentnější a automatizované, což značně ztěžuje jejich odhalování. Sledování chyb a jejich zneužití je monitorováno na všech vrstvách od aplikačních serverů, programových knihoven nebo operačních systémů až po zpracování jednotlivých bitů v procesorech. V posledních několika letech se začaly objevovat bezpečnostní hrozby počítačů, které se konkrétně týkají procesorů. Nejznámějšími jsou Meltdown nebo Spectre útoky, které útočníkům umožňují odcizit citlivá data z téměř jakéhokoliv počítače, mobilního zařízení anebo i z cloudového úložiště. Takové informace nejsou pro dnešní svět internetových technologií vůbec příznivé. Dobrou zprávou je, že se těmito hrozbami odborníci prioritně zabývají a vytváří na ně opravy, aby ochránili co nejvíce ohrožených systémů a výrobků a zároveň se snaží aktualizovat ostatní produkty, aby byly vůči těmto chybám odolné již od výroby. Nevýhodou těchto oprav či záplat, je že tyto nucené aktualizace snižují výkon celého počítače.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem diplomové práce je analýza výkonu procesorů při opravách zranitelností typu útok postranním kanálem Spectre V1, V2, Meltdown a Foreshadow. Výkon bude porovnán pomocí testů na fyzických a virtualizovaných systémech. Dílčím cílem je provést jednoduchou analýzu rozptylu a mnohonásobného porovnávání režimu bez aplikovaných a s aplikovanými mitigacemi v různých konfiguracích pro ověření statistické významnosti ztráty výkonu.

2.2 Metodika

Teoretická část bude zpracována pomocí rešerše odborné literatury a publikovaných článků. Vysvětleny budou základní pojmy stavby a výroby procesorů, techniky zvyšování výkonu procesorů se zaměřením na ty, které se jeví z pohledu bezpečnosti jako problematické. Jedná se především o pipelining, spekulativní vykonávání, predikce procesoru, TLB cache nebo simultánní multi-threading a hyper-threading. Dále budou vysvětleny principy zranitelností procesorů pomocí útoků Spectre V1, V2, Meltdown, Foreshadow a jejich vliv ve virtualizovaném prostředí. Následně bude představeno jejich odstranění a ochrana před těmito hrozbami.

V praktické části bude vybrán operační systém a virtualizované prostředí typu hypervizor 1, testovací komponenty a jednotlivé testy. Následně budou vytvořeny scénáře pro testování ztráty výkonu. Definovány jednotlivé testy, které omezují vstupní a výstupní výkon procesoru ve fyzickém prostředí, vliv chyb na hyper-threading ve virtualizovaném prostředí a jednotlivé testy rychlosti procesoru. Následně se povypínají jednotlivé služby počítače a otestuje se jejich vliv na výkon systému. Nakonec se provedou jednotlivá měření a vyhodnotí intervalové četnosti ztráty výkonu u různých typů měření a procentuální úbytky výkonu mezi všemi naměřenými režimy testování ve fyzickém i virtualizovaném prostředí. Statistickým testováním pomocí jednoduché analýzy rozptylu a mnohonásobného porovnávání bude ověřeno, zda opravdu došlo ke statisticky významnému rozdílu úbytku výkonu ve specifikovaných testech u jednotlivých režimů.

3 Teoretická východiska

3.1 Procesory

Každý počítač, server i mobil se skládá z jednotlivých komponent, bez kterých by jako celek nikdy nemohl fungovat. Srdcem, ale i mozkem celého počítače je centrální procesorová jednotka (zkráceně procesor nebo CPU), která je elektronickou součástkou vykonávající strojové instrukce, které tvoří počítačový program. Na začátku svého vývoje byl tvořen z několika elektronických součástek a postupem času byly všechny jeho nezbytné obvody sloučeny do jednoho celku, integrovaného obvodu, který se označuje mikroprocesor. (1)

3.1.1 Výroba čipu procesoru

Základní stavební součástkou čipu, který je v podstatě všude, jsou miliardy tranzistorů, které se s každou další vydanou architekturou zmenšují. Dnešní technologie už umožňuje vyrábět takové součástky na úrovni 14 nanometrů. Čip je tvořen z křemíku s čistotou 99,9999 %, protože i nepatrné znečištění dokáže velmi ovlivnit kvalitu vyrobených tranzistorů. Vysoce čistý křemík se používá v podobě monokrystalu, kterou je potřeba nařezat na tenké vrstvy tzv. wafery. Každý wafer má běžně velikost desetiny milimetru. Poté je do waferů vyleptána samotná struktura čipů. Používá se proces fotolitografie, kde se nanese na wafr světlocitlivá vrstva, která je následně ozářena ultrafialovým zářením, díky kterému je umožněno kreslit integrovaný obvod na křemík. Tam, kde byla vrstva ozářena byla také narušena struktura světlocitlivé vrstvy, kterou lze speciálním chemickým roztokem následně odstranit. Poté co je vymyta tato vrstva se přechází na fázi leptání samotného materiálu, kterým dochází k prohloubení samotného vzorku na fotocitlivé vrstvě a tím vzniká finální tvar tranzistoru. (1)

V další fázi se znovu využívá nanesení fotocitlivé vrstvy, která brání ta místa, kam nemá být nanesena iontová vrstva. Ta je nanášena na křemík z důvodu úpravy schopnosti tranzistoru vést elektrický proud. Následně je na takový tranzistor nanesena izolační vrstva a jeho tři otvory (výstupy) jsou poté vyplněný mědí, která slouží k propojení s okolními tranzistory. Leštěním měděného povrchu tranzistoru se docílí odstranění přebytečné mědi okolo výstupů tranzistoru, čímž vzniknou jeho konektory. Jednotlivé tranzistory jsou mezi sebou propojeny za pomoci měděných můstků, které reprezentují jednotlivé spoje. Jejich

propojení závisí na konkrétní architektuře čipu. Díky použité nanotechnologii se v tak tenkém procesoru nachází až dvacet vrstev mezi sebou propojených obvodů. Po nanesení všech nezbytných vrstev se přechází k ověření správných reakcí pomocí testovacích sekvencí. Posledním krokem je nařezání waferu na dílčí čipy pomocí vodního paprsku z důvodu zabránění poškození povrchu např. laserem. Z waferu jsou postupně odebírány jednotlivé funkční procesory, zbytky jsou určeny k recyklaci. Na samotném závěru výroby procesoru je zapouzdření čipu do kovového těla, které zajišťuje ideální rozvedení tepla z procesoru a současně tvoří kryt křemíkového jádra. Zespoda těla se nacházejí piny, které slouží k propojení procesoru se základní deskou počítače pomocí procesorové patice. (1)

Dnešní čipy nejsou dokonale křemíkově čisté, nečistoty se obvykle nejvíce nacházejí na okrajích jednotlivých waferů. Proto je tedy logické, že místem, kde je křemík nejčistší je samotný střed waferu, a proto z těchto částí se vyrábějí ty nejvýkonnější a nejdražší modely čipů, které musí být schopné zvládat vysoké frekvence při udržení nezbytné spotřeby. (1)

3.1.2 Stavba procesoru

Jakýkoliv počítač lze dekomponovat na jednotlivé elementární funkční prvky. Hlavními stavebními částmi je samotný procesor, paměť vykonávaného programu, paměť pro data a periferní obvody. Periferie jsou velmi rozmanité a svou skladbou závisí na aplikaci počítače. Vyskytují se ve formě vstupních a výstupních obvodů zajišťující spojení počítače s okolím. Vždy jsou přítomny přinejmenším v podobě převodníků čítačů či časovačů. Všechny jednotky jsou propojeny mezi sebou pomocí sběrnic. Díky této koncepci lze přidávat do počítače další jednotky bez nutnosti změny vnitřního zapojení. Sběrnice je vždy řízena pouze jednou z jednotek, kterou je obvykle procesor. Nevýhodou této koncepce je nerealizovatelnost předávání dat ze dvou různých zdrojů ke dvěma různým příjemcům. V jednu chvíli může být na sběrnici připojen jen jeden zdroj dat. (2)

Připojená jednotka na datovou sběrnici přenáší data a může být jejich zdrojem i příjemcem (nebo střídavě obojím). Dále se zde nachází adresová sběrnice, která zastává činnost adresování do paměti. Jednotlivé operace jako je čtení nebo zápis, jsou řízeny pomocí řídící sběrnice. Řídící signály zprostředkovává procesor a pomocí nich řídí celý chod počítače. Má za úkol provádět jednotlivé instrukce uložené v paměti programu,

zpracovávat data v paměti a řídit příchozí data ze vstupních periferií jejich zpracování a následný tok výstupních dat ven z počítače. (2)

V paměti pro programy se nacházejí instrukce, které svým po sobě jdoucím zpracováním vykonávají požadovanou činnost. Dále se zde nacházejí konstanty, nebo používané tabulky programu. Nejčastěji je taková paměť reprezentována ve formě RAM paměti (paměti s přímým přístupem umožňující čtení i zápis) nebo výjimečně ROM pamětí či jejích ekvivalentů EPROM, EEPROM nebo FLASH (paměti, které uchovávají v podstatě neměnný program, jejich činnost je dána a málokdy se mění). V paměti RAM se po vypnutí napájení neuchovají žádná data, proto při zapnutí počítače je obsah RAM náhodný a nelze s ní tedy hned od začátku pracovat. To řeší paměť ROM, kde je uložený BIOS (základní vstupně-výstupní funkce počítače), který se používá pro inicializaci a konfiguraci připojeného hardwaru a následného spuštění operačního systému načtením z velkokapacitního paměťového úložiště a přesunutím do paměti RAM. Teprve pak přebírá řízení počítače samotný operační systém. (2)

V paměti pro data se nacházejí dočasné hodnoty a mezi výpočty, které byly získány ze vstupu. Takové paměti jsou pouze typu RAM a závisí na architektuře, zda je tato paměť oddělena od programové paměti (Harvardská architektura) nebo zda se využívá jednotné paměti pro data i instrukce současně (architektura John von Neumana). Taková struktura se dnes používá v každém počítači. (2)

Díky technologii ASIC, která sdružuje dílčí osvědčené a ověřené masky (jednotlivé jednotky počítače), se sestaví jádro, kterým je procesor a přilehlé obvody. Spolu s paměťmi se jádro stává základem jednoho integrovaného čipu, který tvoří společné jádro pro jednotlivé rodiny mikropočítačů. Vývoj nových procesorů a zvyšování jejich vnitřních pamětí je oproti periferiím konvenční z důvodu zachování komptability programů. Je to způsobeno finanční stránkou v tomto odvětví, především vysokými investicemi do nových programů, které musí stejným způsobem fungovat jak na nejmodernějších procesorech, tak i na těch předchozích. Novými prvky procesorů tedy bývají obvykle nově přidané instrukce, zvýšení rychlosti nebo přidání speciálního obvodu. Tím se zajistí kompatibilita směrem nahoru, staré programy jsou stejně použitelné i s verzí novou, ale ne nutně naopak. (2)

3.2 Techniky zvyšování výkonu procesorů

Výpočetní výkon procesorů závisí na počtu vykonaných strojových instrukcí za jednotku času. To bylo v průběhu let ovlivněno mnoho faktory, především masivní výrobou integrovaných čipů, kde se vzdálenost mezi jednotlivými spoji tranzistorů, ale i velikost fyzických tranzistorů neustále zmenšovala. Tím také docházelo ke snižování napájecího napětí a množství vyzařovaného tepla do prostoru a možnosti zvyšování taktovací frekvence procesoru. Další důležitou technikou při zvyšování výkonu procesoru je samotný výrobní proces (viz kapitola 3.1.1 Výroba čipu procesoru).

Základní techniky zvyšování výkonu procesorů:

- Rozšíření bitové šířky zpracování dat
- Zvýšení počtu pracovních registrů
- Hierarchické uspořádání paměti

Problematické techniky zvyšování výkonu procesorů:

- Fronta instrukcí
- Pipelining
- Speculative execution
- Branch prediction
- Out of order execution
- Indirect branch prediction
- Return target predictor
- TLB Cache
- Simultánní Multi-threading a Hyper-threading

Všechny techniky budou postupně vysvětleny. Podrobný princip fungování bude uveden u technik, které se z hlediska útoku postranním kanálem jeví jako velice problematické. Abychom byly schopni komplexně chápat jak útoky, nejenom Spectre a Meltdown fungují je potřeba pochopit jednotlivé techniky zvyšování výkonu a základní pojmy.

3.2.1 Rozšíření bitové šířky zpracovávaných dat

Nejjednodušší technikou pro dosažení zvýšení výpočetního výkonu je rozšíření počtu bitů všech zpracovatelných instrukcí pomocí procesoru. Můžeme si povšimnout, že takových to rozšíření bylo v minulosti několik. Od čtyřbitových kalkulátorů se přešlo k osmibitovým mikroprocesorům. Takové mikroprocesory fungují v mnoha zařízeních i dnes v podobě mikrořadičů, které mají v sobě zabudovanou i paměť. Průběhem času jsme se dostali postupným násobným přidáváním až na architekturu třiceti dvou bitovou, kterou dnes postupně nahrazujeme šedesáti čtyř bitovou. (4)

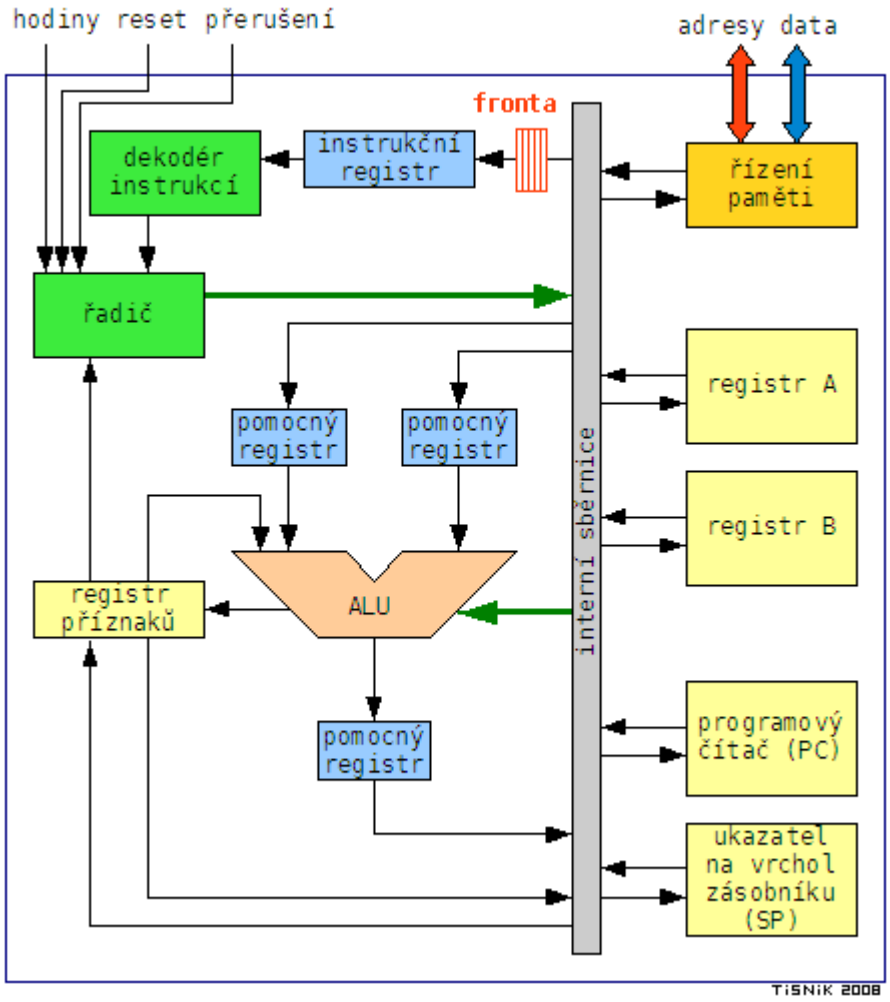
3.2.2 Zvýšení počtu pracovních registrů

S postupným vývojem tranzistorů se jeví možnost pro návrháře zvýšit počet pracovních registrů, kterými se dříve šetřilo v závislosti na výrobní ceně. V tomto momentě jsou již soubory pracovních registrů natolik rozsáhlé, že jejich prostor, který fyzicky zabírají na ploše čipu je minimální oproti jednotkám ALU (Aritmeticko-logická jednotka) či FPU (Matematický koprocesor). (4)

3.2.3 Hierarchické uspořádání paměti

Dnešní výrobní procesy dovolují vyrábět velmi rychlé paměti SRAM, které jsou polovodičového typu RAM realizované bistabilním klopným obvodem. Taková paměť, je ale díky složité konstrukci velmi drahá, a proto se využívá pouze v malém kapacitním provedení přímo v procesoru, označovaná jako hardwarová cache paměť. Těchto pamětí je v procesoru více a jsou rozdělené do jednotlivých úrovní. Na vrcholku hierarchické struktury je paměť L0, která je ze všech pamětí nejmenší, ale také nejrychlejší. Můžeme zde nalézt pracovní registry procesoru. Další pamětí v hierarchii je L1, která zastává funkci vyrovnávací paměti následovaná pamětí L2, která je rovněž vyrovnávací, ale má větší velikost a je pomalejší než ty předchozí. L3 paměť je poslední relativně malou pamětí v procesoru, která přímo komunikuje s RAM pamětí počítače. Ta má výhodu pouze ve velikosti v řádech desítek gigabajtů, ale z hlediska rychlosti procesoru je velmi pomalá. Posledním místem odkud dochází k přesunu dat je z pevného disku počítače do paměti RAM. V reálu je zapotřebí také využití virtuální paměti, kterou již dnes všechny nejmodernější počítače umožňují. Virtuální paměť je způsob správy operační paměti počítače, která je uspořádána jinak, než se ve skutečnosti jeví adresní prostor nebo je i větší

než samotná fyzická paměť RAM. Taková paměť je realizována pomocí stránkovací paměti a stránkování na disk. (4)



Obrázek č. 1 - Blokové schéma mikroprocesoru [4]

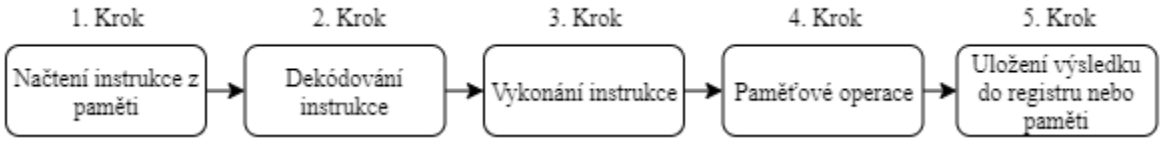
3.2.4 Fronta instrukcí

Při porovnání jednotlivých instrukcí z hlediska doby provádění v počtu taktů je patrné, že se tato doba může odlišovat z důvodu větší rychlosti mikroprocesoru než operační paměti, a i z hlediska jejího využití, které není výhradně uniformní. V takovém případě, by byl procesor nucen počkat na provedení celé instrukce a teprve poté by vykonal další. Tím by v procesoru vznikaly pomyslné výpočetní mezery, ty nejsou využity (v mezičase nevykonávají žádnou úlohu) a z hlediska výkonu dochází k poklesu výkonu (nevyužíváme procesor na 100 %). Jednoduchým řešením je přidání fronty instrukcí do samotného procesoru. Jedná se o paměť typu FIFO (First In First Out, tzv. fronta), do které

jsou postupně ukládány operační kódy instrukcí z operační paměti. Na výstupu paměti si kódy vybírá řadič, viz Obrázek č. 1. Při vykonávání složitějších instrukcí jako je násobení, se mezitím do paměti nahrávají paralelně další operační kódy. Výhodou je i zrychlení při provádění jednoduchých instrukcí, například součet dvou registrů, kde se operační kódy velmi rychle načítají z FIFO paměti a není tak nutno čekat na nahrání přímo z paměti operační. Pokud dojde k přerušení a následnému skoku do podprogramu či k jeho návratu (změně běhu programu) je nutno FIFO paměť zcela vyprázdnit. (4)

3.2.5 Pipelining

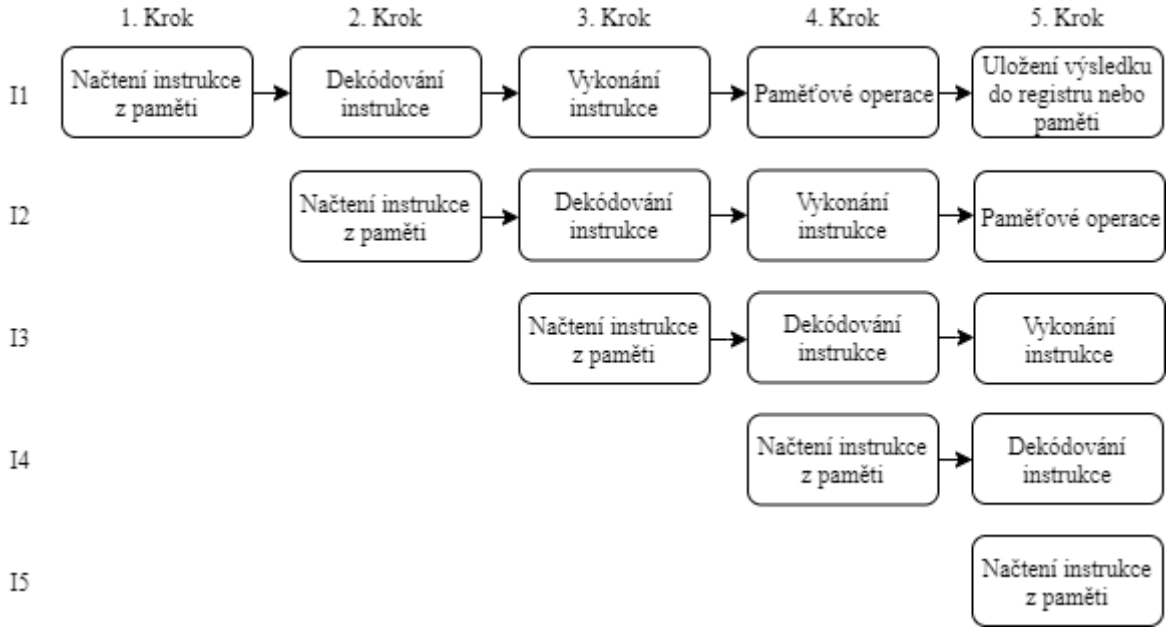
Pipelining je zřetězené zpracování instrukcí v procesoru. Je to další technologie, která byla zavedena do procesoru pro jeho zvýšení výkonu. Při procesu zpracování jedné instrukce je potřeba projít minimálně pěti základními kroky viz Obrázek č. 2. Pokud budeme zpracovávat instrukci součet registrů `A` a `B`, je nejprve potřeba načíst operační kód instrukce z operační paměti. V tomto případě jde o zašifrovanou posloupnost `ADD A, B`. Tento kód se načte do řadiče a ten na základě rozpoznané instrukce spustí proces přenesení obsahu registru `A` do prvního pomocného registru ALU (Aritmeticko Logická Jednotka), který se nachází před touto jednotkou. Stejná operace se vykoná i s registrem `B` – přenese se hodnota registru `B` do druhého pomocného registru před ALU. Následně řadič dá příkaz ALU vykonat instrukci `ADD` (součet) a poté uložit výsledek do třetího pracovního registru, který se nachází na výstupu ALU viz Obrázek č. 1. (4)



Obrázek č. 2 - Jednotlivé kroky při zpracování instrukce [5]

Protože pro vykonání instrukce je potřeba minimálně těchto pět kroků a procesor by neměl funkci pipelining, musel by nejprve vykonat všech těchto pět kroků a teprve poté by mohl začít pracovat na instrukci druhé. Takovýmto způsobem, je ale patrné, že by procesor většinu svého času stál a nepracoval. Proto, aby bylo dosaženo co největší efektivity z hlediska práce procesoru (aby pokaždé něco vykonával) se do procesoru přidal pipelining. Fungování je následující viz Obrázek č. 3. Prvním krokem je načtení první instrukce (I1), následně I1 ve druhém kroku dekóduji, ale zároveň mohu načíst instrukci

druhou (I2), protože když I1 je už ve kroku druhém, nemusím čekat na provedení celé instrukce, ale postupně po jednotlivých krocích mohu zpracovávat paralelně minimálně instrukcí pět (každou v jiném stavu). Výsledkem je plné využití procesoru, které je možno vidět v pátém kroku Obrázeku č. 3. Zde je možné vidět najednou v jednom taktu zapisování výsledku první instrukce I1, zároveň vykonávání paměťových instrukcí s instrukcí druhou I2, také se pracuje na instrukci třetí I3, která se právě vykonává, čtvrtá instrukce I4 je ve stavu dekódování a poslední pátá instrukce I5 se v tentýž době načítá z paměti. Tím, že procesory využívají pipelining dokážou zpracovat jednu instrukci za jeden takt namísto pěti, což je pětkrát rychleji než bez pipeliningu. (5)



Obrázek č. 3 - Tok instrukcí v procesoru pomocí pipeliningu [5]

Pipelining je dnes součástí všech typů procesorů. Na úrovni programátora se s ním lze setkat v programovacím jazyku Assembler, což je nízko úrovňový jazyk, který zpracovává symbolické instrukce, které reprezentují jednotlivé instrukce strojové, ty pak zpracovává procesor. Výhodou také je, že výrobci nemusí investovat vysoké částky do zvyšování počtu tranzistorů v procesoru. U grafických čipů je tento stav více zřetelný, protože zde může mít pipeline až několik stovek kroků. Pro názorné ukázky se využívá zjednodušení na pět kroků, jak již bylo popsáno, ale v realitě jsou počty kroků větší a jednotlivé kroky z hlediska složitosti jednodušší, z toho vyplívá, že mohou trvat kratší dobu (tím se zvyšuje taktovací frekvence). (4)

3.2.6 Speculative execution

Pro správné pochopení odkud se tyto hrozby berou je nutné porozumět procesu, který je prováděn v procesoru každého počítače. Nazýváme ho spekulativním prováděním (ang. Speculative execution). Tento proces je optimalizační metoda, která provádí pomocné práce, které nejsou nezbytně nutné. Umožňuje zařízením provádět určité operace dopředu, pro urychlení vykonání rutinních činností. Taková úloha se začne provádět ještě dříve, než je znám výsledek předchozí operace. Provádí se pro zkrácení doby, která by byla nutná pro provedení navazující operace na předchozí výsledek. Výsledky těchto pomocných prací se pak v budoucnu použijí a sníží se tím výrazně pracovní doba, nebo jsou zahozeny a nevyužity. Jejich využití primárně závisí na dané práci a volných zdrojích, ale také na provádění jiné nedokončené práce, která je na ní závislá. (3, 7)

Využitím na nízké úrovni je použití při předvídání skoků v procesorech s využitím překryvného zpracování strojových instrukcí (ang. Pipelining). Algoritmus předpovídá provádění větví programu na základě historie skutečně provedeného větvení. Při zpracování větvící instrukce nastane pro procesor problém, protože neví, jakou další instrukci má načíst z důvodu dvou možností běhu následujících instrukcí. Kontrolní jednotka tedy zastaví načítání dalších instrukcí z pipeline do té doby, než je zcela jasné, zda podmíněná instrukce skončí stavem logické 1 neboli true nebo logickou 0 čili false. Na jiné úrovni může být tento algoritmus použit například při přednačtení dat či celých souborů do mezipaměti procesoru na základě odhadu blízkého užití těchto dat v budoucnu. (3, 7)

Všechny výsledky, které nebyli predikcí využity končí v nezabezpečené části procesoru, v jeho cash paměti, ke které útočník může přistupovat pomocí postranního kanálu. Nezabezpečení těchto dat vychází z historie šedesátých let minulého století, kdy bylo spekulativní provádění vynalezeno, známé pod pojmem Tomasulův algoritmus. Jednotlivé počítače byly samostatné a soběstačné. Nikdo neuvažoval o možnosti tohoto rizika, protože neexistoval žádný způsob, jak tato zahozená data přečíst. V dnešní době, kdy počítače a mobilní zařízení sdílí systémové zdroje s mnoha aplikacemi a prostředími je to problém, protože se zahozená a nezabezpečená data nacházejí ve sdílené paměti. Útočník využívá postranní kanál k nepozorovanému vniknutí do této paměti a odchytává tato data. V horším případě může přimět počítač načítat citlivá data do této sdílené paměti jako jsou informace o účtech nebo hesla a zneužít je ve svůj prospěch. Po odhalení této

chyby byly kontaktovány přední firmy ve výrobě procesorů a tisíce inženýrů se spojilo, aby vytvořili záplaty na tyto hrozby, jako je Meltdown či Spectre využívající této chyby. Proto je nezbytné instalovat záplaty okamžitě a udržovat vlastní operační systém v nejnovější verzi. V budoucnu se návrh obvodu tohoto kritického místa změní, aby se hrozbám předešlo. Tím že se změní návrh, nebude nutné již vydané záplaty používat a odstraní se ztráta výkonu, kterou záplaty spotřebovávaly. (3, 7)

3.2.7 Branch prediction

Pro přiblížení skutečného chování prediktorů skoků v součinnosti se spekulativním prováděním instrukcí v procesoru lze využít jednoduchého příkladu kódu (7) (uvedený níže), který je napsaný v jazyce Assembler. Uvažujme o procesoru, který je schopen predikovat (jednobitový prediktor skoků) a využívá instrukční pipeline, kde dokáže dopředu připravit čtyři instrukce. Nyní postupně provedeme celý ukázkový kód a budeme sledovat hodnoty `ZACATEK=0` a `KONEC=10`. (5, 7)

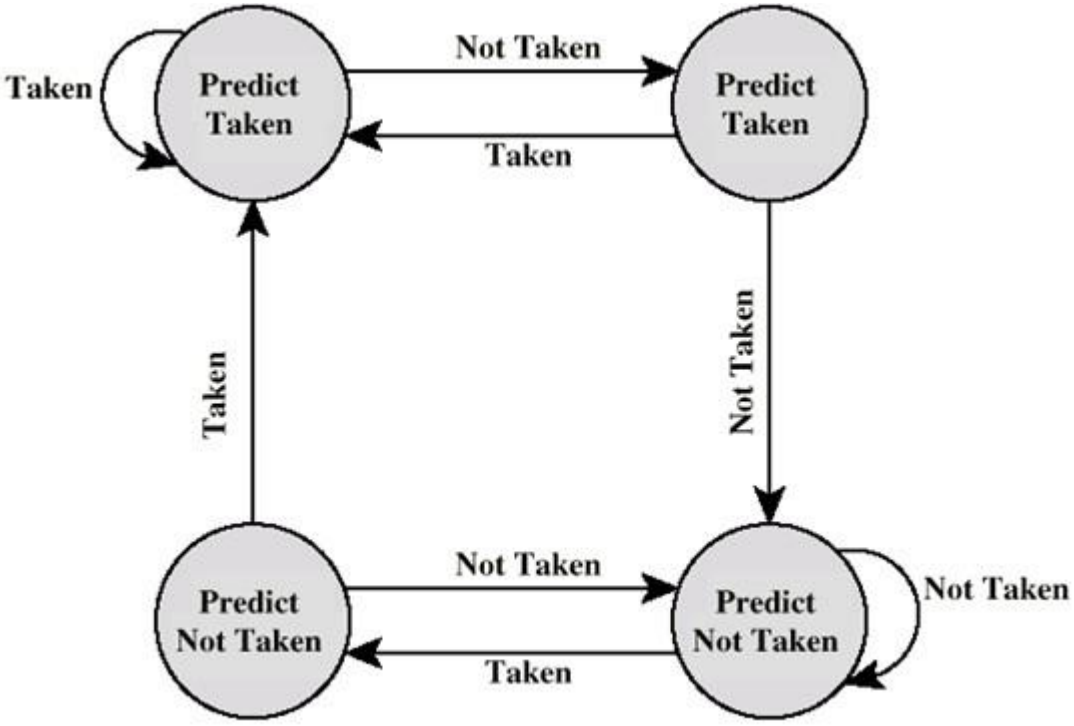
Následující kód je napsán v jazyce Asembler, který demonstruje chování prediktoru skoků. Tišnovský uvádí tento kód (7):

```
; jednoduchá počítaná smyčka typu „for“
     LD A, ZACATEK   ;počáteční hodnota smyčky
     LD B, KONEC     ;koncová hodnota smyčky

SMYCKA
     příkaz x1       ;libovolné instrukce, jejichž celková
     příkaz x2       ;délka musí být menší než cca 120 bytů
     příkaz x3       ;(kvůli omezení relativního skoku)
     INC A           ;zvýšení počitadla smyčky o jedničku
     CMP A,B
     JNZ SMYCKA      ;příznak „Zerro flag“ se nastaví při rovnosti
A a B

     příkaz y1       ;libovolné instrukce provedené až po ukončení
smyčky
     příkaz y2
     příkaz y3
```

Při prvním spuštění programu jsou obvykle nastaveny prediktory skoků na preferenci skoku směrem vzad, která vychází z předpokladu smyčky programu. Předpokládá se, že pokud je v programu cyklus, opakované provádění části kódu bude vykonáno nejméně dvakrát, jinak by byla smyčka z logického hlediska v kódu zbytečná. Při prvním vstupu do cyklu (`SMYCKA`) načteme do pipeline instrukce `x1, x2, x3` a hned je načítáme ještě jednou znovu aniž bychom věděli, jak bude vyhodnocena podmínka, za které se `SMYCKA` opakuje. Jakmile je dokončena instrukce porovnání `CMP A,B` rozhodující o skoku zpět, procesor rovnou použije predikci, protože byla úspěšná a danou posloupnost instrukcí opravdu provede. Takovýmto způsobem bude kód proveden ještě desetkrát. Poté se vyskytne nová okolnost, a to že hodnota registru `A` je rovna deseti. Porovnávač `CMP A,B` zabrání dalšímu skoku cyklu zpět, tím že nastaví příznak Zerro flag registru na hodnotu jedna. Prediktor ovšem s touto variantou nepočítal, protože vychází z historie predikcí, v tomto případě má svůj předchozí stav uložen v jednom bitu. Protože v posledním kroku vždy skočil zpět tak i nyní má připravené instrukce pro vykonání dalšího cyklu `SMYCKA`, tedy načtené `x1, x2, x3`. Toto je ovšem chybná predikce a nezbude tedy nic jiného než pipeline vyprázdnit a začít do ní načítat následující instrukce `y1, y2 a y3`. (7, 8)

Dá se tedy odvodit, že pokud by smyčka měla proběhnout například stokrát, což procesor dopředu neví. Prvním průchodem by predikoval pokračování v provádění kódu nikoliv smyčku. Byl by penalizován, vyprázdnil by pipeline a zapamatoval si předchozí výsledek skoku, podle kterého by dalších 98 průchodů odhadl správně až na poslední, kdy se pokračuje v provádění následného kódu. Znamená to, že úspěšnost predikce je v ideálním případě z 98% správná a v 98 případech velmi rychlé provedení všech instrukcí díky přednačtení do pipeline. (7)

Jiným případem může být použití dvoubitového prediktoru podobající se více dnešním prediktorům v procesorech. Funguje obdobným způsobem. V jednom bitu uchovává výsledek skoku předcházejícího a v druhém bitu informaci o minulém stavu predikce. Výhodou dvoubitového prediktoru je, že jedena špatná predikce nezmění stav prediktoru. (8)



Obrázek č. 4 - Stavový diagram dvou bitového prediktoru [6]

3.2.8 Out of order execution

Dalším zvýšením výkonu procesorů je posílení systému vykonávání instrukcí mimo pořadí. Ten si dává za cíl snížit počet prováděných cyklů procesoru, kdy nevykonává žádnou akci. In order procesory se potýkají se situací, kdy je následující instrukce závislá na stavu instrukce předcházející a musí vyčkat na její dokončení z důvodu možné změny hodnoty pracovního registru nebo příznaku, který následující instrukce využívá. To vyvolává zdržování procesoru, procesor stojí. Tím dochází k nevyužívání všech cyklů procesoru dochází ke snižování se pracovního výkonu. Pokud je procesor architektury out of order nebude stát, ale začne vykonávat jiné instrukce, kde je bezpečně jisté, že využívá jiné pracovní registry či příznaky. Jednotlivé následující instrukce jsou postupně ukládány do paměti takzvané rezervační stanice, ze které se načítají, jakmile jsou známi hodnoty jejích operandů. Touto technikou určuje procesor sám pořadí vykonávání jednotlivých instrukcí bez ohledu na binární kód. Instrukce uspořádá tak, aby optimalizoval jejich vykonávání. (7)

Provádění instrukcí mimo pořadí

Pro zjednodušení interpretace chování out of order execution mějme procesor se dvěma pracovními registry. V případě použití procesoru s pouze dvěma registry, je využití takové technologie výhradně mrhání prostorem a počtem tranzistorů na čipu. Vykonáme následující kód (kód (7) uveden níže v levé části), který je vykonáván velmi pomalu v závislosti na výsledku všech čtyř předchozích instrukcí. V takovém případě není využito instrukční pipeline. (7)

Při využití technologie out of order, optimalizační algoritmus rozezná použití příznaků Zerro flag či Carry flag. V tomto případě je kód nevyužívá a může tedy dojít k dynamickému setřídění a spojení instrukcí, kde není nutné vyčkávat na výsledek (kód (7) uveden níže v pravé části). Takovou optimalizaci by měl dnes zajistit při překladu kódu již samotný kompilátor, který zahrnuje i další optimalizační metody. Kód předpřipraví takovým způsobem, aby mohlo být vykonáno co nejvíce instrukcí zároveň a nemusel být touto prací zatížen samotný procesor. V levé části kódu (7) můžeme vidět separované instrukce nejdříve pro registry `A`, následně pro registry `B`. Při provádění tohoto kódu by nejprve byly provedeny všechny instrukce týkající se registru `A`, až poté by se prováděly všechny s registrem `B`. Pokud je kód setříděný tedy optimalizovaný, můžeme instrukce, které nejsou na sobě závislé provádět hned jak je to možné. Můžeme proto načíst registr `A` následně registr `B`. Poté lze vykonávat další požadované instrukce s registry, které je možno vidět v pravé části kódu. (7)

Tišnovský (7) uvádí pro bližší přiblížení tento kód. V levé části nalezneme nesetříděný kód instrukcí a v části pravé setříděný kód instrukcí s využitím technologie out of order:

```
LD   A, [adresa_x]
RL   A
XOR  A, 0x7f
ST   A, [adresa_y]

LD   B, [adresa_z]
INC  B
COM  B
ST   B, [adresa_w]
```

```
LD   A, [adresa_x]
LD   B, [adresa_z]

RL   A
INC  B

XOR  A, 07xf
COM  B

ST   A, [adresa_y]
ST   B, [adresa_w]
```

3.2.9 Indirect branch prediction

Předvídání nepřímých skoků nebo nepřímých volání (volání adresy podle registru), je další nezbytnou součástí pro porozumění a pochopení útoků skrze postranní kanál. Jedná se o předvídání nepřímého větvení. To je takové větvení, kdy umístění následného kódu není předně dáno, ale adresu další instrukce je nutno zjistit pomocí výpočtu. V Assembleru se s nepřímým voláním nejčastěji setkáme ve formě instrukcí `JMP` (jump) a `CALL`. (5)

Při volání samotné instrukce `JMP` nebo `CALL`, kdy získám adresu a následně na ní chci skočit nastává problém pro načtení další instrukce z paměti do pipeline (první krok viz Obrázek č. 3), protože ještě neproběhlo načtení dat z této adresy. Proto tento prediktor nepřímých skoků, na základě aktuální načtené adresy skoku, odhaduje, na jakou adresu v paměti se skočí. Tento proces je poměrně složitý, protože není předpovídána pouze možnost ze dvou (nula nebo jedna), ale výstupem musí být skutečná adresa. (5)

Nepodmíněný skok – JMP

Nepodmíněný skok `JMP` je instrukce, která vykoná skok na návěští, které musí být předem definováno pomocí parametru instrukce. Pravidlem bývá označovat tyto návěští v kódu pomocí dvou znaků zavináče před samotným pojmenováním `@@LoopStart` (jako je to mu v ukázce kódu (9) níže). Překladač kompiluje jednotlivé instrukce v kódu, po objevení skokové instrukce je nucen nalézt a uchovat adresu, na kterou bude skákat, respektive adresu následující hned za návěštím `@@LoopStart` a také adresu následující instrukce po skoku. V tomto případě adresu instrukce `ADD`. Po zjištění těchto dvou hodnot provede rozdíl těchto adres a tím dostane relativní adresu skoku. Tato adresa neznačí přesné místo skoku, ale pouze udává počet bitů, o které budeme skákat. (5, 9)

Assembler, ukázka kódu (9) volání nepodmíněného skoku:

```
mov             ecx, 0

@@LoopStart:                    ; Návěští – začátek cyklu
     add        eax, eax        ; Instrukce uvnitř cyklu
     jmp        @@LoopStart     ; Skok na začátku cyklu
```

3.2.10 Return target predictor

Předvídání návratů pomocí instrukce `RETURN`. Procedura na konci obsahuje instrukci `RET` (zkráceně return). Pro její vykonání je nutno ze zásobníku načíst adresu kam se máme vrátit, tam poté skočí vykonávání. Paměť je pomalá a pokud adresy nejsou v paměti cash, může tato operace trvat stovky cyklů, a proto by hrozilo, že procesor bude opět v nečinnosti, což je nechtěný stav. Chtěným stavem je zaplněná pipeline bez mezer, proto je v CPU další prediktor, který funguje jako stínový zásobník, na kterém jsou uloženy návratové adresy volaných procedur. Následně mohou být prováděny různé operace se zásobníkem a nezávisle na tom co bylo vykonáno je spekulativně vykonán pokus o návrat na adresu, která je uložena v interním stínovém zásobníku. Branch Target Buffer je struktura, která tento šestnácti položkový zásobníček obsahuje. (5)

Volání procedury – CALL

Voláním instrukce `CALL` ve skutečnosti voláme proceduru, která slouží k opakovanému použití stejné funkčnosti. Jde o kus stejného kódu, který by se v hlavním prováděném kódu vyskytoval vícekrát. Proto ho separujeme do podprogramu (procedury), kterou budeme na místech, kde bude třeba volat pomocí instrukce `CALL`. Ta následně vloží adresu bezprostředně následující instrukce do zásobníku, a skočí na adresu, která je dána instrukcí `CALL`. Adresu ukládáme, aby bylo umožněno skočit zpět (`RETURN`) do hlavního programu a provádět další instrukce hned po proceduře. Z vrcholu zásobníku je vzata adresa zpětného skoku a hned poté je skok proveden. Tím je také zajištěn návrat na správnou adresu při opakovaném volání procedury tzv. rekurzivní volání. Pokud by útočník dokázal uskutečnit přepis hodnoty v tomto zásobníku či změnil adresu přímo v registru ESP (Extended Stack Pointer), mohl by při zpětném volání například skočit na adresu první instrukce vlastního kódu, který by byl následně vykonáván. Níže je uveden kód (10), je zde možné vidět instrukci `PUSH`, kterou předáváme parametry funkce do zásobníku a následně volání procedury `MessageBox` pomocí instrukce `CALL`. (5, 10)

Ukázka kódu (10) v Assembleru – volání procedury:

```
push      MB_OK
push      offset szTitle
push      offset szMessage
push      0
call      MessageBox
```

3.2.11 TLB Cache a tabulka stránek

Způsob realizace virtuální paměti pomocí procesoru se provádí prostřednictvím tabulky stránek. Virtuální paměť je režimem adresování umožňující každému procesu podat paměť jiným způsobem, než tomu doopravdy je. Přidělený adresní prostor v paměti se procesu jeví tak, že začíná na nule a lineárně je uspořádán za sebou až do velikosti, kterou potřebuje nebo i větší, než je skutečná velikost RAM. Reálně v paměti takovéto souvisle přidělené bloky adresních prostor pro jednotlivé procesy nenalezneme. Virtuální adresní prostor je rozdělen na jednotlivé kusy, kterými je reálná paměť vyplňována z důvodu optimalizace využití paměti. To umožňuje snadno alokovat stránky jednotlivým úlohám. Pokud je operační paměť plná, začnou se data, se kterými dlouho nebyla prováděna žádná operace, v tuto chvíli nepotřebná, přesouvat z hlavní paměti na úložiště pevného disku. Hlavní paměť i paměť disku je rozdělena na stránky s pevnou délkou 4 kB (v nejmenším adresovacím režimu). Mezi diskem a hlavní pamětí dochází k přesunům stránek, které jsou aktuálně potřeba. U stránkování rozlišujeme lineární adresu, která je pomocí algoritmu převáděna na adresu fyzickou. (5, 11, 12, 13)

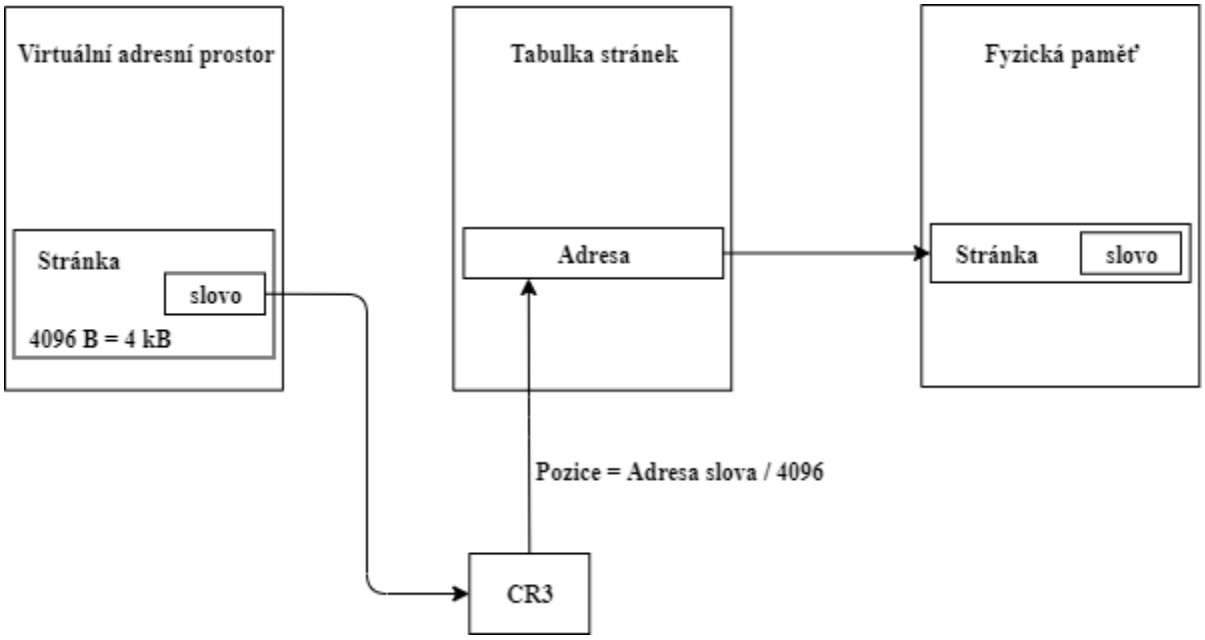
Lineární adresa se skládá z 32 bitů:

1. Prvních 10 bitů nese informaci indexu do stránkovacího adresáře
2. Dalších 10 bitů je vyhrazeno pro index do tabulky stránek
3. Obsahem posledních 12 bitů je offset ve stránce

Procesor vykonává úlohu a potřebuje přistoupit na určité slovo alokované ve virtuálním adresním prostoru nějaké stránky viz Obrázek č. 5. Načte adresu slova, kterou vydělí 4 kB (získá prvních 20 bitů adresy) a načte adresu z registru CR3, který se nachází v procesoru. Ten svojí adresou udává, kde v paměti začíná tabulka stránek. Následně sečte adresu CR3 s výsledkem po 4 kB dělení. Výsledkem je číslo, které udává pozici v tabulce stránek, ze které budeme číst poslední adresu. Ta uvádí, kde ve fyzické paměti opravdu začíná hledaná stránka. Tuto adresu sečte s offsetem (posledních 12 bitů z adresy slova) a dostane skutečnou adresu ve fyzické paměti odkud budeme data číst. (5, 11, 12, 13)

Pokud je třeba přepnout se na provádění jiné úlohy, je potřeba nejprve přepsat registr CR3, čímž dojde v procesoru k přemapování adresních prostor v celé virtuální paměti, protože proces, se kterým je potřeba pracovat má data uložená v paměti jinde. Pokud by procesor při každém přepnutí nejprve četl adresy z tabulky stránek, než přečte nebo zapíše

data z paměti, nedělal by nic jiného než tyto operace. Došlo by ke snížení výkonu celého procesoru z důvodu dvojnásobného přístupu do paměti, než je ve skutečnosti potřeba. Řešením je translation lookaside buffer – TLB. (5, 11, 12, 13)



Obrázek č. 5 – Čtení dat z virtuálního adresního prostoru [20]

Translation lookaside buffer – TLB

Jedná se o specifickou hardwarovou cache paměť v procesoru, která zahrnuje mapování položek z tabulky stránek, do kterých bylo v nedávné minulosti procesorem přistupováno a předpokládá se jejich použití v nejbližší budoucnosti. Při přístupu do paměti procesor nejprve prohledá TLB, když odhalí absenci položek, které hledá, přistoupí do registru CR3. Následně si načte z tabulky stránek adresu, kde ve fyzické paměti je hledaný obsah, ten si poté přenese do TLB, aby při příštím přístupu do paměti mohl data číst rovnou ze stránek z TLB. Tím se ušetří čas z načítání a zvýší se výkon procesoru.

Přepnutím na provádění jiné úlohy je potřeba přepsat registr CR3, tím dojde k přemapování paměti a obsah v TLB je neplatný, protože obsahuje data z jiné tabulky stránek, než využívá přepnutý proces. Při změně registru CR3 operační systém spouští instrukci `INVLPG` (Invalidate TBL Entry), která zneplatní obsah celé cash paměti TLB. Tato operace je z hlediska rychlosti procesoru velmi pomalá, proto je vhodnější se této instrukci, pokud možno vyvarovat. (5, 12)

Pavlík (5) uvádí: *„Zejména si představte, že by se tohle mělo dít nejenom, když přepínám mezi ls a bashem, což se neděje tak často, maximálně stokrát do vteřiny, ale kdybych chtěl přepínat tímhle způsobem mezi adresním prostorem bashe nebo nějakého adresního výpočtu nebo databáze a jádra. Protože jádro také potřebuje mít svůj adresní prostor, kam normální procesy nemůžou.“*.

Adresní prostor je pro každou úlohu oddělen a je hlídáno, aby nebylo přistupováno úlohou jinam mimo přidělený paměťový rámec. To je důležité z hlediska bezpečnosti dat v paměti, především, aby nebylo umožněno číst data z jádra. Pavlík (5) dále uvádí, že takovou operaci právě bezpečnostní hrozba Meltdown dovoluje.

Aplikační program smí žádat služby nabízené operačním systémem, resp. jeho jádrem například přístup k hardwarovému zařízení nebo komunikaci s jinými procesy. Pokud by bylo vstoupeno rovnou při požadavku do kódu jádra, mohlo by dojít k jeho porušení, které by způsobilo pád systému. Proto se musí využít systémového volání `SYSCALL`, které vyvolá službu jádra takovým způsobem, aby neohrozil běh jádra.

Pokud bychom měnili registr CR3 a na každém volání `SYSCALL`, které je možno provádět i milionkrát za sekundu, procesor by ztratil svůj veškerý výpočetní výkon a měnil by pouze registr CR3 a volal instrukci `INVLPG` pro smazání TLB. (5, 12, 14, 15)

Supervisor bit

V tabulce stránek se na jednotlivých pozicích nacházejí adresy odkazující na fyzická místa, kde jsou skutečně uloženy stránky v paměti. Každá taková adresa má své nejnižší bity rovny nule. Je to dáno tím, že adresy odkazují na stránky v paměti, které vždy začínají na offsetu, který je dělitelný 4 kB. Protože je těchto 12 bitů nevyužito dají se na ně zaznamenávat příznaky. Ty podle své hodnoty určují, jaké operace byly nebo mohou být se stránkami provedeny. Jedním z nich je bit nazývaný supervisor, který drží informaci o úrovni oprávnění instrukce, která je nutná pro přístup z jádra (ring 0). Zda tato stránka je nebo není přístupná pro běžné procesy. Když při procházení procesorem, tabulkou stránek nebo TLB, nalezneme tento bit pozitivní a nejsme v režimu supervizor (kernel – ring 0), nelze na tyto stránky přistoupit a číst z nich data. (5, 16)

V dnešní době se do paměti každého procesu zároveň mapuje celý obsah paměti jádra. Na procesorech x86-64 je adresní prostor rozdělen na kladné a záporné hodnoty,

přičemž kladný adresní prostor je přidělen procesu úlohy a záporný jádru, kde na všech takových stránkách je nastaven supervisor bit na hodnotu 1. (5, 16)

Při zavolání instrukce `SYSCALL` jsme přepnuti do režimu jádra na úrovni ringu 0. V tomto režimu jsme schopni přistupovat do obou adresních prostor jak jádra, tak i úlohy a lze tedy jednoduše například nakopírovat data načtená z disku. Jedná se o optimalizaci, která je rychlá, ale z hlediska útoků postranním kanálem velmi zranitelná. (5, 16)

3.2.12 Simultánní Multi-threading a Hyper-threading

Simultánní Multi-threading (zkratka SMT) je v informatice další možností, jak zvýšit výkon samotného procesoru, pokud v pipeline vznikají výpočetní mezery. Jedná se především o situace, kdy procesor vykonává paměťově intenzivní práci. Přistupuje často do paměti a tím se ztrácí výkon procesoru. V tu chvíli, by mohl vykonávat další instrukce jiné úlohy souběžně na stejných vykonávacích jednotkách s rozdílem využití jiné sady registrů. Jedná se tedy o rozdělení jednoho fyzického jádra procesoru na dvě virtuální jádra, která jsou označována jako vlákna. Každé jádro tak může zpracovávat dva různé toky instrukcí zároveň a dochází ke zvýšení celkového výkonu procesoru o 5 % - 20 %. Společnost Intel označuje tento proces jako Hyper-threading, nýbrž je to to stejné jako SMT. (5, 17)

3.3 Bezpečnostní hrozby

Novinkou v posledních letech je cílený útok na chyby komponent, především způsobených díky přenesení staré koncepce do moderního prostředí, či samotných výrobních chyb. To se týká především procesorů a jejich zvyšování výkonu. Nejvíce nebezpečnými jsou útoky Spectre a Meltdown. Jejich oprava vyžaduje záplaty jader operačního systému nebo i firmwaru v CPU. To má ovšem neblahý dopad na snižování výkonu těchto součástek.

3.3.1 Spectre

Na začátku roku 2018, byla odhalena zranitelnost procesorů s názvem Spectre, která ovlivňuje moderní procesory při vykonávání predikcí. Útok spočívá v oklamání spekulativního provádění. Pomocí chyby lze přistoupit a číst data v paměti, která mohou odkrývat útočníkovi citlivá osobní data oběti. Spectre dokáže přimět aplikace

k libovolnému přístupu v paměti. Tento typ útoku je proveditelný skoro na všech běžných procesorech od AMD, Intel, ARM, IBM Power a dalších. (18)

Spectre-V1

Spectre varianta 1 se zabývá útokem skrze špatné předpovědi podmíněného větvení (využití podmínky mispredikce). Útočník se snaží přimět prediktor procesoru větvení přistoupit na kus kódu, který by neměl být v rámci dodržení podmínky vykonán. Jedná se tedy o útok, kdy při nalezení podmínky v kódu budeme procházet větví kódu, která byla podmínkou zamítnuta. Toto nesprávné predikování spekulativního provádění umožní útočníkovi číst data v adresním prostoru programu. (19)

Příklad podmínky v kódu pro útok Spectre-V1 (18):

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

Pro přiblížení, jak tento útok funguje předpokládejme podmínku, kdy kód uvedený výše, je součástí systémového volání nebo knihovna přijímající unsigned integer (v programování se jedná o celočíselný datový typ bez znaménka). V tomto případě je reprezentován proměnnou `x`, jejíž obsah přiděluje útočník. (18)

Pokud bude chtít útočník získat data, ke kterým nemá přístup a využije útoku Spectre-V1, bude postupovat následovně. Jeho cílem bude zavolat například tisíckrát funkci jádra, v tomto případě `SYSCALL` s parametrem 0 pomocí `for` cyklu, viz níže Ukázka kódu (5) pro využití útoku Spectre-V1. (5, 18)

Musí se jednat o funkci v jádře, které obsahuje Double Indirection (`t = a[b[idx]]`). Jedná se o využití dat, která se nachází v cache, pomocí nichž dokážeme se znalostí algoritmu určit, kde a jaké se ve fyzické paměti nacházejí čtené hodnoty. Díky znalosti hodnoty `t` a `idx` v poli `b[]` lze jednoduše odvodit hodnota v poli `a[]`. (20, 18)

Následně funkce `SYSCALL` pomocí podmínky `if` zkontroluje, že předávaný parametr `idx` je menší než velikost pole, do kterého se bude adresovat. Bez této podmínky, by šla číst celá paměť i bez útoku Spectre. Díky tomu, že byla tato funkce volána v minulosti tisíckrát a dopadla pozitivně, útočník natrénoval branch predictor, aby

při kontrole podmínky `if` vždy do pipeline načítal instrukce uvnitř `if`u, protože je podmínka splněna a urychlí se tím činnost procesoru. Útočník dále pomocí funkce `clFlash` vyprázdní celý obsah L1 cache. Od této chvíle je zapotřebí všechna data načítat nejprve z paměti, protože L1 byla vymazána. Zavoláním funkce `SYSCALL` s parametrem 1000 nyní útočník oklame prediktor, protože podmínka `if`, která v předchozích tisících krocích dopadla pozitivně nyní dopadne negativně, ale prediktor začne ještě před zamítnutím průchodu do podmínky již zpracovávat `t = a[b[idx]]` což je chyba. Spekulativně se načte hodnota z pole `b[]` naindexována hodnotou 1000 a přistoupí se do pole `a[]` na téže adrese pole `b[]`. Do paměti L1 jsou nyní zapsána data z paměti z adresy, která odpovídá hodnotě pole `b[]`. Posledním krokem je pomocí `for` cyklu projít celou paměť cache a změřit časy potřebné pro přístup na jednotlivé položky. Ty, které jsou v cache paměti potřebují výrazně méně času pro jejich načtení. Díky tomu útočník přesně ví, kam procesor spekulativně přistoupil do paměti a díky Double Indirection, odvodí hodnotu pole `b[]`. Výhodou útočníka je to, že může využít za hodnotu idx jakoukoli hodnotu, protože se jedná o unsigned integer, čímž může naadresovat celou paměť jádra a tím je dokáže i přečíst. Data mohou být čtena díky tomu, že vykonávání uživatelské aplikace a jádro sdílí stav branch predictoru a cache. (5, 18, 19, 20)

Samotnou chybou v procesoru je přímo fungování spekulativního vykonávání. Pokud je v průběhu programu zjištěna chyba spekulativní předpovědi, měla by se všechna data o této špatné předpovědi smazat. Jedná se především o změny v registrech a příznacích, nicméně se nezmění stav cache a predictoru, což dovoluje ovlivnit provádění aktuálních instrukcí pomocí minulého stavu prediktoru. (5, 18, 19, 20)

Základní opravou bylo přidání funkce `lfence` hned za podmínku `if`. Ta slouží jako bariéra a při jejím načtení zastavuje spekulativní vykonávání. Tím dochází k výraznému snížení výkonu procesoru, protože při každém zavolání systémové funkce se nevykoná předpověď. Lepší variantou opravy této chyby je po přístupu do pole `b[]` omezit hodnotu `idx` pomocí aritmetické operace, která zajistí, že přistoupíme vždy do rozsahu paměti a tím nebude třeba vykonávat spekulativní předpovědi. (5, 18, 19, 20)

Ukázka kódu (5) pro využití útoku Spectre-V1:

```
// Kód útočníka
for (i = 0; i < 1000; i++){
     SYSCALL(0);
}

clFlush();

SYSCALL(1000);

for (i = 0; i < CACHE_SIZE; i++){
     MEASURE_ACCESS(i);
}

// Kód v jádru
int SYSCALL(int idx){
     if(idk < 16){
          t = a[b[idx]];
     }
     else{
          // ošetření chybového hlášení
     }
     return 0;
}
```

Spectre-V2

Útok pomocí Spectre varianty dva je prováděn skrze slabinu v Indirect branch predictoru. Jedná se o místa, kde je pomocí objektu volána metoda nebo volána funkce pomocí pointeru (ukazatele). Způsob fungování kódu je obdobný útoku Spectre-V1.

Nejprve je pomocí `for` cyklu tisíckrát volána vlastní funkce viz níže Ukázka kódu (5) pro využití útoku Spectre-V2. Dochází k natrénování prediktoru, na jakou adresu má při volání funkce skočit. I když prediktor dokáže pracovat pouze na posledních 20 bitech adresy, a ne na celém jejím rozsahu, útočníkovi to nevadí, protože prediktor natrénuje v prostoru pro uživatelskou aplikaci a poté je zajištěno totožné fungování i v jádře díky prediktoru, který je sdílen pro oba adresní prostory a nevymazává informace předcházející. Dále útočník vymaže cache pomocí `clFlush` a zavolá systémové volání `SYSCALL(parametr)` s jakýmkoliv parametrem. Stačí vybrat takové systémové volání, o kterém dopředu víme, že obsahuje nepřímé volání, na které byl prediktor natrénován. V okamžiku, kdy procesor narazí na nepřímé volání `objekt → metoda();` není jasné, na jakou adresu se má skočit, nicméně prediktor předpovídá skok mimo adresní prostor

jádra na adresu vybranou útočníkem. Skáče tam, kam přistupoval dříve pomocí útočníkovi funkce `struet → funkce();`. Útočník naučil prediktor přistupovat na adresu, která obsahuje Double Indirection (dvojitou dereferenci). Dále je útok shodný jako ve variantě jedna. Do cache jsou zapsána nová data. Pomocí `MEASURE_CACHE();` dochází k proměření časů přístupů do paměti L1. Nyní útočník ví, kam procesor přistoupil a díky dvojité dereferenci si přečte i data. (5, 18)

Indirect Branch Restricted Speculation – IBRS

Místa, kde jsou aplikována nepřímá volání je nespočet, a proto je výhodnější upravit fungování procesoru. Opravou je nahrání nového mikrokódu, přidání nového registru MSR, do procesoru, které pozmění jeho chování i ovládání prediktorů. `IBRS` (Indirect Branch Restricted Speculation) je nová funkce pro omezení spekulativního vykonávání v procesoru. Jedná se o pomíjení všech spekulativních údajů vykonaných v prostoru pro aplikace tzv. user space.

Pavlík (5) dále uvádí, že fungování části jádra po přidání `IBRS` zatím není oficiálně zveřejněno ani dokumentováno. Nejpravděpodobněji, ale jádro pracuje tak, že procesor označuje všechny položky v branch prediktoru, které pocházejí z user space a ty v jádru ignoruje, pracuje pouze s nově přidanými. Po přepnutí zpět do user space jsou označené položky opět používány, aby bylo možno využít plnou kapacitu celého prediktoru.

Nevýhodou je ztráta výkonu, protože při každém přepnutí do jádra je nutné vykonat tuto operaci a výsledkem je neobsazený branch prediktor. Jádro v toto chvíli neví, na jaké adresy bude skákat a budou vznikat mezery v pipeline. Tato ztráta výkonu není až tak závažná, protože jádro běžně zpracovává 0,5 % - 1 % doby běhu aplikace, všechny ostatní výpočty provádí aplikace v závislosti na dobře odladěném operačním systému. Dle četnosti systémových volání a jejich délky je možno ztratit až 30 % výkonu procesoru. (5, 18, 21, 22)

Indirect Branch Predictor Barrier – IBPB

Další funkce má zkratku IBPB (Indirect Branch Predictor Barrier). Ta při každém přepnutí úlohy, vymaže všechny údaje spekulativního vykonávání. Ztráta výkonu je výrazná z důvodu učení prediktoru při každém přepnutí. Smazání je nutné vykonávat pro

zabránění útoku z procesu na proces. Zejména je to důležité v kryptografických procesech využívající šifrování a sady klíčů. (5, 18, 21, 22)

Ukázka kódu (5) pro využití útoku Spectre-V2:

```
// Kód útočníka
for (i = 0; i < 1000; i++){
     struet -> funkce();
}

clFlush();

SYSCALL(parametr);

MEASURE_CACHE();

// Kód v jádru
int SYSCALL(parametr){
     objekt -> metoda();
}
```

Retpolíny

Retpolines (česky retpolíny) je alternativní mechanismus vyvinutý společností Google proti napadnutelnosti nepřímých volání. Jedná se o nahrazení veškerých skokových instrukcí za mikrokód. Jedna instrukce je nahrazena více instrukcemi, které vyžadují změnu především v kompilátorech, v jádru a na dalších místech. Hlavní výhodou používání retpolín je ignorace IBRS, které mnohem více omezuje činnost jádra. Retpolíny zaměňují nepřímá volání za návratové instrukce return. Mikrokód dále obsahuje konstrukci, kdy je predikce určení návratové adresy odchycena do nekonečné smyčky a dále je vykonáváno pouze skutečné vykonávání instrukcí, které určí návratovou adresu. Není nutné nijak redukovat spekulativní vykonávání, díky jejímu odchycení. V reálném prostředí se používají retpolíny a IBPB. (5, 18)

Single Thread Indirect Branch Predictors – STIBP

STIBP (Single Thread Indirect Branch Predictors) souvisí s Hyper-threadingem. Jádro může vykonávat dvě vlákna zároveň a díky společnému prediktoru a paměti cache pro obě vlákna je sdílen i jejich obsah. Lze tedy útočit z jednoho vlákna na druhé bez nutnosti přístupu do jádra. Společnost Intel řeší tento problém pomocí rozdělení jednoho fyzického prediktoru na dva virtuální. Tím jsou separovány predikce pro každé vlákno

zvlášť. Tato oprava je aplikována mikrokódem jen na procesorech řady Skylake a novějších. Tato operace sice přinese zabezpečení proti Spectre útoku, ale sníží výkon procesoru až o dalších 30 %, protože spekulativní vykonávání je implementováno hardwarově a nyní se pozastavuje nebo nahrazuje mikrokódem. Proto dochází ke spekulacím, zda není lepší vypnout Hyper-threading a ztratit pouze 20 % výkonu. To není možné z důvodu nastavení běhu aplikace vývojáři na různých vláknech, tím by museli být přepsány všechny programy s takovouto konfigurací. (5, 23)

3.3.2 Meltdown

Meltdown je třetí variantou útoku skrze postranní kanál a majoritně se týká procesorů Intel, IBM Power a některých z řad ARM. Útočníkovi je dovoleno skrze napadenou aplikaci číst celou paměť, i když k ní nemá oprávnění. Jednou z oprav společnosti Intel pro snižování ztráty výkonu procesoru na aplikovaných záplatách, bylo rozhodnutí zanedbání zamítnutí přístupu ve spekulativním vykonávání. V rámci spekulací je jedno co se předpovídá a pokud je predikce špatná, vše je zapomenuto a smazáno. Jak již bylo zmíněno u útoku Spectre, smazáno je vše až na stav cache, branch prediktoru, TLB a přidružených částí. (5, 24)

Při útoku Meltdown se útočník snaží číst reálnou část paměti, do které nemá jako běžný uživatel oprávnění. Útočníkovi stačí spustit vlastní kód a nechat ho počítač provádět s běžně nastavenými uživatelskými právy. Nicméně útočník nemá fyzický přístup k napadenému stroji. Dále je možno předpokládat i nejmodernější softwarové zabezpečení stroje a zároveň ideální operační systém zcela bez chyb. Neexistuje slabina, kterou by bylo možno získat oprávnění na úrovni jádra nebo kterou by unikaly informace ze systému ven. Útočník se snaží vlastním kódem (viz níže Ukázka kódu (5) pro využití útoku Meltdown) s tajnou hodnotou uloženou ve fyzické paměti natrénovat prediktor tak, aby podmínka kontroly `if`, zda predikoval špatně, byla vyhodnocena jako `false`. V reálném prostředí je nejdříve smazán obsah cache pomocí `clFlush();` a poté je vyhodnocení vlastní predikce považováno za správnou a do ifu se nevstupuje. Ve spekulativním prostředí dojde k přistoupení do `if`u, a provedení příkazu `a[kernel[idx]] = 1`. Dochází tak k pokusu vstoupit do adresního prostoru jádra s předanou hodnotou idx, což je nějaká hodnota adresy v jádře. Tím přistoupí po paměti a načte hodnotu, kterou uloží v adresním prostoru user space s tajnou hodnotou, v tomto případě 1. Tato hodnota je propsána i do

cache L1. Poté dochází k vyhodnocení predikce, které upozorní na špatnou předpověď a všechny data o této chybné predikci by měla být smazána. V user space se nyní už nenachází pole `a[]` s hodnotou rovno 1, ale protože nedošlo k promazání cache L1, je možné si změřit čas přístupu do pole `a[]`, tím útočník zjistí, na kterou položku se přistupovalo a dokáže odvodit, že na adrese `idx` v jádře se nachází příslušný bajt za pomoci dvojité dereference. Nutné je podotknout, že celý tento útok se odehraje v prostředí user space nikoliv v jádru a je poté možno si přečíst celou paměť kernelu. (5, 24)

Ukázka kódu (5) pro využití útoku Meltdown:

```
// Kód útočníka
u8 meltdown(long idx){
     clFlush();

     if(mispredicted){
          a[kernel[idx]] = 1;
     }

     return MEASURE_CACHE(a);
}
```

U procesorů AMD se tato chyba nevyskytuje z důvodu používání Supervisor bitu a dopředného kontrolování tohoto bitu před samotným spekulativním provedením. Procesory od Intelu nejprve provedou spekulativní provádění a až teprve poté vyhodnotí Supervisor bit, podle kterého zamítnou další operace. Tato souslednost byla navržena z důvodu optimalizace a zvýšení výpočetního výkonu. (5, 24)

Kernel Page Table Isolation – KPTI

Řešením pro útok Meltdown je oprava KPTI (Kernel Page Table Isolation) neboli vynechání Supervizor bitu a přepínání celého adresního prostoru. Pokud se v tabulkách stránek jádro vůbec nevyskytuje, tak tam ani nelze spekulativně přistupovat, protože není možno odvodit další adresu a končí s chybovou hláškou. Toto řešení nutí vykonávat při každém systémovém volání instrukci INVLPG, čímž dochází až ke čtyřnásobnému zpomalení výkonosti procesoru. (5, 24)

Process Context ID – PCID

Částečného zmírnění tak velkého zpomalení je možnost využití PCID (Process Context ID). Hlavním užitím této funkce je optimalizace při přepínání mezi úlohami, přičemž lze její funkčnost využít i pro pčepínání mezi jádrem a úlohou. Všechny položky v TLB jsou označeny číslem PCID, které je zapisováno do nejnižších bitů registru CR3. Procesor nemá poté možnost operovat s jinými položkami než s těmi, které se nacházejí v CR3. Tím není nutné při systémovém volání provádět mazání celé paměti cash a mohou být vedle sebe položky z jádra i z user space. Je nezbytné, ale při přepsání adresy v CR3 zároveň přepsat hodnotu PCID. Toto vylepšení zmírní ztrátu výkonu na 50 %. (5, 24)

3.3.3 Foreshadow

Útok L1TF, lidově Forshadow, je technicky podobný útoku Meltdown. Útok je veden na další stavový bit u jednotlivých položek v tabulce stránek. Jedná se o Present bit. Jeho stav je měněn podle aktuálního umístění stránky v paměti. Pokud je stránka jádrem vyhodnocena v tuto chvíli jako nepotřebná je přemístěna na pevný disk a Present bit je nastaven na příslušnou hodnotu. Tomu se říká Swapování. Při přístupu procesoru do paměti, je zjišťováno, zda se stránka fyzicky nachází v paměti. Pokud se procesor pokusí přistoupit na takové místo, které není v paměti, respektive snaží se přistoupit na stránku s nastaveným present bitem, je vyvolána výjimka a počká, až je stránka nahrána z disku a teprve poté pokračuje v dalším provádění procesu. (20)

Při spekulativním provádění je tento bit přehlížen a tím je umožněno číst data ze stránek kam ukazují položky, které ve fyzické paměti nejsou. Pokud jádro odswapuje určité stránky na disk, u jednotlivých položek si ukládá informaci, malé adresní číslo, kde přesně na pevném disku jsou zapsány a na kterém zařízení. Díky tomu. Že jsou tato čísla ve svém rozsahu malá, lze díky nim číst pouze malý úsek začátku paměti. Na začátcích paměti nebývají uloženy žádné citlivé informace, tím je tento útok relativně málo nebezpečný. (20)

Opravou této chyby je změna bitů v nejvyšších adresních bitech odswapované položky při změně Present bitu. Tím je ukazováno na část paměti, která je prázdná. Jedná se o ošetření chování při změně Present bitu, které díky přidání dalších instrukcí snižuje výkonost procesoru. (20)

Virtualizace

Využití chyby Foreshadow je kritické na virtuálním stroji. Především na úrovni emulace hardwaru, která je jednou z technik virtualizace. Jeden fyzický server obsahuje hypervizor, který řídí a odděluje chod pro ostatní virtualizované počítače s možností běhu různých operačních systémů. Při běhu dvou operačních systémů ve virtualizovaném prostředí, má každý ze systémů plný přístup ke svým tabulkám stránek. Při odswapování stránek na pevný disk, nastavení Present bitu a změnění nejvyšších adresních bitů, spustí virtuální stroj spekulaci, která ignoruje nejen Present bit, ale i celkovou virtualizaci a přistoupí na místo prostřednictvím L1 cache. Pokud útočník využije dvojité indiference, které je posíláno do spekulativního vykonávání, může číst data z L1 paměti a díky virtualizaci lze kontinuálně číst celý obsah L1 cache, která je sdílená pro ostatní virtuální stroje. Lze tedy číst citlivá data, která používal jiný operační systém.

Opravou je smazání L1 cache při každém přepnutí na jiný virtuální stroj, tedy vždy, kdy je hypervizorem volána instrukce `VMENTER()` pro vstup do virtuálního stroje. To způsobuje velký dopad na ztrátu výkonu. Proto je dále nutno zavést optimalizaci hypervizorů tak, aby co nejméně volali instrukci pro změnu virtuálního prostředí. (20, 25, 26)

Pavlík (20) vysvětluje jednu z možností, jak se taková optimalizace v reálném světě dá provést: *„Jednotlivé virtuální stroje si připíchnete na jednotlivá jádra a pokud možno nepřepínáte a hypervizor zoptimalizovaný tak, aby dělal co nejméně hypercallů, aby se vše řešilo přes sdílenou paměť a pak to běží relativně dobře.“*.

Problém nastává, pokud virtuální stroje běží v režimu Hyper-threadingu. Každý virtuální stroj je spuštěn na jednom vláknu. V takovém případě nelze mazat L1 cache, navíc k této operaci ani hypervizor nikdy nemůže přistoupit. Díky možnosti čtení dat z jednoho i druhého vlákna na stejném jádře a využívání virtualizace na procesoru napadnutelným útokem Foreshadow, je nutno vypnout Hyper-threading a tím také ztrátu až 50 % výkonu. Jedním z příkladů takového dopadu je u poskytovatelů cloudových služeb, kdy jsou nuceni zdvojnásobit svojí výpočetní kapacitu pro zachování stávající rychlosti serverů, aby předešli útoku L1TF. (20, 25, 26)

4 Vlastní práce

V této části bude proveden výběr operačního systému a hardwaru. Dále připraveny scénáře pro výkonnostní testy na měření ztráty výkonu. Uskutečněny budou jak ve fyzickém, tak i virtuálním prostředí. Jednat se bude především o vliv na vstupní, výstupní výkon a hyper-threading. Dále bude provedeno vypínání jednotlivých služeb počítače a testován jejich dopad na ztrátu výkonu. Volba operačního systému a komponent je důležitým aspektem, který je nutno vykonat před vytvořením jednotlivých scénářů pro měření ztráty výkonu. Dále bude provedeno statistické testování jednoduché analýzy rozptylu a mnohonásobného porovnávání za účelem zhodnotit průkaznost změny naměřeného výkonu.

4.1 Výběr operačního systému a virtualizačního prostředí

S vlivem zranitelností procesorů souvisí také odladěnost samotného operačního systému a jeho schopnost dokázat vypnout jednotlivé opravy, které sice udělají systém zranitelným, ale zato výkonnějším. Problém útoků na procesory pomocí postranního kanálu se týká všech uživatelů počítačů. Všechna testování budou provedena na operačním systém Linux Ubuntu 19.10 s verzí jádra 5.3.0-40-generic (x86_64), která je nejnovější verzí tohoto systému a zároveň nejrozšířenější mezi uživateli. Tento operační systém vybral z důvodu možnosti vypnutí zranitelností procesoru pomocí změn parametrů v jádře.

Pro provedení testů na virtualizovaném systému byl zvolen Proxmox Virtual Environment 6.1. Proxmox, jedná se o serverové virtualizační prostředí, které je postaveno na Linuxové distribuci Debian. Využívá KVM (kernel-based virtualization machine), který umožňuje fungovat jako hypervizor typu 1, tedy plnou virtualizaci, protože běží na hostovaném fyzickém hardwaru bez nutnosti vrstvy operačního systému. Dokáže spravovat virtuální stroje a kontejnery.

4.2 Výběr komponent

Neméně důležitou částí je výběr vhodných počítačových komponent pro následné testování. Je nutno uvažovat stávající situaci, kdy existuje jen malé množství uživatelů, kteří disponují vždy nejmodernějšími a nejvýkonnějšími počítačovými komponentami. Běžný uživatel mění vlastní hardware v průměru jednou za pět let, ať už z důvodů poruchy nebo nedostatečného výkonu v porovnání s nejmodernějšími technologiemi. Pro testování

byly vybrány procesory vyrobené v období 2010 – 2017, čímž bude pokryta široká škála pro srovnání výkonových úbytků procesorů v běžném i vytíženém provozu desktopů tak i virtualizovaných serverů.

4.2.1 Testovací komponenty

Dle autorova výběru byly do testování zařazeny tři počítačové sestavy. Pro jejich přehlednost budou dále nazývány podle typu procesoru a prostředí, ve kterém bylo testování prováděno. Nejdůležitější komponenty těchto systémů jsou vypsány níže.

Systém č. 1 – i5 - 7267U

Procesor:

- Intel® Core™ i5-7267U
- Sedmá generace procesorů Intel® Core™ i5 s kódovým označením Kaby Lake
- 2 jádra, 4 vlákna
- Základní frekvence 3,10 GHz, Max Turbo frekvence 3,50 GHz
- Cache 4 MB
- 14 nm výrobní proces
- Uveden na trh v prvním kvartálu roku 2017

Operační systém a jádro:

- Ubuntu 19.10
- 5.3.0-40-generic (x86_64)

Paměť RAM:

- 8 GB, LPDDR3 SDRAM

Úložiště:

- 500 GB, 2,5“ SSD Western Digital 3D NAND

Systém č. 2 – i7 - 4790K

Procesor:

- Intel® Core™ i7-4790K
- Čtvrtá generace procesorů Intel® Core™ i7 s kódovým označením Haswell s otevřeným násobičem pro přetaktování
- 4 jádra, 8 vláken

- Základní frekvence 4,00 GHz, Max Turbo frekvence 4,40 GHz
- Cache 8 MB Intel® Smart Cache
- 22 nm výrobní proces
- Uveden na trh v druhém kvartálu roku 2014

Operační systém a jádro:

- Ubuntu 19.10
- 5.3.0-40-generic (x86_64)

Paměť RAM:

- 2x8 GB, DDR3, HyperX Fury Black

Úložiště:

- 500 GB, 2,5“ SSD Western Digital 3D NAND

Systém č. 3 – i7 - 740Q

Procesor:

- Intel® Core™ i7-740Q
- Legacy Intel® Core™ i7 s kódovým označením Clarksfield
- 4 jádra, 8 vláken
- Základní frekvence 1,73 GHz, Max Turbo frekvence 2,93 GHz
- Cache 6 MB Intel® Smart Cache
- 45 nm výrobní proces
- Uveden na trh ve třetím kvartálu roku 2010

Operační systém a jádro:

- Ubuntu 19.10
- 5.3.0-40-generic (x86_64)

Paměť RAM:

- 2x2 GB, DDR3 SOIMM

Úložiště:

- 500 GB, 2,5“ SSD Western Digital 3D NAND

4.3 Příprava scénářů

Bude provedena série třinácti testů opakovaně, z důvodu různého nastavení služeb na úrovni jádra. Pro fyzickou část budou provedeny čtyři různá nastavení jádra, pro virtualizovanou tři. Každý procesor bude postupně otestován ve všech nastaveních. Dojde

k testování procesorů s nastavením různých příznaků v jádře systému, která budou měněna pomocí konfigurace GNU GRUB zavaděče operačního systému. GRUB má vlastní konfigurovatelný soubor, který je umístěn v **`/etc/default/grub`**.

Úpravou souboru na řádce **`GRUB_CMDLINE_LINUX=""`** lze měnit jednotlivé příznaky jádra, které dokážou vypínat a zapínat jednotlivé záplaty Spectre, Meltdown a dalších podobných útoků. Po úpravě a uložení souboru je třeba spustit příkaz **`update-grub`**, který změní samotný konfigurační soubor zavaděče umístěný v **`/boot/grub/grub.cfg`**. Posledním krokem k aplikování změn pro jádro je nutné nainstalovat zavaděč s novým nastavením na správné místo na disku. Nejběžněji pomocí příkazu **`grub-install /dev/sda`** a poté restartovat stroj. Ve virtualizované části budou příznaky měněny na obou úrovních jádra, tedy na úrovni serveru (hypervizoru) i virtuálního stroje.

Fyzická část bude provedena ve čtyřech režimech nastavení. V prvním režimu `off` bude testován výkon, kdy budou všechny opravy, které byly přidány z důvodu zajištění bezpečnosti, vypnuté. Jde tedy o stav a výkon, ve kterém se počítač nacházel před identifikací hrozeb Spertre, Meltdown. V režimu `Spe=on` budou zapnuty záplaty proti útoku Spectre v1 a v2, ostatní vypnuty. V režimu `Mel=on` budou zapnuty záplaty pouze proti útoku Meltdown, ostatní vypnuty. V posledním režimu `auto` budou zapnuty všechny záplaty proti Spectre v1, v2, Meltdown. Jedná se o stav, který je v dnešní době výchozím u většiny počítačů a počítač je tedy proti těmto útokům ochráněn.

Virtualizovaná část bude provedena ve třech režimech nastavení. V prvním režimu off bude testován výkon bez zapnutých záplat na Spectre v1, v2 a Meltdown. V druhém režimu auto bude testován výkon za normálních podmínek s aktivními záplatami proti všem útokům s výjimkou Foreshadow. Poslední režim nosmt je režim, kde jsou zapnuty všechny mitigace proti útokům Spectre, Meltdown i Foreshadow. V tomto režimu je vypnutý Hyper-threading. Virtualizovaná část testování bude ve všech případech pro rozlišení režimů podbarvena žlutou barvou.

Fyzická část		
č.	**Název testování**	**Nastavené příznaky**
1.	auto	`GRUB_CMDLINE_LINX="mitigations=auto"`
	Poznámka: Jedná se o výchozí nastavení příznaků jádra. Všechny dostupné mitigace jsou zapnuty.	
2.	Spe=on	`GRUB_CMDLINE_LINUX="noibrs noibpb nopti l1tf=off nospec_store_bypass_disable no_stf_barrier mds=off"`
	Poznámka: Vypnuty jsou všechny dostupné mitigace kromě Spectre v1 a Spectre v2.	
3.	Mel=on	`GRUB_CMDLINE_LINUX="noibrs noibpb kpti=1 nospectre_v2 nospectre_v1 l1tf=off nospec_store_bypass_disable no_stf_barrier mds=off"`
	Poznámka: Vypnuty jsou všechny dostupné mitigace kromě Meltdown.	
4.	off	`GRUB_CMDLINE_LINUX="mitigations=off"`
	Poznámka: Vypnuty jsou všechny dostupné mitigace a systém je zranitelný vůči všem útokům.	

Tabulka č. 1 – Natavení parametrů jádra pro fyzickou část testování [autor]

Virtualizovaná část		
č.	**Název testování**	**Nastavené příznaky**
1.	auto	`GRUB_CMDLINE_LINUX="mitigations=auto"`
	Poznámka: Jedná se o výchozí nastavení příznaků jádra. Všechny dostupné mitigace jsou zapnuty.	
2.	nosmt	`GRUB_CMDLINE_LINUX="mitigations=auto, nosmt"`
	Poznámka: Zapnuty jsou všechny mitigace a navíc je vypnutý Hyper-threading proti napadnutelnosti systému pomocí útoku L1TF.	
4.	off	`GRUB_CMDLINE_LINUX="mitigations=off"`
	Poznámka: Vypnuty jsou všechny dostupné mitigace a systém je zranitelný vůči všem útokům.	

Tabulka č. 2 – Natavení parametrů jádra pro virtualizovanou část testování [autor]

4.4 Testy a naměřené hodnoty

Pro testování vstupně výstupního výkonu, výkonu procesoru i celého systému jako celku autor zvolil testovací software Phoronix Test Suite. Jedná se o sadu, která provádí automatické testy. Nabízí více než 450 testovacích profilů, které jsou přizpůsobitelné. Sám

dokáže stáhnout testovací závislosti, jejich instalaci i provedení testů k dosažení výsledků, které je možno nahrát a uložit na webový server OpenBnechmarking.org, kde mohou být výsledky porovnány s ostatními uživateli.

Všechny testy budou automaticky spuštěny postupně za sebou pomocí příkazu:

```
phoronix-test-suite benchmark compilebench fio postmark
build-linux-kernel glibc-bench hackbench openssl
ttsiod-renderer apache nginx pgbench redis stress-ng
```

Byla provedena instalace a nezbytné nastavení systémů i programů pro testování. Všechny výsledky interpretuje pomocí tabulek s naměřenými hodnotami u příslušných testů. Zeleně budou označeny nejlepší výsledky testů a červeně nejhorší z dosažených hodnot. Výsledné hodnoty zanesené v tabulkách jsou průměrné hodnoty, které byly získány z testovacích scénářů. Počet měřených vzorků u každého testu a konfigurace je závisí na hodnotě směrodatné odchylky mezi hodnotami. Základním měřením jsou 3 vzorky. Po naměření hodnot je vypočtena směrodatná odchylka. Pokud je tato hodnota větší než 3,5 % dojde k automatickému navýšení počtu vzorků v měření. Tímto způsobem se pokračuje, dokud není směrodatná odchylka menší než definovaná hladina za účelem zajištění přesnosti výsledku. Maximální dosažená hranice v naměřeném počtu vzorků byla patnáct hodnot, nejčastěji stačilo provést měření se třemi až šesti hodnotami pro každé měření.

4.4.1 Vstupní a výstupní výkon

Testování vstupně-výstupního výkonu (I/O) je důležitou součástí výkonových testů. Každý počítač nebo server komunikuje s vnějším světem a okolními zařízeními. Nejčastější operace, které stroj vykonává je komunikace a předávání dat s diskovými jednotkami sloužící pro trvalé i dočasné uložení dat. Pro testování vstupně-výstupního výkonu diskových jednotek byly zvoleny testy Compile Bench, Flexible IO Tester a PostMark.

Compile Bench

Tento test se zaměřuje na testování souborového systému tím, že simuluje běžné diskové I/O operace jako je vytváření, kompilace, opravování nebo čtení celého větvení jádra (kernel tree). Nepřímo měří, jak dobře dokáže souborový systém udržovat rozmístění adresářů potom co je disk zaplněn a adresáře stárnou. Test je nastaven v režimu – makej

s deseti počátečními adresáři. Makej vytváří, čte a odstraňuje jednotlivé kernel tree, čímž dosahuje rozložení souborů do všech adresářů v jádře. Testuje schopnost udržování metadat a zápisu velkého počtu adresářů najednou.

Testování bude provedeno ve všech nabízených konfiguracích (vyšší naměřené hodnoty jsou lepší):

- Kompilace [MB/s]
- Počáteční vytvoření [MB/s]
- Čtení kompilovaného kernel tree [MB/s]

i7-740Q								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Compile	MB/s	280,0	272,7	275,1	271,8	229,7	250,9	237,5
Initial Create	MB/s	153,8	173,0	155,2	174,8	86,8	87,4	101,0
Read Compiled Tree	MB/s	414,5	529,9	418,2	533,9	246,2	231,8	380,4
i7-4790K								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Compile	MB/s	874,3	879,7	910,1	943,0	398,0	410,0	446,8
Initial Create	MB/s	506,1	556,1	538,0	559,5	264,3	266,6	342,1
Read Compiled Tree	MB/s	2804,6	3335,1	3164,0	3536	679,9	317,7	941,0
i5-7267U								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Compile	MB/s	419,0	432,6	445,5	447,3	282,8	293,3	289,8
Initial Create	MB/s	333,2	347,7	360,8	375,4	212,4	217,7	248,6
Read Compiled Tree	MB/s	474,4	436,5	460,1	427,1	489,4	511,2	994,0

Tabulka č. 3 – Naměřené hodnoty testu Compile Bench [autor]

Flexible IO Tester

FIO je pokročilý nástroj pro testování disků realizované pomocí kernel AIO access library. Test vytváří několik procesů a podprocesů, které simulují běžné I/O operace. Nastavení testu závisí na konkrétním typu měření. Lze zvolit velikost zapisovaných dat, typ I/O operací, I/O engine, I/O depth a další.

Provedeno bude měření s touto konfigurací pro všechny testy: Engine: Linux AIO; Buffered: Yes; Direct: No; Size 4KB (vyšší naměřené hodnoty jsou lepší):

- Náhodné čtení [MB/s, IOPS]
- Náhodný zápis [MB/s, IOPS]
- Sekvenční čtení [MB/s, IOPS]
- Sekvenční zápis [MB/s, IOPS]

i7-740Q								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Random Read	MB/s	20,6	22,9	21,2	22,6	7,112	7,144	7,334
Random Read	IOPS	5274	5879	5423	5770	1774	1782	1830
RandomWrite	MB/s	155	197	169	195	111	114	179
RandomWrite	IOPS	39633	50267	43233	49680	28508	29167	45667
Sequential Read	MB/s	232	251	238	252	193	226	260
Sequential Read	IOPS	59400	64200	61033	64633	49533	49300	57933
Sequential Write	MB/s	200	220	214	235	200	192	235
Sequential Write	IOPS	54533	60133	56367	59800	51350	49267	60100
i7-4790K								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Random Read	MB/s	41,2	31,6	32,1	41,7	30	25,6	26,8
Random Read	IOPS	10633	8163	8199	10667	7682	6538	6939
RandomWrite	MB/s	349	381	383	377	313	326	374
RandomWrite	IOPS	89367	97500	97967	96533	80147	83460	95883
Sequential Read	MB/s	492	503	500	507	466	475	491
Sequential Read	IOPS	126000	129000	128000	130000	119667	121667	126000
Sequential Write	MB/s	383	372	377	381	396	372	437
Sequential Write	IOPS	97960	95400	96400	97567	101433	95060	112000
i5-7267U								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Random Read	MB/s	18,0	15,1	16,2	14,9	16,4	15,0	15,4
Random Read	IOPS	4619	3871	4149	3817	4190	3831	3952
RandomWrite	MB/s	300	341	335	328	188	199	220
RandomWrite	IOPS	76613	87364	85573	83867	48100	50933	56367
Sequential Read	MB/s	380	394	391	396	369	373	378

Sequential Read	IOPS	97400	101000	99867	101333	94433	95067	95791
Sequential Write	MB/s	351	368	356	376	356	339	388
Sequential Write	IOPS	89967	94300	91233	96220	91000	86700	99033

Tabulka č. 4 – Naměřené hodnoty testu Flexible IO Tester [autor]

PostMark

PostMark má za úkol simulovat zapisování malých souborů, které je možné nejčastěji pozorovat na webovém či mailovém serveru. Postupně provádí testování s 25000 transakcemi s 500 soubory v rozmezí 5 až 512 kilobajtů. V první fázi testu jsou vytvořeny jednotlivé soubory, které jsou čteny, připojovány a následně mazány. Postupně se vytvoří 500 souborů a provede 500 transakcí. Následně je test opakován se zvýšenou intenzitou vytížení až na 20000 souborů s prováděním 200000 transakcí. Autor provede test PostMark k měření diskových transakcí [TPS] (vyšší naměřené hodnoty jsou lepší).

i7-740Q								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Disk Transaction Performance	TPS	2659	3061	2787	3219	1720	1712	2330
i7-4790K								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Disk Transaction Performance	TPS	6199	7010	7076	7812	3826	3846	6199
i5-7267U								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Disk Transaction Performance	TPS	5000	6000	6578	7377	3025	3024	5434

Tabulka č. 5 – Naměřené hodnoty testu PostMark [autor]

4.4.2 Výkon procesoru

Při testování výkonu procesorů je možné se setkat s pojmem CPU benchmarking, což je série testů navržených pro jejich měření. Slouží k porovnání různých systémů za použití stejných testovacích metod. Typickými měřitelnými veličinami u procesorů jsou taktování, počet provedených instrukcí za sekundu, doba nutná pro zavolání registrů a také celkové skóre procesoru.

Timed Linux Kernel Compilation

V tomto testu se analyzuje doba, jakou procesor potřebuje k sestavení (kompilaci) výchozího linuxového jádra [sekundy] (nižší naměřené hodnoty jsou lepší).

i7-740Q								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Time To Compile	Seconds	457,1	442,6	454,2	441,6	588,5	1096,4	538,0
i7-4790K								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Time To Compile	Seconds	160,6	158,2	157,8	155,9	191,6	230,9	165,7
i5-7267U								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Time To Compile	Seconds	357,1	328,6	330,8	332,7	411,9	517,9	346,3

Tabulka č. 6 – Naměřené hodnoty testu Timed Linux Kernel Compilation [autor]

Glibc bench

Sada mikro benchmarků glibc automaticky generuje kód specifikovaných funkcí, sestavuje je a následně opakovaně volá pro poskytnutí základních výkonových vlastností. Využívá knihovnu GNU C, která dokáže běžet na různých jádrech i hardwarových architekturách nejčastěji na linuxovém jádru x86. Na procesorech x86 RDTSCP instrukce poskytují přesnější data v časování než instrukce RDTSC. Test volá specifikované funkce jádra a zaznamenává čas jejich vykonání.

Konfigurace testu glibc bench (nižší naměřené hodnoty jsou lepší):

- FFS [nanosekundy] – funkce ffs() vrací první pozici bitu v daném slově
- SQRT [nanosekundy] – funkce sqrt() vrací nezápornou hodnotu druhé odmocniny vstupního čísla
- FFSLL [nanosekundy] – stejný princip chování jako funkce ffs() s rozdílem jiného vstupního argumentu jiné velikosti
- PTHREAD_ONCE [nanosekundy] – tato funkce může být volána vláknem pouze jednou a hlídá, aby kód, který vytváří vlákna byl volán více vlákny

i7-740Q								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
ffs	Nanosec.	3,29221	3,00579	3,23915	2,97390	4,05951	4,01808	3,25734
sqrt	Nanosec.	9,65225	9,25342	9,46042	9,20204	10,7340	10,7860	10,3833
ffsll	Nanosec.	3,30907	3,00744	3,24558	2,97876	4,03006	4,01873	3,25627
pthread_once	Nanosec.	3,30329	3,01179	3,24865	2,97196	4,00456	3,97551	3,28177
i7-4790K								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
ffs	Nanosec.	1,75825	1,50023	1,58493	1,47567	1,96409	1,94712	1,5058
sqrt	Nanosec.	3,18761	2,92997	3,01400	2,90628	3,41219	3,40051	2,94069
ffsll	Nanosec.	1,75780	1,49855	1,58350	1,47595	2,01764	1,95899	1,50847
pthread_once	Nanosec.	1,76306	1,50445	1,58568	1,49543	1,97412	1,94805	1,50877
i5-7267U								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
ffs	Nanosec.	2,23886	1,90583	2,0229	1,88366	2,6189	2,62493	1,89813
sqrt	Nanosec.	2,52833	2,19484	2,31065	2,1678	2,92864	2,92328	2,18062
ffsll	Nanosec.	2,23942	1,90773	2,02212	1,88244	2,60509	2,61791	1,89569
pthread_once	Nanosec.	2,24052	1,90899	2,02827	1,87733	2,61583	2,62035	1,89840

Tabulka č. 7 – Naměřené hodnoty testu Glibc bench [autor]

Hackbench

Jedná se o zátěžový test jádra linuxového plánovače. Hlavním cílem tohoto testu je vytvoření párů vláken nebo procesů komunikujících pomocí soketů nebo pipe. Měří, jak dlouho trvá poslání dat tam a zpět.

Bude uskutečněno testování s následujícími parametry (nižší naměřené hodnoty jsou lepší):

- Počet vláken:
  - 4 [sekundy]
  - 8 [sekundy]
  - 16 [sekundy]

- Počet procesů:
  - 4 [sekundy]
  - 8 [sekundy]
  - 16 [sekundy]

i7-740Q								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
4 Thread	Seconds	95,5	82,1	92,6	79,5	199,2	319,4	167,1
8 Thread	Seconds	160,7	133,1	159,2	132,8	405,9	546,4	345,4
16 Thread	Seconds	331,6	250,8	342,3	250,2	799,0	952,9	656,3
4 Process	Seconds	88,8	75,9	87,569	74,5	189,3	246,5	162,9
8 Process	Seconds	149,9	123,1	144,3	121,1	350,8	443,7	318,3
16 Process	Seconds	282,5	235,1	280,6	229,3	703,3	847,1	519,5
i7-4790K								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
4 Thread	Seconds	27,2	19,2	21,4	24,7	60,9	75,3	39,5
8 Thread	Seconds	56,2	38,5	40,7	39,9	110,0	141,9	65,5
16 Thread	Seconds	116,6	74,0	80,6	72,2	228,8	267,5	94,9
4 Process	Seconds	26,1	17,1	18,2	17,9	54,2	68,1	34,3
8 Process	Seconds	55,6	35,1	38,9	35,6	97,5	117,0	46,6
16 Process	Seconds	111,3	73,6	77,2	64,1	181,5	236,2	72,1
i5-7267U								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
4 Thread	Seconds	65,9	53,5	55,7	57,4	138,4	179,6	70,0
8 Thread	Seconds	137,2	89,8	102,7	102,9	315,4	340,6	131,9
16 Thread	Seconds	271,2	168,8	188,6	187,5	591,9	631,1	210,9
4 Process	Seconds	61,0	35,7	41,2	47,4	118,2	144,6	52,1
8 Process	Seconds	135,2	80,6	88,8	82,1	224,1	265,6	85,6
16 Process	Seconds	282,7	176,5	192,0	147,0	460,5	514,1	170,8

Tabulka č. 8 – Naměřené hodnoty testu Hackbench [autor]

OpenSSL

OpenSSL je nástrojová sada, která implementuje protokoly SSL a TLS. Měří výkonnost RSA 4096-bit. Používá se pro měření výkonnosti procesoru pomocí zpracování

kryptografických algoritmů [počet jednotek za sekundu] (vyšší naměřené hodnoty jsou lepší).

i7-740Q								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
RSA 4096-bit Performance	Signs per Seconds	176,4	177,0	177,8	178,1	172	101	171
i7-4790K								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
RSA 4096-bit Performance	Signs per Seconds	765,3	764,2	764,9	764,4	758	745	759
i5-7267U								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
RSA 4096-bit Performance	Signs per Seconds	517,6	527,0	521,5	524,1	337	319	336

Tabulka č. 9 – Naměřené hodnoty testu OpenSSL [autor]

TTSIOD 3D Renderer

Test provádí renderování 3D objektů v reálném čase pomocí procesoru v různých režimech vykreslování. Podporuje OpenMP a Intel Threading Building Blocks. Měření jsou vyhodnocena v jednotkách FPS – frames per seconds (vyšší naměřené hodnoty jsou lepší).

i7-740Q								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Phong Rendering With Soft-Shadow Mapping	FPS	87,1591	87,4203	87,5296	87,6840	74,7067	39,8758	74,3404
i7-4790K								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Phong Rendering With Soft-Shadow Mapping	FPS	236,880	236,115	237,552	234,743	213,5	174,1	223,8

i5-7267U								
Část testování		**Fyzická**				**Virtualizovaná**		
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**	**auto**	**nosmt**	**off**
Phong Rendering With Soft-Shadow Mapping	FPS	107,956	112,900	110,266	110,206	102,9	82,7	107,3

Tabulka č. 10 – Naměřené hodnoty testu TTSIOD 3D Renderer [autor]

4.4.3 Výkon systému

V této části vybraných testů systému lze nalézt ty, které simulují zpracování dat v různých databázích či další testy, které souvisí s testováním procesorů a systému počítače jako celku.

Apache Benchmark

Nástroj slouží pro měření počtu požadavků za sekundu, které je server Apache schopen zpracovat při obdržení 1000000 požadavků a zpracovávání 100 požadavků soubežně [Requests per second] (vyšší naměřené hodnoty jsou lepší).

i7-740Q					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Static Web Page Serving	Request per Sec.	6796,91	7692,75	6962,62	7895,24
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
Static Web Page Serving	Request per Sec.	4373,0	4384,2	5539,6	
i7-4790K					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Static Web Page Serving	Request per Sec.	26900,9	30621,4	31332,5	32392,2
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
Static Web Page Serving	Request per Sec.	15926,6	24980,4	26175,3	
i5-7267U					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Static Web Page Serving	Request per Sec.	21810,9	25419,9	26010,0	27580,3

Část testování		Virtualizovaná		
Název konfigurace		auto	nosmt	off
Static Web Page Serving	Request per Sec.	14019,4	10822,5	25299,7

Tabulka č. 11 – Naměřené hodnoty testu Apache Benchmark [autor]

NGINX Benchmark

Jedná se o testovací nástroj, který vykonává benchmarky pomocí nástroje Apache Benchmark. Tento test je prováděn oproti serveru Nginx, který měří počet požadavků za sekundu. Na rozdíl oproti serveru Apache test provádí měření s 2000000 požadavků, přičemž 500 požadavků provádí současně [Requests per second] (vyšší naměřené hodnoty jsou lepší).

i7-740Q					
Část testování		Fyzická			
Název konfigurace		auto	Spe=on	Mel=on	off
Static Web Page Serving	Request per Sec.	9232,7	10561,6	9651,0	10856,2
Část testování		Virtualizovaná			
Název konfigurace		auto	nosmt	off	
Static Web Page Serving	Request per Sec.	6894,4	7059,9	9373,1	
i7-4790K					
Část testování		Fyzická			
Název konfigurace		auto	Spe=on	Mel=on	off
Static Web Page Serving	Request per Sec.	35918,4	41305,0	41884,4	45052,7
Část testování		Virtualizovaná			
Název konfigurace		auto	nosmt	off	
Static Web Page Serving	Request per Sec.	22370,2	23108,3	37410,8	
i5-7267U					
Část testování		Fyzická			
Název konfigurace		auto	Spe=on	Mel=on	off
Static Web Page Serving	Request per Sec.	28001,8	34410,3	35089,2	38239,8
Část testování		Virtualizovaná			
Název konfigurace		auto	nosmt	off	
Static Web Page Serving	Request per Sec.	16899,3	17975,3	29633,5	

Tabulka č. 12 – Naměřené hodnoty testu NGINX Benchmark [autor]

PostgreSQL pgbench

PostgreSQL provádí jednoduchá testování databázového systému. Spouští za sebou stejnou sekvenci příkazů SQL, pokud možno ve více souběžných relacích databáze a vypočítává průměrnou rychlost transakcí. Základní testování obsahuje příkazy jako SELECT, UPDATE nebo INSERT prováděné v transakcích.

Testy budou provedeny v režimu měření Scaling Buffer Test (vyšší naměřené hodnoty jsou lepší):

- Test: Normal Load – Mode: Read Only [TPS]
- Test: Normal Load – Mode: Read Write [TPS]
- Test: Single Thread – Mode: Read Only [TPS]
- Test: Single Thread – Mode: Read Write [TPS]
- Test: Heavy Contention – Mode: Read Only [TPS]
- Test: Heavy Contention – Mode: Read Write [TPS]

i7-740Q					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Normal Load Read Only	TPS	32082,7	34991,9	32465,5	35260,1
Normal Load Read Write	TPS	3713,8	3909,2	3748,0	3920,0
Single Thread Read Only	TPS	3101,6	3167,3	3113,2	3229,0
Single Thread Read Write	TPS	334,6	346,1	338,8	351,8
Heavy Contention Read Only	TPS	36341,0	38647,1	36342,8	38874,3
Heavy Contention Read Write	TPS	4093,5	4260,8	4098,6	4257,0
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
Normal Load Read Only	TPS	23663,6	11598,1	26901,0	
Normal Load Read Write	TPS	2178,5	1047,1	2415,2	
Single Thread Read Only	TPS	6406,1	3851,1	7478,7	
Single Thread Read Write	TPS	406,4	309,5	425,5	
Heavy Contention Read Only	TPS	24785,3	11774,4	29430,9	
Heavy Contention Read Write	TPS	2071,7	1095,6	2272,6	
i7-4790K					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Normal Load Read Only	TPS	101620,5	110410,9	109087,7	111888,0

Normal Load Read Write	TPS	7383,1	7896,1	7499,9	7213,3
Single Thread Read Only	TPS	23476,5	24516,2	24428,9	25105,1
Single Thread Read Write	TPS	1018,7	1074,1	975,6	1042,1
Heavy Contention Read Only	TPS	104135,3	111517,6	109885,2	112182,5
Heavy Contention Read Write	TPS	10451,6	10626,4	10347,3	10557,8
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
Normal Load Read Only	TPS	70487,2	50039,6	94868,6	
Normal Load Read Write	TPS	7048,1	5565,8	8555,0	
Single Thread Read Only	TPS	15297,8	13093,1	20149,4	
Single Thread Read Write	TPS	855,8	820,9	971,2	
Heavy Contention Read Only	TPS	79484,3	56546,5	103005,6	
Heavy Contention Read Write	TPS	7792,7	5785,6	9514,6	
i5-7267U					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Normal Load Read Only	TPS	45482,7	51168,2	50481,9	52443,6
Normal Load Read Write	TPS	3827,9	4154,8	4104,6	4126,8
Single Thread Read Only	TPS	19410,7	21239,1	20738,9	21522,1
Single Thread Read Write	TPS	551,8	514,6	503,0	515,3
Heavy Contention Read Only	TPS	45757,7	52271,1	51615,5	53805,6
Heavy Contention Read Write	TPS	5638,8	6303,3	6142,7	6347,1
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
Normal Load Read Only	TPS	35463,4	28731,9	51087,6	
Normal Load Read Write	TPS	3327,5	1789,1	2731,1	
Single Thread Read Only	TPS	12986,6	12005,5	18057,0	
Single Thread Read Write	TPS	659,1	662,3	761,6	
Heavy Contention Read Only	TPS	39332,3	28259,9	54429,5	
Heavy Contention Read Write	TPS	3328,6	2085,2	3301,2	

Tabulka č. 13 – Naměřené hodnoty testu PostgreSQL pgbench [autor]

Redis

Tento test využívá databáze Redis pro simulaci spouštění příkazů N klienty najednou a zároveň odesíláním M dotazů jako odpověď.

Testování proběhne s následující systémovou konfigurací (vyšší naměřené hodnoty jsou lepší):

- LPOP [Request per second]
- SADD [Request per second]
- GET [Request per second]

i7-740Q					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
LPOP	Request per Sec.	278830,3	445860,2	285556,7	458552,3
SADD	Request per Sec.	259243,3	413045,7	264932,2	427595,3
GET	Request per Sec.	275862,6	440673,5	286187,9	457536,9
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
LPOP	Request per Sec.	259218,2	252554,9	822441,5	
SADD	Request per Sec.	237914,4	231765,7	724594,1	
GET	Request per Sec.	251856,7	245231,2	829599,4	
i7-4790K					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
LPOP	Request per Sec.	2884744,7	3009213,8	2991052,7	3089998,8
SADD	Request per Sec.	2323917,0	2368300,6	2449002,5	2373687,5
GET	Request per Sec.	2773364,2	2912790,7	2933553,54	3039866,8
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
LPOP	Request per Sec.	2427891,0	2334802,5	3071414,2	
SADD	Request per Sec.	1937197,0	1949352,2	2339997,3	
GET	Request per Sec.	2287161,5	2324414,0	2997653,3	
i5-7267U					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
LPOP	Request per Sec.	2330004,5	2557247,5	2529138,1	2547234,2
SADD	Request per Sec.	1947302,4	1982766,1	2153654,0	2108566,6
GET	Request per Sec.	2344529,6	2463414,2	2395753,2	2729813,1
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
LPOP	Request per Sec.	1979765,5	1986346,8	2511210,9	
SADD	Request per Sec.	1634004,9	1706646,0	2046975,3	
GET	Request per Sec.	1914568,4	1959912,5	2479376,0	

Tabulka č. 14 – Naměřené hodnoty testu Redis [autor]

Stress-NG

Stress-NG testuje systém pomocí jeho vytěžování (stress-test) různými způsoby Autor se zaměřil především na ty, které souvisí s testováním procesoru.

Vytěžování systému bude provedeno těmito testy (vyšší naměřené hodnoty jsou lepší):

- Crypto [Bogo Ops/s]
- CPU Stress [Bogo Ops/s]
- Socket Activity [Bogo Ops/s]
- Context Switching [Bogo Ops/s]

i7-740Q					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Crypto	Bogo Ops/s	270,9	271,4	270,9	270,7
CPU Stress	Bogo Ops/s	584,6	595,8	592,5	589,5
Socket Activity	Bogo Ops/s	972,7	1163,1	1019,1	1209,5
Context Switching	Bogo Ops/s	484911,0	536802,5	526815,7	563184,0
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
Crypto	Bogo Ops/s	263,4	167,1	263,5	
CPU Stress	Bogo Ops/s	640,3	315,2	640,3	
Socket Activity	Bogo Ops/s	600,6	328,3	1002,5	
Context Switching	Bogo Ops/s	266643,2	150663,7	366048,2	
i7-4790K					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Crypto	Bogo Ops/s	954,3	955,5	954,8	955,5
CPU Stress	Bogo Ops/s	1966,1	1950,6	1976,5	1937,6
Socket Activity	Bogo Ops/s	3572,6	4625,6	4530,7	4805,1
Context Switching	Bogo Ops/s	1450457,5	2844841,5	3165435,2	2921220,8
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
Crypto	Bogo Ops/s	941,5	911,2	944,2	
CPU Stress	Bogo Ops/s	1956,3	1333,8	1964,1	
Socket Activity	Bogo Ops/s	2085,7	1657,6	3763,1	
Context Switching	Bogo Ops/s	918168,5	747197,4	1582264,8	

i5-7267U					
Část testování		**Fyzická**			
Název konfigurace		**auto**	**Spe=on**	**Mel=on**	**off**
Crypto	Bogo Ops/s	384,1	401,2	397,4	399,1
CPU Stress	Bogo Ops/s	926,1	908,7	905,6	910,0
Socket Activity	Bogo Ops/s	1282,4	1800,8	1772,5	1998,5
Context Switching	Bogo Ops/s	599856,0	781542,1	765261,0	814060,2
Část testování		**Virtualizovaná**			
Název konfigurace		**auto**	**nosmt**	**off**	
Crypto	Bogo Ops/s	400,9	390,0	401,1	
CPU Stress	Bogo Ops/s	922,2	635,1	918,9	
Socket Activity	Bogo Ops/s	804,4	621,0	1593,4	
Context Switching	Bogo Ops/s	406151,4	313491,0	755299,4	

Tabulka č. 15 – Naměřené hodnoty testu Stress-NG [autor]

4.5 Jednoduchá analýza rozptylu s mnohonásobným porovnáváním

Pomocí jednoduché analýzy rozptylu bude otestována hypotéza o shodě průměrů výběrových souborů jednotlivých režimů. Pro tato statistická testování autor použil data, z již naměřených testů. Jelikož byly v předcházející kapitole testy rozčleněny do tří skupin (vstupní a výstupní výkon, výkon procesoru a výkon systému) bude z každé skupiny vybrána alespoň jedna z testovacích konfigurací, na které bude analýza provedena. Celkem bude provedeno osm statistických testování. Testován bude fyzický i virtuální režim v téže konfiguraci. Bude tedy zkoumán vliv jediného faktoru na vybraný statistický znak tzv. analýza rozptylu při jednoduchém třídění.

$$H_0: \mu_1 = \mu_2 = \mu_3 = \mu_4 \tag{4.1}$$

Testovaní jednoduché analýzy rozptylu bude provedeno u těchto testovacích konfigurací:

- Vstupní a výstupní výkon
  - Flexible IO Tester – Random Read [MB/s]
- Výkon procesoru
  - Hackbench – 16 Thread [sekundy]
- Výkon systému
  - Stress-NG – CPU Stress [Bogo Ops/s]
  - Stress-NG – Context Switching [Bogo Ops/s]

Statistické testování bude provedeno ve dvou etapách. V první části je nutno otestovat globálně nulovou hypotézu (4.1) o shodě průměrů. Pokud při analýze rozptylu nedojde k zamítnutí nulové hypotézy, jedná se o shodu průměrů a není již nutno dále provádět další testování. Pokud nulová hypotéza bude zamítnuta, bude testování pokračovat do své druhé části, kde dojde k mnohonásobnému porovnávání. Teprve poté bude možné zhodnotit, které z testovaných výběrových souborů se od sebe statisticky významně liší. U každého jednotlivého testování bude uveden pokusný plán. Z pokusných plánů z řádku rozsah lze vyčíst, zda se bude jednat o ortogonální či neortogonální plán. Všechny výpočty a grafické výstupy budou provedeny pomocí statistického softwarového programu SAS 9.4.

```
data stat;
input rezim $ contextSwitching @@;
datalines;
7267Uauto 602788.38 7267Uspe 785757.01 7267Umel 775098.02 7267Uoff 826378.34
7267Uauto 599322.86 7267Uspe 780091.66 7267Umel 754540.97 7267Uoff 806351.26
7267Uauto 597457.04 7267Uspe 778777.66 7267Umel 766144.20 7267Uoff 809451.22
;

proc boxplot data=stat;
plot contextSwitching*rezim;
run;

proc glm data=stat;
class rezim;
model contextSwitching=rezim;
means rezim/hovtest scheffe;
means rezim/duncan waller;
run;
```

Obrázek č. 6 – Ukázka příkazů pro jednoduchou analýzu rozptylu v programu SAS [autor]

V první části příkazů pro SAS (Obrázek č. 6) jsou definována vstupní data. Proměnná `rezim` definuje možnost vlastního označení, `contextSwitching` pak zastupuje příslušnou naměřenou hodnotu. Všechna data musí splňovat podmínku normálního rozdělení hodnot ve výběrovém souboru. V tomto případě lze využít testu normality Shapiro-Wilkova testu a jeho p-hodnoty. Pokud je p-hodnota větší než definovaná hladina významnosti α = 0,05 lze považovat hodnoty v souboru za normálně rozdělené. V druhé části příkazů se nachází procedura `boxplot` pomocí níž je možné prvotního posouzení o shodě průměrů pomocí grafického znázornění. Poslední část obsahuje příkazy pro samotný výpočet jednoduché analýzy rozptylu a mnohonásobného porovnávání procedurou `glm`. Argumentem `hovtest` v příkazu `means` se

prostřednictvím Leveneova testu ověří předpoklad použitelnosti analýzy rozptylu. Testuje se shoda rozptylů porovnávaných souborů. Následující argumenty `scheffe`, `duncan`, `waller` slouží k podání výstupu o mnohonásobném porovnávání průměrů pomocí Scheffeho metody (S-metoda) nebo Duncan-Wallerovi metody. Přednostně bude využit výstup S-metody. Pouze v případech, kdy výstup S-metody podává pouze párové porovnávání, namísto označování skupin průměrů bude použita Duncan-Wallerova metoda, která párové porovnávání S-metody transformuje do skupin průměrů, tak aby byla zachována jednotná koncepce komparace u všech osmi statistických testování.

4.5.1 Statistická testování

U každého provedeného testování je uveden pokusný plán s přehledem hodnot, `glm` procedura analýzy rozptylu s mnohonásobným porovnáváním pomocí Scheffeho nebo Duncan-Wallerovi metody. Shodně jako v předcházejících kapitolách žlutě podbarvená pole platí pro virtualizovanou část testování. Pro fyzické části testování byla stanovena shodná nulová hypotéza (4.1). Pro virtualizované části testování byla stanovena hypotéza (4.2), která je shodná ve všech případech testování.

$$H_0: \mu_1 = \mu_2 = \mu_3 \tag{4.2}$$

Všechny výstupy procedur jsou ilustrativně uvedeny pouze u první testované konfigurace Flexible IO Tester. U ostatních konfigurací budou pro přehlednost uvedeny zkrácené tabulky s p-hodnotami, podle kterých se určuje potvrzení či zamítnutí nulové hypotézy. Ostatní pokusné plány, boxploty, a kompletní přehled naměřených hodnot je připojen v příloze A (8.1), příloze B (8.2) a příloze C (8.3)

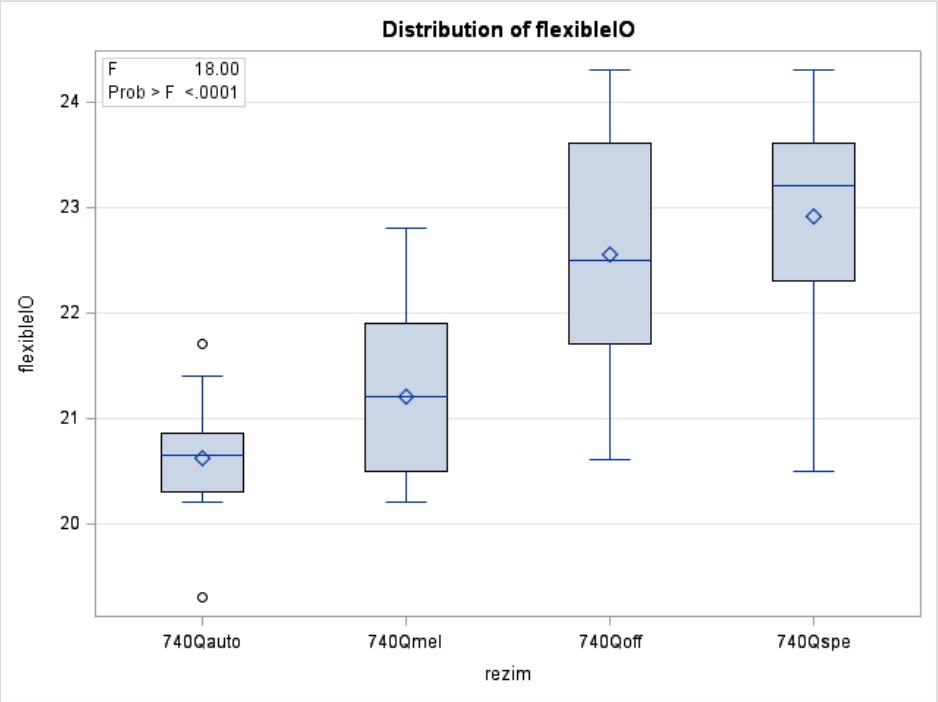
Flexible IO Tester

Aby bylo možné zhodnotit, zda u naměřených hodnot rychlosti čtení z disku v konfiguraci Random Read v různých režimech procesorů skutečně došlo k poklesu výkonu anebo se jedná jen o odchylku způsobenou náhodným vlivem či chybou měření je nutné provést analýzu rozptylu. Tam, kde bude prokázán statisticky významný rozdíl mezi soubory dojde prokazatelně k úbytku výkonu. Poté je dobré zhodnotit, mezi kterými soubory došlo k významnému poklesu.

Konf.	Random read – Fyzická část											
CPU	i7-740Q				i7-4790K				i5-7267U			
Třída	auto	Spe=on	Mel=on	off	auto	Spe=on	Mel=on	off	auto	Spe=on	Mel=on	off
Pozorované hodnoty	20,8	23,5	22,8	22,2	41,9	32,6	31,8	42,2	16,4	15,2	16,4	13,8
	19,3	21,3	22,8	23,6	40,0	31,2	32,7	41,1	18,5	15,3	15,7	14,1
	21,7	22,9	20,3	24,3	41,7	31,0	31,7	41,8	18,4	14,9	16,6	14,7
	20,8	22,3	20,5	23,9					18,3			15,3
	20,3	20,5	20,4	20,6					18,1			15,9
	21,4	23,6	21,2	24,2					18,1			15,4
	20,4	22,8	20,5	22,8					18,4			15,2
	20,3	23,2	20,2	20,9					17,9			15,5
	20,2	24,2	20,7	22,2					18,0			15,7
	20,7	23,4	22,2	21,6					18,1			14,1
	20,6	24,3	21,5	22,5					17,0			14,6
	20,9	23,7	20,5	22,2					17,8			13,9
		23,3	21,2	22,7					18,3			15,2
		22,6	21,4	21,7					18,4			14,5
		22,0	21,9	22,9								15,1
Rozsah	12	15	15	15	3	3	3	3	14	3	3	15

Tabulka č. 16 – Souhrnný pokusný plán pro Flexible IO Tester fyzické části [autor]

Z řádku Rozsah, v tabulce č. 16, lze vyčíst neortogonální pokusný plán.



Obrázek č. 7 – Ukázka výstupu procedury Boxplot u procesoru i7-740Q [autor]

V tabulce č. 17, řádek Rozsah informuje o neortogonálním pokusném plánu.

Na obrázku číslo 7 je zobrazen výstup procedury `boxplot`. Jedná se o diagram znázorňující pětičíselný souhrn. Horní a dolní kvartil znázorňují hrany krabice. Prostřední úsečka je rovna mediánu. Vybíhající úsečky z krabic znázorňují další hodnoty. V tomto případě se jedná o asymetrické rozložení dat u všech krabic. Hodnoty, které jsou označeny kroužky mimo vybrané vybíhající úsečky považujeme za odlehlá pozorování.

Konf.	**Random read – Virtualizovaná část**								
CPU	**i7-740Q**			**i7-4790K**			**i5-7267U**		
Třída	**auto**	**nosmt**	**off**	**auto**	**nosmt**	**off**	**auto**	**nosmt**	**off**
Pozorované hodnoty	6,651	7,883	7,061	29,900	25,500	26,000	16,500	14,900	15,400
	7,219	7,224	7,470	30,200	25,600	27,000	16,600	15,100	15,300
	7,305	7,440	7,430	29,900	25,600	27,300	16,000	14,900	15,600
	7,420	6,965	7,373						
	7,329	6,912							
	6,788	7,545							
	7,100	6,811							
	7,402	7,068							
	6,936	6,944							
	7,029	6,941							
	7,223	6,996							
	7,119	6,950							
	7,148	6,998							
	7,089	7,347							
	6,927	7,131							
Rozsah	15	15	4	3	3	3	3	3	3

Tabulka č. 17 – Souhrnný pokusný plán pro Flexible IO Tester virtualizované části [autor]

Dále jsou v tabulkách zobrazeny výstupy jednotlivých statistických testování. V tabulce č. 18 jsou červenou barvou zvýrazněny p-hodnoty příslušných testů, které vypovídají o potvrzení či zamítnutí nulové hypotézy. Tabulka č. 19 a 20 uvádí výstup mnohonásobných porovnávání.

i7-740Q						i7-4790K				
Flexible IO Tester - Random Read - Fyzické prostředí										
GLM procedura										
	DF	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Model	3	48,5631	16,1877	18,00	<.0001	3	278,1425	92,7141	150,55	<.0001
Error	53	47,6726	0,8994			8	4,9266	0,6158		
Corrected Total	56	96,2357				11	283,0691			
Leveneův test homogenity										
	DF	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
rezim	3	5,1940	1,7313	1,31	0,2796	3	0,5826	0,1941	1,19	0,3725
Error	53	69,8394	1,3177			8	1,3025	0,1628		
i5-7267U						**i7-740Q**				
Fyzické prostředí						**Virtualizované prostředí**				
GLM procedura										
	DF	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Model	3	74,2771	24,7590	66,91	<.0001	2	0,1564	0,0782	1,22	0,3094
Error	31	11,4702	0,3700			31	1,9893	0,0641		
Corrected Total	34	85,7474				33	2,1457			
Leveneův test homogenity										
	DF	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
rezim	3	0,5095	0,1698	0,66	0,5822	2	0,0141	0,0070	0,71	0,5014
Error	31	7,9628	0,2569			31	0,3098	0,0099		
i7-4790K						**i5-7267U**				
Flexible IO Tester - Random Read - Virtualizované prostředí										
GLM procedura										
	DF	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Model	2	31,5488	15,7744	95,28	<.0001	2	3,0488	1,5244	32,67	0,0006
Error	6	0,9933	0,1655			6	0,2800	0,0466		
Corrected Total	8	32,5422				8	3,3288			
Leveneův test homogenity										
	DF	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
rezim	2	0,1778	0,0889	3,71	0,0893	2	0,0065	0,0032	2,56	0,1570
Error	6	0,1437	0,0240			6	0,0076	0,0013		

Tabulka č. 18 – Výstupy procedury GLM a Leveneova testu pro Flexible OI Tester [autor]

Z provedené procedury GLM lze vyčíst p-hodnotu ze sloupce Pr > F. Tam, kde hodnota překračuje hodnotu zvolené hladiny α = 0,05 došlo k potvrzení nulové hypotézy. V tomto případě tak došlo pouze u testování i7-740Q virtualizované části, kde je p-hodnota 0,3094. Zde je nulová hypotéza potvrzena a platí shoda průměrů výběrových souborů. V ostatních případech je nutno nulovou hypotézu zamítnout a konstatovat že se alespoň v jednom případě režimy od sebe významně odlišily. Tyto režimy jsou mnohonásobně porovnávány v tabulkách č. 19 a 20. P-hodnota Leveneova testu informuje o posouzení shody rozptylů v těch případech, kde hodnota překročila hladinu α = 0,05.

i7-740Q				i7-4790K				i5-7267U			
Duncan	**Mean**	**N**	**Režim**	**Scheffe**	**Mean**	**N**	**Režim**	**Duncan**	**Mean**	**N**	**Režim**
A	22,9067	15	Spe=on	A	41,7000	3	off	A	17,9786	14	auto
A	22,5533	15	off	A	41,2000	3	auto	B	16,2333	3	Mel=on
B	21,2067	15	Mel=on	B	32,0667	3	Mel=on	C	15,1333	3	Spe=on
B	20,6167	12	auto	B	31,6000	3	Spe=on	C	14,8667	15	off

Tabulka č. 19 – Výstup mnohonásobného porovnávání pro Flexible IO Tester fyzické části [autor]

Výsledek mnohonásobného porovnávání souborů (tabulka č. 19 a 20) mezi sebou je prezentován pomocí skupin průměrů. Stejným písmenem jsou označeny vždy ty skupiny průměrů, jejichž členy se od sebe statisticky významně neliší.

Hodnoty z tabulky číslo 19 lze pro procesor i7-740Q ve fyzickém prostředí zhodnotit takto. Významně se od sebe neliší režimy Spe=on a off a režimy Mel=on a auto. Dále se prokázalo, že rozdíly mezi režimy Spe=on a Mel=on a režimy Spe=on a auto a režimy off a Mel=on a režimy off a auto jsou statisticky významné.

Testování pro procesor i7-7490K ve fyzickém prostředí udává statisticky nevýznamný rozdíl mezi režimy off a auto a mezi režimy Mel=on a Spe=on. Statisticky významný rozdíl mezi soubory nastal mezi režimy off a Mel=on a mezi režimy off a Spec=on a mezi režimy auto a Mel=on a mezi režimy auto a Spe=on.

U posledního procesoru i5-7267U u testování ve fyzickém prostředí lze zhodnotit, že režimy Spe=on a off se od sebe statisticky neliší. Režim auto se od všech ostatních režimů významně odlišil. Dále se významně mezi sebou liší režimy Mel=on a Spe=on a režimy Mel=on a off.

i7-4790K				i5-7267U			
Scheffe	**Mean**	**N**	**Režim**	**Scheffe**	**Mean**	**N**	**Režim**
A	30,0000	3	auto	A	16,3667	3	auto
B	26,7667	3	off	B	15,4333	3	off
C	25,5667	3	nosmt	B	14,9667	3	nosmt

Tabulka č. 20 – Výstup mnohonásobného porovnávání pro Flexible IO Tester virtualizované části [autor]

Ve výsledcích testování mnohonásobného porovnávání pro virtualizovanou část se nenachází výsledek pro i7-740Q, protože u něj byla potvrzena nulová hypotéza (4.1).

Všechny režimy procesoru i7-4790K ve virtualizovaném prostředí se svými výsledky od sebe statisticky významně liší.

Statisticky významný rozdíl nebyl naměřen mezi režimy off a nosmt u procesoru i5-7267U a zároveň se oba statisticky významně odlišují od režimu auto.

Hackbench

Zátěžový test jádra linuxového plánovače v režimu konfigurace 16 Thread [sekundy] proběhl z důvodu zhodnocení, zda se mezi sebou liší naměřené soubory hodnot a lze konstatovat, že prokazatelně došlo ke ztrátě výkonu procesoru. Testovací plán, boxplot a kompletní přehled výstupů testovacích procedur lze nalézt v příloze A (8.1). Ve zjednodušených tabulkách jsou uvedeny p-hodnoty, podle kterých je o výsledcích tesů rozhodováno. Zkratka GLM zastupuje p-hodnotu procedury `glm`, zkratka Levene zastupuje p-hodnotu Leveneova testu homogenity. Žlutě podbarvená pole jsou výsledky pro testování ve virtualizovaném prostředí.

i7-740Q				i7-4790K				i5-7267U			
GLM	**Levene**	**GLM**	**Levene**	**GLM**	**Levene**	**GLM**	**Levene**	**GLM**	**Levene**	**GLM**	**Levene**
<.0001	0,4414	<.0001	0,4092	<.0001	0,0039	<.0001	0,0507	<.0001	0,4886	<.0001	0,1599

Tabulka č. 21 – Zkrácený výstup procedury GLM pro testování analýzy rozptylu u testu Hackbench [autor]

U všech konfigurací procesorů byla naměřena p-hodnota procedury `glm` menší než referenční hladina $\alpha = 0{,}05$. Lze tedy konstatovat, že ve všech případech se průměry souborů alespoň v jednom případě od sebe významně liší (zamítáme nulové hypotézy (4.1), (4.2)) a je možno provést podrobnější zhodnocení pomocí mnohonásobného

porovnávání. Leveneův test indikuje shodu rozptylů jednotlivých souborů v případě fyzické části testování u procesoru i7-4790K byla naměřena p-hodnota = 0,0039. Zde nebyla potvrzena shoda rozptylů jednotlivých souborů a je nutno stabilizovat rozptyl pomocí transformace proměnných. Pokud se rozptyl stabilizovat nepodaří je možno provést neparametrickou analýzu rozptylu pomocí neparametrického Kruskal-Wallisova testu. P-hodnota Kruskal-Wallisova testu Pr > Chi-Square = <.0001.

i7-740Q				i5-7267U			
Duncan	**Mean**	**N**	**Režim**	**Duncan**	**Mean**	**N**	**Režim**
A	342,315	9	Mel=on	A	271,230	3	auto
A	330,352	9	auto	B	188,693	12	Mel=on
B	250,830	3	Spe=on	B	187,520	3	off
B	250,294	3	off	C	168,890	3	Spe=on

Tabulka č. 22 – Hackbench mnohonásobné porovnání fyzické části [autor]

Fyzická část statistického testování lze zhodnotit statistickou významností mezi režimy Mel=on – Spe=on, Mel=on – off, auto – Spe=on, auto – off u procesoru i7-740Q a statisticky nevýznamným rozdílem mezi Mel=on – auto a Spe=on – off.

Výsledek testování pro procesor i7-4790K uvádí statisticky významný rozdíl mezi režimy Spe=on – off. Zbylé režimy, podle přílohy A (8.1) obrázek č. 24, nepřekročili referenční hranici pro porovnávání a není tedy mezi nimi statisticky významný rozdíl.

Procesor i5-7267U má odlišné statisticky významné režimy auto, Spe=on, které se významně liší od sebe i ostatních. Statisticky se od sebe významně neliší pouze režimy Mel=on – off.

i7-740Q				i7-4790K				i5-7267U			
Duncan	**Mean**	**N**	**Režim**	**Duncan**	**Mean**	**N**	**Režim**	**Duncan**	**Mean**	**N**	**Režim**
A	952,95	6	nosmt	A	267,546	3	nosmt	A	631,19	3	nosmt
B	799,04	3	auto	B	228,873	3	auto	B	592,00	6	auto
C	656,30	9	off	C	94,930	15	off	C	210,97	3	off

Tabulka č. 23 -Hackbench mnohonásobné porovnání virtualizované části [autor]

Ve virtualizované části statistického testování konfigurace Hackbench došlo k významnému rozdílu mezi všemi výběrovými soubory u všech třech procesorů. Všechny se od sebe statistický významně liší.

Stress-NG

Statistické testování Stress-NG bylo provedeno ve dvou konfiguracích. Zvlášť pro konfiguraci Stress CPU [Bogo Ops/s] a Context Switching [Bogo Ops/s], které souvisí s různými variantami vytěžování procesoru. Podrobný přehled výstupů je uveden v příloze B (8.2) a příloze C (8.3). Žlutou barvou jsou opět podbarveny testy provedené ve virtualizovaném prostředí.

Stress CPU											
i7-740Q				i7-4790K				i5-7267U			
GLM	Levene	GLM	Levene	GLM	Levene	GLM	Levene	GLM	Levene	GLM	Levene
0,9177	0,3803	<.0001	0,2551	0,0507	0,299	<.0001	0,0883	0,0364	0,1397	<.0001	0,1075

Tabulka č. 24 – Stress CPU zkrácený výstup procedury GLM [autor]

Tabulka č. 24 uvádí, že fyzická část testování u procesorů i7-740Q a i7-4790K nenabyla statisticky významného rozdílu a proto v obou případech nezamítáme nulovou hypotézu (4.1) a lze tyto rozdíly mezi soubory považovat za statisticky nevýznamné. U všech ostatních prostředí se prokázal statisticky významný rozdíl.

i5-7267U			
Scheffe	Mean	N	Režim
A	926,133	3	auto
A	910,077	3	off
A	908,783	3	Spe=on
A	905,647	3	Mel=on

Tabulka č. 25 – Stress CPU mnohonásobné porovnávání fyzické části [autor]

Mnohonásobné porovnávání bylo provedeno ve fyzické části pouze u procesoru i5-7267U. Jelikož Scheffeho rozdělení přiřadilo všechny režimy do skupiny A, nastala chyba 1. řádu (nesprávné zamítnutí nulové hypotézy), tedy ani v tom to případě nedošlo k potvrzení významného rozdílu mezi výběrovými soubory a lze tedy říct, že rozdíl mezi nimi je statistický nevýznamný, a proto nakonec můžeme přijmout nulovou hypotézu (4.1) o shodě průměrů.

i7-740Q				i7-4790K				i5-7267U			
Scheffe	**Mean**	**N**	**Režim**	**Scheffe**	**Mean**	**N**	**Režim**	**Scheffe**	**Mean**	**N**	**Režim**
A	640,380	3	off	A	1964,157	3	off	A	922,243	3	auto
A	640,303	3	auto	A	1956,350	3	auto	A	918,973	3	off
B	315,270	3	nosmt	B	1333,873	3	nosmt	B	635,113	3	nosmt

Tabulka č. 26 – Stress CPU mnohonásobné porovnávání virtualizované části [autor]

Tabulka č. 26 je výstupem mnohonásobného porovnávání pro virtualizovaný režim. U všech procesorů nastalo stejné rozdělení. Jako jediný se ode všech statisticky odlišil režim nosmt. Ostatní režimy off a auto nejsou statisticky rozdílné.

Context Switching											
i7-740Q				i7-4790K				i5-7267U			
GLM	**Levene**	**GLM**	**Levene**	**GLM**	**Levene**	**GLM**	**Levene**	**GLM**	**Levene**	**GLM**	**Levene**
<.0001	0,2613	<.0001	0,4289	<.0001	0,0841	<.0001	0,2321	<.0001	0,1689	<.0001	0,3777

Tabulka č. 27 – Context Switching zkrácený výstup procedury GLM [autor]

Dle výstupu procedury GLM lze zhodnotit u všech procesorů i režimů, že nastal prokazatelný rozdíl alespoň v jednom případě od ostatních, a proto je nutno provést mnohonásobné porovnávání. Všechny p-hodnoty GLM jsou menší než referenční hladina $\alpha = 0{,}05$ a je tedy potřeba zamítnou nulovou hypotézu (4.1) a (4.2).

i7-740Q				i7-4790K				i5-7267U			
Duncan	**Mean**	**N**	**Režim**	**Duncan**	**Mean**	**N**	**Režim**	**Scheffe**	**Mean**	**N**	**Režim**
A	563184	3	off	A	3165435	15	Mel=on	A	814060	3	off
B	536803	3	Spe=on	A	2921221	15	off	B	781542	3	Spe=on
B	526816	15	Mel=on	A	2844842	15	Spe=on	B	765261	3	Mel=on
C	497650	6	auto	B	1450458	3	auto	C	599856	3	auto

Tabulka č. 28 – Context Switching mnohonásobné porovnávání fyzické části [autor]

Ve fyzické části statistického testování mnohonásobného porovnávání pro procesor i7-740Q a i5-7267U lze konstatovat, že režim off a auto se od ostatních významně odlišily. Statisticky nevýznamný rozdíl nastal mezi režimy Spe=on – Mel=on.

Pro procesor i7-4790K se statisticky významně odlišil režim auto, ostatní režimy se od sebe významně neliší.

i7-740Q				i7-4790K				i5-7267U			
Duncan	**Mean**	**N**	**Režim**	**Scheffe**	**Mean**	**N**	**Režim**	**Scheffe**	**Mean**	**N**	**Režim**
A	366048	6	off	A	1582265	3	off	A	755299	3	off
B	266643	3	auto	B	918169	3	auto	B	406151	3	auto
C	150664	3	nosmt	C	747197	3	nosmt	C	313491	3	nosmt

Tabulka č. 29 – Context Switching mnohonásobné porovnávání virtualizované části [autor]

Tabulka č. 29 uvádí statisticky významný rozdíl, který nastal mezi všemi režimy každého procesoru virtualizovaného prostředí. Lze tedy říct, že se všechny režimy od sebe statisticky významně liší.

5 Výsledky a diskuse

Aby bylo možné zhodnotit výsledky ztráty výkonu procesorů, vstupně výstupního výkonu a systému jako celku, zvlášť ve fyzickém a virtualizovaném prostředí, je nutno výsledky rozdělit do dvou částí.

5.1 Výsledky ztrát na výkon dle naměřených hodnot

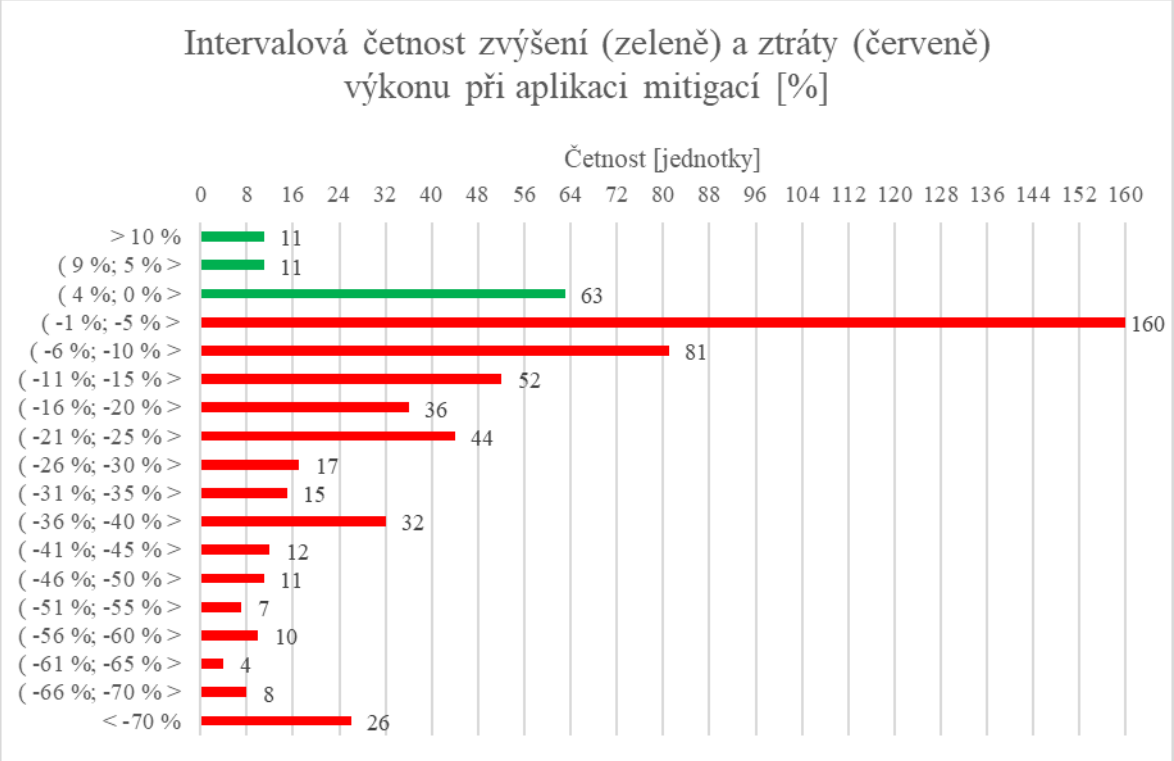
Po dokončení fáze, ve které došlo k naměření hodnot autor provedl výpočet ztráty a nárůstu výkonu u všech typů procesoru, provedených testů i konfigurací (příloha D (8.4)). Ztráta byla měřena ve všech konfiguracích: Auto – všechny mitigace zapnuty, Spe=on – zapnuty mitigace pro Spectre útok, Mel=on – zapnuty mitigace pro útok Meltdown, vůči výchozímu stavu Off – všechny mitigace vypnuty. Dále autor provedl rozdělení výsledných ztrát do intervalů.

	Počet výskytů v intervalu ve fyzickém prostředí				Počet výskytů v intervalu ve virtualizovaném prostředí		
Interval	**i7-740Q**	**i7-4790K**	**i5-7267U**	**Interval**	**i7-740Q**	**i7-4790K**	**i5-7267U**
> 0 %	14	27	32	**> 0 %**	3	2	7
(-1 %; -5 % >	46	47	36	**(-1 %; -5 % >**	10	10	11
(-6 %; -10 % >	21	22	25	**(-6 %; -10 % >**	3	6	4
(-11 %; -15 % >	14	5	10	**(-11 %; -15 % >**	10	5	8
(-16 %; -20 % >	8	4	5	**(-16 %; -20 % >**	6	11	2
(-21 %; -25 % >	9	8	2	**(-21 %; -25 % >**	10	9	6
(-26 %; -30 % >	0	1	3	**(-26 %; -30 % >**	6	5	2
(-31 %; -35 % >	1	0	3	**(-31 %; -40 % >**	9	12	14
< -35 %	7	6	4	**(-41 %; -60 % >**	13	9	13
				(-61 %; -70 % >	7	2	1
				< -70 %	3	9	12

Tabulka č. 30 – Četnost výskytu procentní ztráty výkonu u různých procesorů všech testů [autor]

Autor dospěl ke zjištění, že velikost ztráty výkonu souvisí se stářím procesorů, jejich vylepšenou architekturou v průběhu let, ale také na prováděných úkonech. Procesor i7-740Q je nejstarším měřeným procesorem, který vykázal ze všech testů ve fyzické části testování nejčetnější ztrátu 1 – 5 %. Nepatrný nárůst výkonu byl zaznamenán ve čtrnácti případech. Největší ztráta výkonu se pohybovala nad 35 %. Díky zvyšování výkonu v průběhu let a nových implementovaných mikro technologií do čipů se u dalších měřených procesorů četnost největší ztráty výkonu postupně snižovala. U procesoru

i5-7267U nejnovějšího měřeného byl zaznamenán mírný nárůst výkonu až v třiceti dvou případech a ustálení ztráty výkonu mezi 1 – 15%. Virtualizované prostředí je z hlediska hardwarových prostředků více náročné na provoz. V případě i7-740Q bylo nejčastěji dosaženo ztráty ve virtualizovaném režimu mezi 11 – 60%. S četností 3 byla pozorována ztráta výkonu o více než 70 %. U nejnovějšího procesoru i5-7267U byla ztráta naměřena nejčastěji mezi 1 – 40%. Jedná se o snížení minimální hranice intervalu směrem k nule a také 20% rozdílu u hranice maximální. To ovšem nevylučuje, že ztráta nemůže být vyšší než 70 %. U tohoto intervalu byla naměřena četnost 12 což je o 9 více než u procesoru i7-740Q. Je to dáno optimalizací navrženou přímo pro lepší práci s mitigacemi, která je implementována přímo v procesoru. Virtualizovaný režim dosahuje tím větší ztráty u všech procesorů, když je chráněn i proti útoku typu Foreshadow, díky němuž musí být vypnut i hyper-threading a tím je rovnou dána minimálně poloviční ztráta výkonu procesoru.



Obrázek č. 8 – Rozdělení četností ztrát výkonu u všech měřených testů, režimů a procesorů [autor]

Pokud bychom rozdělili do intervalů všechny procentní ztráty výkonu u všech provedených testů, procesorů i režimů (Obrázek č. 17), je patrné, že nejčastěji došlo ke ztrátě mezi 1 – 5%. Velkou četnost můžeme pozorovat i u intervalů až do ztráty 40% výkonu. Ztráta nad 70 % byla naměřena ve 26 případech, a to nejčastěji z důvodu vypnutí

simultánního hyper-threadingu a aplikování všech možných mitigací procesorů. Dále lze pozorovat nárůst výkonu ve více jak 63 případech až o více než 10 %.

Dále autor provedl výpočet četností, k jak velkému celkovému poklesu výkonu došlo u jednotlivých typů měření. Měření bylo rozděleno do třech částí: Vstupně výstupní výkon, Výkon procesoru, Výkon systému. V každé části byla provedena série testů.

Výkon	Nejčastější výskyt intervalu ztráty výkonu					
	Fyzické prostředí			Virtualizované prostředí		
	i7-740Q	i7-4790K	i5-7267U	i7-740Q	i7-4790K	i5-7267U
Vstupně výstupní	6 – 10 %	1 – 10 %	1 – 10 %	11 – 40 %	6 – 20 %	1 – 10 %
Procesoru	1 – 5 %	1 – 10 %	<0 – 5 %	46 – >70 %	36 – >70 %	21 – >70 %
Systému	1 – 20 %	1 – 10 %	1 – 15 %	56 – 70 %	16 – 40 %	11 – 35 %

Tabulka č. 31 –Nejčastější výskyt intervalu ztráty výkonu u různých typů měření [autor]

Autor došel ke zjištění, že ve fyzické části testování došlo k poklesu výkonu až 10 %, ve virtualizované části až ke 40%. Ve fyzické části měření výkonu procesoru došlo k poklesu až 10 % a u procesoru i5-7267U dokonce i k mírnému nárůstu výkonu díky optimalizaci vnitřní struktury. Ve virtualizované části došlo u všech procesorů k poklesu až nad 70 %. U systémových testů se ztráta pohybovala okolo 20% ve fyzické a okolo 50% ve virtualizovaném prostředí. Je možné dobře sledovat, že u i7-740Q je nárůst ztráty při aplikaci mitigací větší než u nového typu procesoru i5-7267U. Totéž platí i ve virtualizovaném prostředí, pouze s výjimkou procesorových testů, kde se pouze snížila minimální hranice ztráty výkonu z 46 – 70% na 21 – 70% procent.

Vstupně výstupní výkon zaznamenal nejvyšší nárůst ztráty výkonu u testů Disk Transaction Performance a Random Write a Read Compiled Tree. Nejméně se ztráta dotkla Sequential Read testu.

Ztráta na výkon procesoru bude tím nejznatelnější čím více bude spuštěno procesů najednou a také zpracováváno více vláken. Tím více když bude stroj pracovat ve virtualizovaném režimu. Až 30% ztrátu výkonu byla zaznamenána u operací ffsll a pthread_once a Time to Compile. Nepatrná změna byla zaznamenána u renderování 3D grafiky pomocí procesoru a při kryptografických operacích.

Mitigace nejvíce ovlivnili systémový výkon u testů, které simulovali operace databázových serverů LPOP, SADD, GET. Dalšími testy, které zaznamenali závažnou ztrátu výkonu byly simulované operace webových serverů a počet zpracovaných requestů za sekundu. Posledním výrazným zpomalením bylo přepínání kontextu procesoru u testu Context Switching. Nejmenší změnu zaznamenalo databázové testování PostgreSQL při čtení a zápisu v různých režimech.

5.2 Výsledky jednoduché analýzy rozptylu

U čtyřech různých testů autor provedl analýzu rozptylu s vícenásobným porovnáváním. Dílčím cílem bylo zjistit, zda provedená měření v různých režimech jsou opravdu rozdílná a statisticky významná. Testuje se, zda nedošlo k chybě měření, která mohla být ovlivněna náhodným vlivem.

Výsledky analýzy rozptylu - stat. významná změna při poklesu výkonu						
Procesor	**i7-740Q**		**i7-4790K**		**i5-7267U**	
Režim	**Fyz.**	**Virt.**	**Fyz.**	**Virt.**	**Fyz.**	**Virt.**
Random Read	Ano	Ne	Ne	Ano	Ano	Ano
16 Thread	Ano	Ano	Ne	Ano	Ano	Ano
Stress CPU	Ne	Ne	Ne	Ne	Ne	Ne
Context Switching	Ano	Ano	Ano	Ano	Ano	Ano

Tabulka č. 32 – Výsledky analýzy rozptylu – statisticky významná změna při poklesu výkonu [autor]

Zda se opravdu jedná o statisticky významný rozdíl mezi daty pro režim bez mitigací a s jimi aplikovaným. Naměřené soubory jednotlivých procesorů, byly testovány ve čtyřech testech, kde alespoň vždy jeden pochází z každé skupiny testování vstupně výstupního výkonu, procesorového výkonu a systémového výkonu. Tabulka č. 32 zobrazuje výsledky prokázání statisticky významného rozdílu mezi naměřenými režimy. Zelenou barvou jsou podbarveny ty hodnoty, kde došlo k významnému rozdílu mezi režimem bez mitigací a s aplikovanými mitigacemi. Test Stress CPU, který se skládá z podtestů vytěžování procesoru různými způsoby. Výsledná hodnota je pak prezentována jako sumarizovaný celek. Tak tomu bývá například u benchmarkovacích programů typu Cinebench. Jelikož se ani v jednom z případů takového sumarizovaného testu neprokázal statisticky významný rozdíl mezi naměřenými hodnotami, autor došel k závěru, že takovými to programy nelze spolehlivě otestovat ztrátu výkonu procesorů a místo toho doporučuje testování provádět pomocí jednotlivých testů, které vždy měří určitou operaci.

V ostatních případech testování byly prokázány statisticky významné rozdíly mezi režimy, lze konstatovat, že aplikováním mitigací opravdu došlo k poklesu výkonu procesoru i dalších částí počítače. Nejedná se tedy o chybu v měření či jiný náhodný vliv, který by zkreslil výsledky měření. Statistická nevýznamnost rozdílu byla prokázána především u procesoru i7-4790K, která může být způsobena malým vzorkem testovacích dat, či jejich normálním rozdělením, které kolísalo těstě u referenční hodnoty $\alpha = 0{,}05$.

6 Závěr

Autor analyzoval ztrátu výkonu procesorů, při opravách zranitelností typu útok postranním kanálem (Spectre, Meltdown, Forshadow). Výkon komparoval pomocí výkonnostních testů jak na fyzických, tak virtualizovaných systémech. Definoval základní pojmy a jednotlivé chyby procesorů, jejich odstranění a boj proti nim. Dále uvedl odstranění chyb ve virtualizovaném prostředí. Autor vybral testovací programy, vypracoval testovací scénáře pro otestování procesorů v různých omezených režimech výkonu. Testoval oblast vstupně výstupního výkonu ve fyzickém prostředí i vliv chyb na hyper-threading v prostředí virtualizovaném. Postupně povypínal jednotlivé služby počítače pro testování jejich vlivu na výkon systému. Nakonec provedl jednotlivá měření a vyhodnotil intervalové četnosti ztrát výkonu v různých režimech testování. Zda bylo testování statisticky významné potvrdil provedením jednoduché analýzy rozptylu a mnohonásobného porovnávání jednotlivých režimů procesorů u vybraných typů testů.

Autor dospěl ke zjištění, že výše ztráty výkonu procesorů jednoznačně závisí na stáří procesoru, typu prostředí a také práci, kterou vykonává. U starších typů procesorů dochází k větším ztrátám z důvodu staršího návrhu architektury, která je potřeba opravit pomocí softwarových záplat pro zabezpečení uživatelských dat. Nové procesory mají již vůči známým hrozbám aplikovány záplaty přímo v čipu, a proto dopady na výkon ve fyzickém prostředí nejsou tak znatelné. Ve virtualizovaném prostředí dochází ke stejným chybám, ale s mnohonásobně větším dopadem na ztrátu výkonu při aplikování mitigací. Jedním z nejzásadnějších ztrát výkonu je ochrana před útokem Foreshadow, kde je nutno vypnout simultánní multi-threading, čímž je snížen výkon až o polovic. Důležitou roli také hraje to, jaký účel stroj plní a jaké funkce nejčastěji vykonává. Pokud jde o osobní počítače, kde se provádí především kancelářská práce, e-maily a surfování na internetu, ztráta výkonu nebude skoro znatelná. Vetší ztrátu nesou stroje, kde je velmi často prováděna kompilace kódu, zpracování více procesů či vláken najednou a také tam, kde procesor musí často volat instrukce přerušení a přepínat mezi user a kernel space. Nejrizikovější skupinou, co se týče ztráty výkonu jsou databázové a webové servery, kde dochází ke znatelnému snížení výkonu. Připočteme-li provozování databázových a webových serverů ve virtualizovaném prostředí s maximální možnou ochranou před útoky, jedná se o tak razantní snížení výkonu, že by provozovatelé museli minimálně zdvojnásobit své

hardwarové zdroje pro udržení aktuálních provozovaných služeb. Další možností je upgradovat procesory na nejnovější a nejvýkonnější modely, čímž zároveň není do budoucna zajištěno dalších ztrát výkonu díky novým neodhaleným útokům vedeným postranním kanálem.

7 Seznam použitých zdrojů

1. **Trčálek, Antonín.** Jak se vyrábí procesory: Od písku po čip. *Živě.cz.* [Online] 28. 3 2013. [Citace: 12. 11 2019.] https://www.zive.cz/clanky/jak-se-vyrabi-procesory-od-pisku-po-cip/sc-3-a-168201/default.aspx#articleStart.

2. **Pinker, Jiří.** *Mikroprocesory a mikropočítače.* Praha : Nakladatelství BEN, 204. ISBN: 80-7300-110-1.

3. **Meltdown & Spectre** - Kernel Side-Channel Attacks - CVE-2017-5754 CVE-2017-5753 CVE-2017-5715. *RedHat customers portal.* [Online] 3. 1 2018. [Citace: 19. 11 2019.] https://access.redhat.com/security/vulnerabilities/speculativeexecution.

4. **Tišnovský, Pavel.** Techniky zvýšení výpočetního výkonu mikroprocesorů. *root.cz.* [Online] 16. 5 2008. [Citace: 20. 11 2019.] https://www.root.cz/clanky/techniky-zvyseni-vypocetniho-vykonu-mikroprocesoru/#k05.

5. **Pavlík, Vojtěch.** Spectre a Meltdown - jak fungují a co s nimi (Vojtěch Pavlík). *YouTube.* [Online] InstallFest, 4. 3 2018. [Citace: 26. 11 2019.] https://www.youtube.com/watch?v=rwbs-PN0Vpw.

6. **Owens, Clinton.** SlidePLayer - Pipelining, Branch Prediction, Trends - Slide 25/41. *SlidePLayer.* [Online] [Citace: 4. 12 2019.] https://slideplayer.com/slide/6829903/.

7. **Tišnovský, Pavel.** Techniky zvýšení výpočetního výkonu počítačů. *Root.cz.* [Online] 29. 5 2008. [Citace: 4. 12 2019.] https://www.root.cz/clanky/techniky-zvyseni-vypocetniho-vykonu-pocitacu/#k02.

8. **Tišnovský, Pavel.** Techniky zvýšení výkonu mikroprocesorů 2. *Root.cz.* [Online] 22. 5 2008. [Citace: 4. 12 2019.] https://www.root.cz/clanky/techniky-zvyseni-vykonu-mikroprocesoru-2/#k04.

9. **Zezula, Ladislav.** Výuka assembleru - 4. Porovnávací instrukce a instrukce skoků. *Výuka assdembleru.* [Online] 2003. [Citace: 15. 1 2020.] http://www.zezula.net/cz/teach/assembler_04.html.

10. **Zezula, Ladislav.** Výuka assembleru - 7. Procedury a funkce. *Výuka Assembleru.* [Online] 2003. [Citace: 15. 1 2020.] http://www.zezula.net/cz/teach/assembler_07.html.

11. **Novotný, Daniel.** Root.cz - Pod pokličkou vašeho počítače: adresování procesorů Intel x86. *Root.cz.* [Online] 15. 2 2005. [Citace: 18. 1 2020.] https://www.root.cz/clanky/adresovani-procesoru-intel-x86/.

12. **Intel - Support** - Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 3A: System Programming Guide, Part 1. *Intel.com.* [Online] 24. 5 2018. [Citace: 18. 1 2020.] https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html.

13. **is.mendelu.cz/eknihovna** - Virtuální paměť. *IS Mendelova univerzita v Brně - Eknihovna.* [Online] [Citace: 18. 1 2020.] https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=9986.

14. **Arpaci-Dusseau, Remzi H a Arpaci-Dusseau, Andrea C.** *Operating Systems: Three Easy Pieces.* místo neznámé : CreateSpace Independent Publishing, 2018. ISBN: 9781985086593.

15. **Kerrisk, Michael.** *The Linux Programming Interface: A Linux and UNIX System Programming Handbook.* místo neznámé : No Starch Press, 2010. ISBN: 978-1-59327-220-3.

16. **Dráb, Martin.** *Jádro systému Windows.* Brno : Computer Press, 2011. ISBN: 978-80-251-2731-5.

17. **Harding, Scharon.** toms'sHardware - What Is Simultaneous Multithreading? A Basic Definition. *tom'sHardware.* [Online] 23. 8 2018. [Citace: 21. 1 2020.] https://www.tomshardware.com/reviews/simultaneous-multithreading-definition,5762.html.

18. **Kocher, Paul, a další.** Spectre Attacks: Exploiting Speculative Execution. *Spectreattack.com.* [Online] 2019. [Citace: 21. 1 2020.] https://spectreattack.com/spectre.pdf.

19. **Redakce.** V čem spočívají Meltdown a Spectre? Zneužívají optimalizací procesorů. *Root.cz.* [Online] 9. 1 2018. [Citace: 21. 1 2020.] https://www.root.cz/clanky/v-cem-spocivaji-meltdown-a-spectre-zneuzivaji-optimalizaci-procesoru/.

20. **Pavlík, Vojtěch.** LinuxDays 2018 - Meltdownem to neskončilo. L1TF, POP SS, TLBleed a další. - Vojtěch Pavlík. *YouTube.com.* [Online] 16. 10 2018. [Citace: 21. 1 2020.] https://www.youtube.com/watch?v=mIKSXv0Cgjg.

21. **Intel.** Deep Dive: Indirect Branch Predictor Barrier. *Software.Intel.com - Security software, Developer Zone.* [Online] 2020. [Citace: 24. 1 2020.] https://software.intel.com/security-software-guidance/insights/deep-dive-indirect-branch-predictor-barrier.

22. **Intel.** Deep Dive: Indirect Branch Restricted Speculation. *Software.Intel.com - Security software, Developer Zone.* [Online] 2020. [Citace: 24. 1 2020.] https://software.intel.com/security-software-guidance/insights/deep-dive-indirect-branch-restricted-speculation.

23. **Intel.** Deep Dive: Single Thread Indirect Branch Predictors. *Software.Intel.com - Security software, Developer Zone.* [Online] 2020. [Citace: 24. 1 2020.] https://software.intel.com/security-software-guidance/insights/deep-dive-single-thread-indirect-branch-predictors.

24. **Lipp, Moritz, a další.** Meltdown: Reading Kernel Memory from User Space. *Meltdownattack.com.* [Online] 2018. [Citace: 24. 1 2020.] https://meltdownattack.com/meltdown.bib.

25. **Kilián, Karel.** Více systémů na jednom počítači: Vybrali jsme nejlepší programy pro virtualizaci Více na: https://www.zive.cz/clanky/vice-systemu-na-jednom-pocitaci-vybrali-jsme-nejlepsi-programy-pro-virtualizaci/sc-3-a-195920/default.aspx#part=1. *Živě.cz.* [Online] 2. 1 2013. [Citace: 28. 1 2020.] https://www.zive.cz/clanky/vice-systemu-na-jednom-pocitaci-vybrali-jsme-nejlepsi-programy-pro-virtualizaci/sc-3-a-195920/default.aspx#part=1.

26. **Intel.** Protecting Our Customers through the Lifecycle of Security Threats. *Newsroom.intel.com.* [Online] 14. 8 2018. [Citace: 28. 1 2020.] https://newsroom.intel.com/editorials/protecting-our-customers-through-lifecycle-security-threats/#gs.usanxg.

8 Přílohy

8.1 Příloha A – Statistická testování Hackbench

Konf.	16 Thread											
CPU	i7-740Q				i7-4790K				i5-7267U			
Třída	auto	Spe=on	Mel=on	off	auto	Spe=on	Mel=on	off	auto	Spe=on	Mel=on	off
Pozorované hodnoty	349,0	252,7	316,9	246,5	116,6	71,9	86,1	78,8	266,0	170,1	176,1	185,3
	365,9	250,9	341,2	253,8	114,7	81,2	77,3	65,6	280,0	165,9	199,9	191,6
	317,8	248,9	322,8	250,6	118,8	80,1	77,4	75,5	267,7	170,7	194,4	185,7
	326,0		341,9			72,7	76,9	65,4			194,4	
	319,8		339,7			72,4	86,7	74,2			184,6	
	332,3		355,4			71,9	86,8	66,4			192,2	
	313,1		360,6			72,6	84,6	76,9			182,9	
	327,4		361,0			72,4	88,4	65,6			186,7	
	321,8		341,2			72,1	77,1	65,7			190,5	
						72,4	76,1	64,9			192,0	
						73,2	76,9	75,3			182,6	
						72,6	76,5	78,1			188,1	
						72,0	85,4	75,9				
						81,3	77,0	79,3				
						72,3	76,8	75,7				
Rozsah	9	3	9	3	3	15	15	15	3	3	12	3

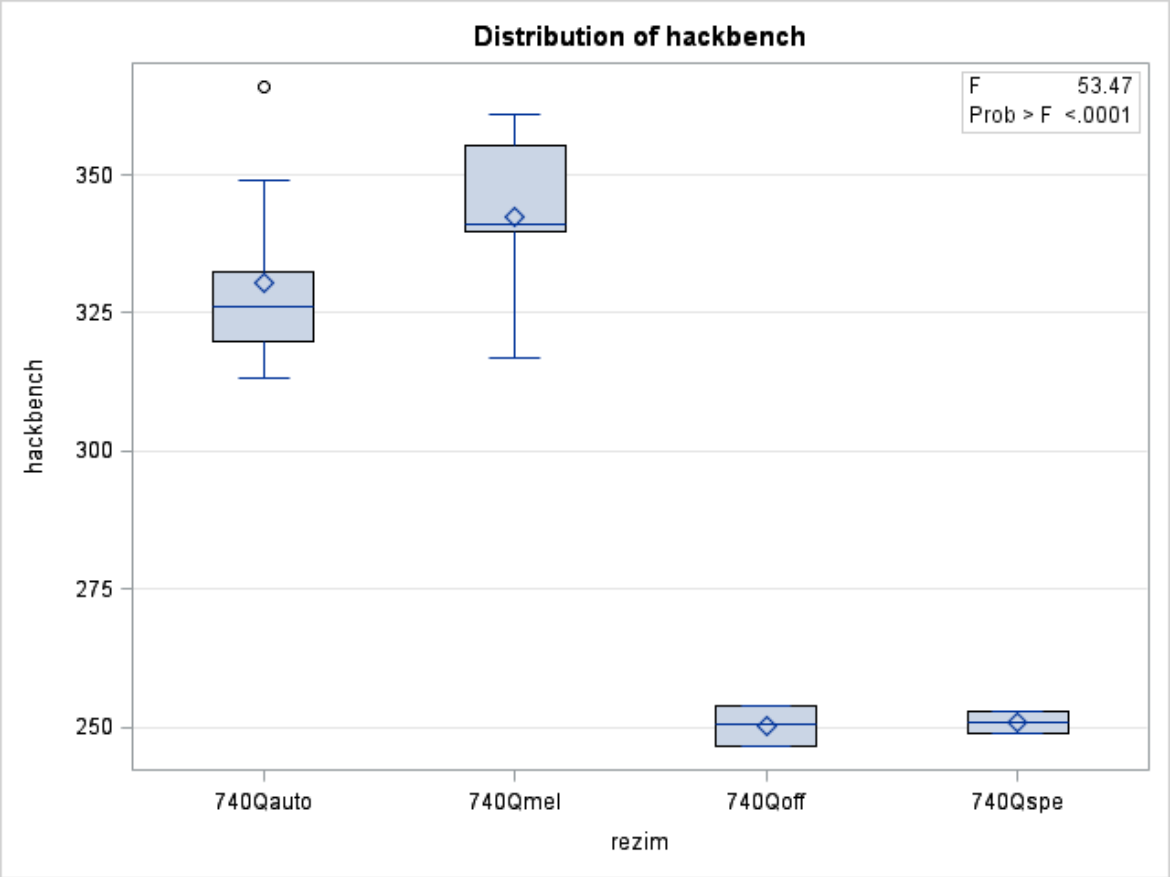
Tabulka č. 33 – Pokusný plán pro fyzickou část testu Hackbench [autor]

The GLM Procedure

Dependent Variable: hackbench

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	33749.27151	11249.75717	53.47	<.0001
Error	20	4207.90236	210.39512		
Corrected Total	23	37957.17387			

Obrázek č. 9 – Procedura GLM pro fyzickou část testu Hackbench i7-740Q [autor]



Obrázek č. 10 – Procedura Boxlpot pro fyzickou část testu Hackbench i7-740Q [autor]

Levene's Test for Homogeneity of hackbench Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	239164	79721.2	0.94	0.4414
Error	20	1702345	85117.3		

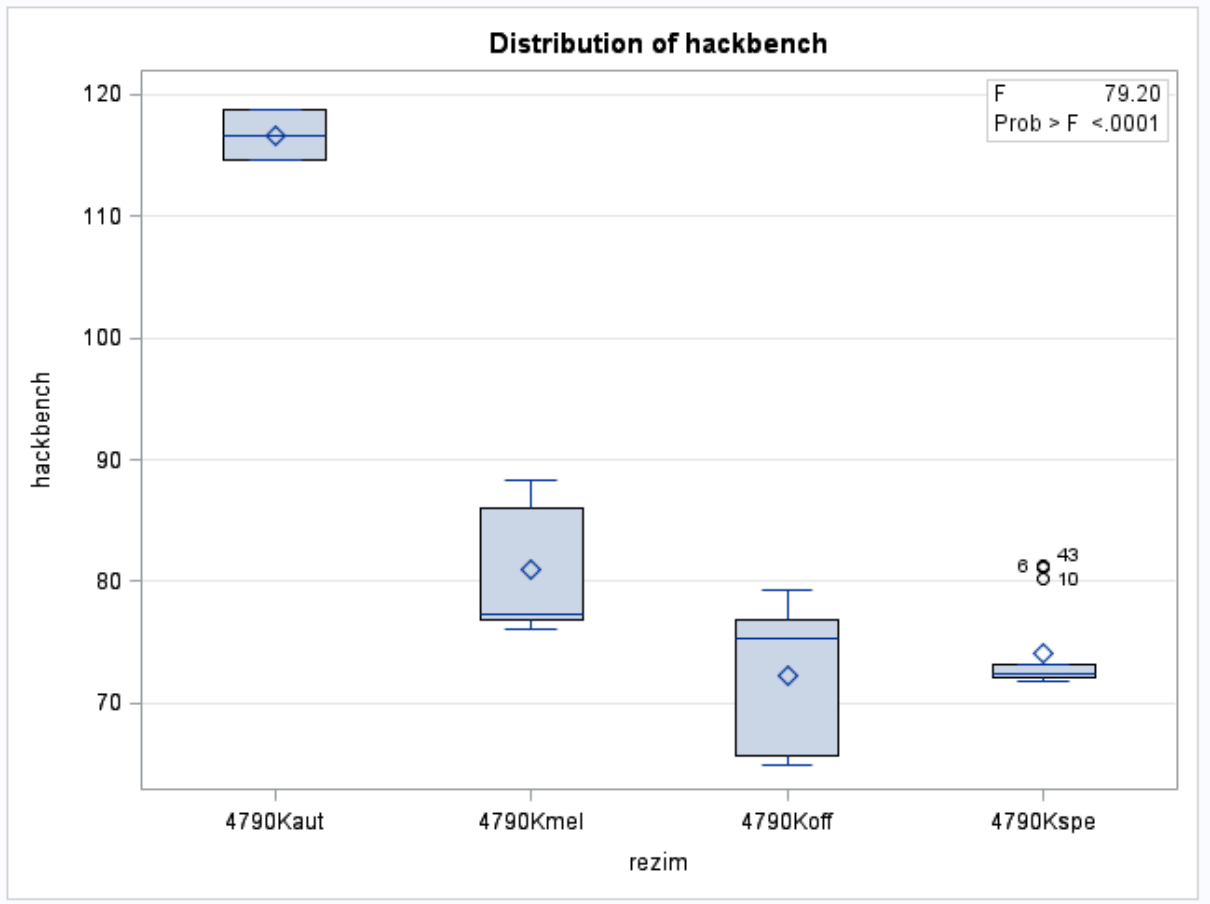
Obrázek č. 11 – Leveneův test pro fyzickou část testu Hackbench i7-740Q [autor]

The GLM Procedure

Dependent Variable: hackbench

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	5336.716034	1778.905345	79.20	<.0001
Error	43	965.834989	22.461279		
Corrected Total	46	6302.551023			

Obrázek č. 12 – Procedura GLM pro fyzickou část testu Hackbench i7-4790K [autor]



Obrázek č. 13 – Procedura Boxlpot pro fyzickou část testu Hackbench i7-4790K [autor]

Levene's Test for Homogeneity of hackbench Variance ANOVA of Squared Deviations from Group Means					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	3815.2	1271.7	5.15	0.0039
Error	43	10609.6	246.7		

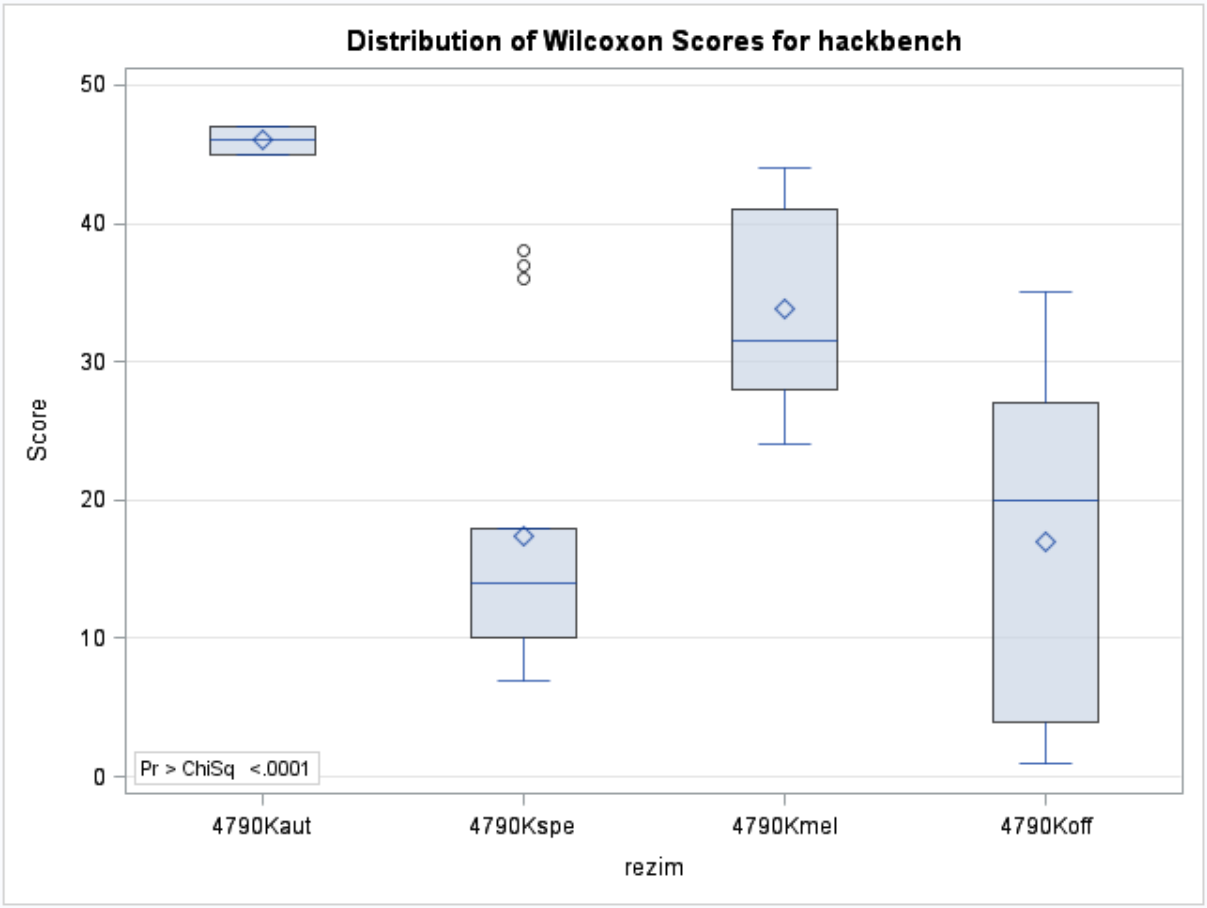
Obrázek č. 14 – Leveneův test pro fyzickou část testu Hackbench i7-4790K [autor]

Kruskal-Wallis Test	
Chi-Square	22.3441
DF	3
Pr > Chi-Square	<.0001

Obrázek č. 15 – Neparametrický Kruskal-Wallisuv test pro fyzickou část Hackbench i7-4790K[autor]

Pairwise Two-Sided Multiple Comparison Analysis			
Dwass, Steel, Critchlow-Fligner Method			
Variable: hackbench			
rezim	Wilcoxon Z	DSCF Value	Pr > DSCF
4790Kaut vs. 4790Kspe	2.6656	3.7697	0.0385
4790Kaut vs. 4790Kmel	2.6458	3.7417	0.0407
4790Kaut vs. 4790Koff	2.6656	3.7697	0.0385
4790Kspe vs. 4790Kmel	-3.5351	4.9994	0.0023
4790Kspe vs. 4790Koff	0.1867	0.2640	0.9977
4790Kmel vs. 4790Koff	3.4042	4.8143	0.0037

Obrázek č. 16 – Párové porovnání procedury npar1way pro fyzickou část Hackbench i7-4790K [autor]



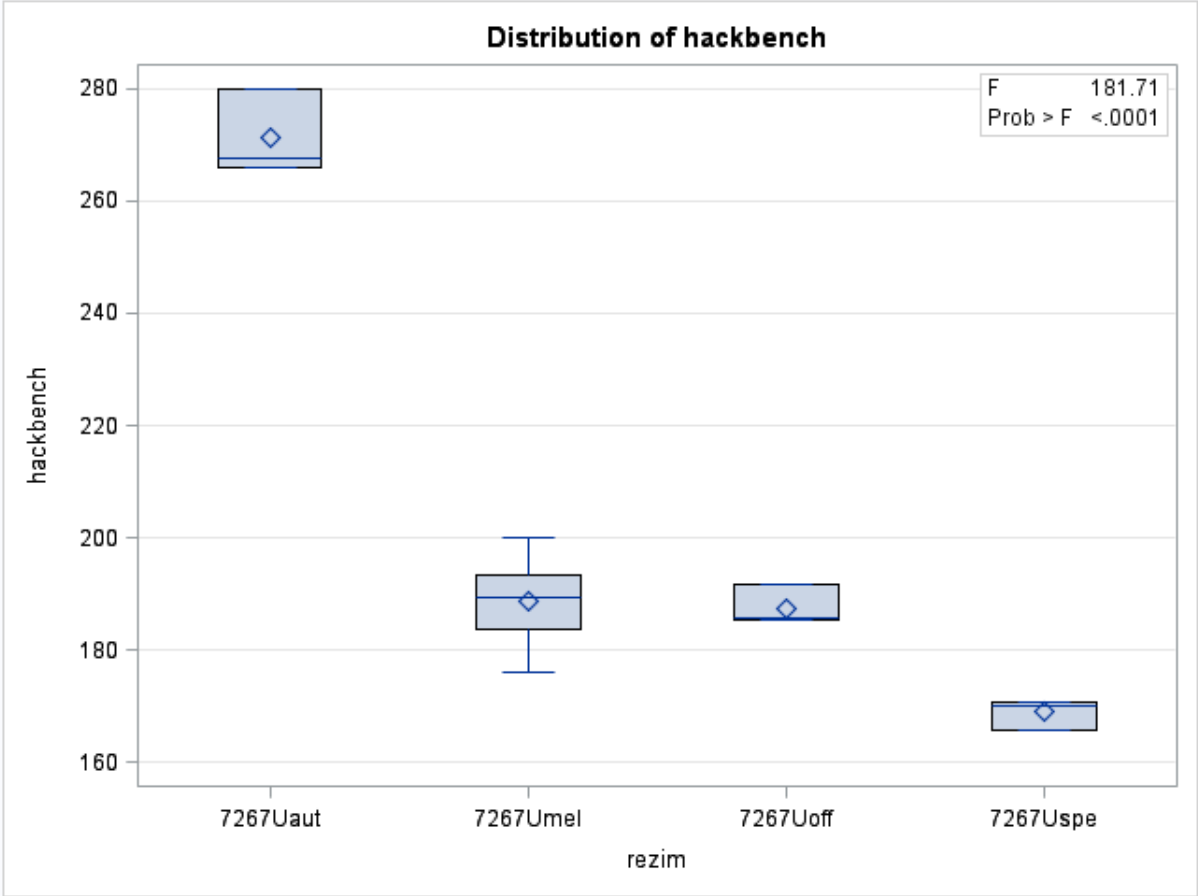
Obrázek č. 17 – Procedura npar1way pro fyzickou část testu Hackbench i7-4790K [autor]

The GLM Procedure

Dependent Variable: hackbench

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	19993.69212	6664.56404	181.71	<.0001
Error	17	623.49652	36.67627		
Corrected Total	20	20617.18864			

Obrázek č. 18 – Procedura GLM pro fyzickou část testu Hackbench i5-7267U [autor]



Obrázek č. 19 – Procedura Boxlpot pro fyzickou část testu Hackbench i5-7267U [autor]

Levene's Test for Homogeneity of hackbench Variance
ANOVA of Squared Deviations from Group Means

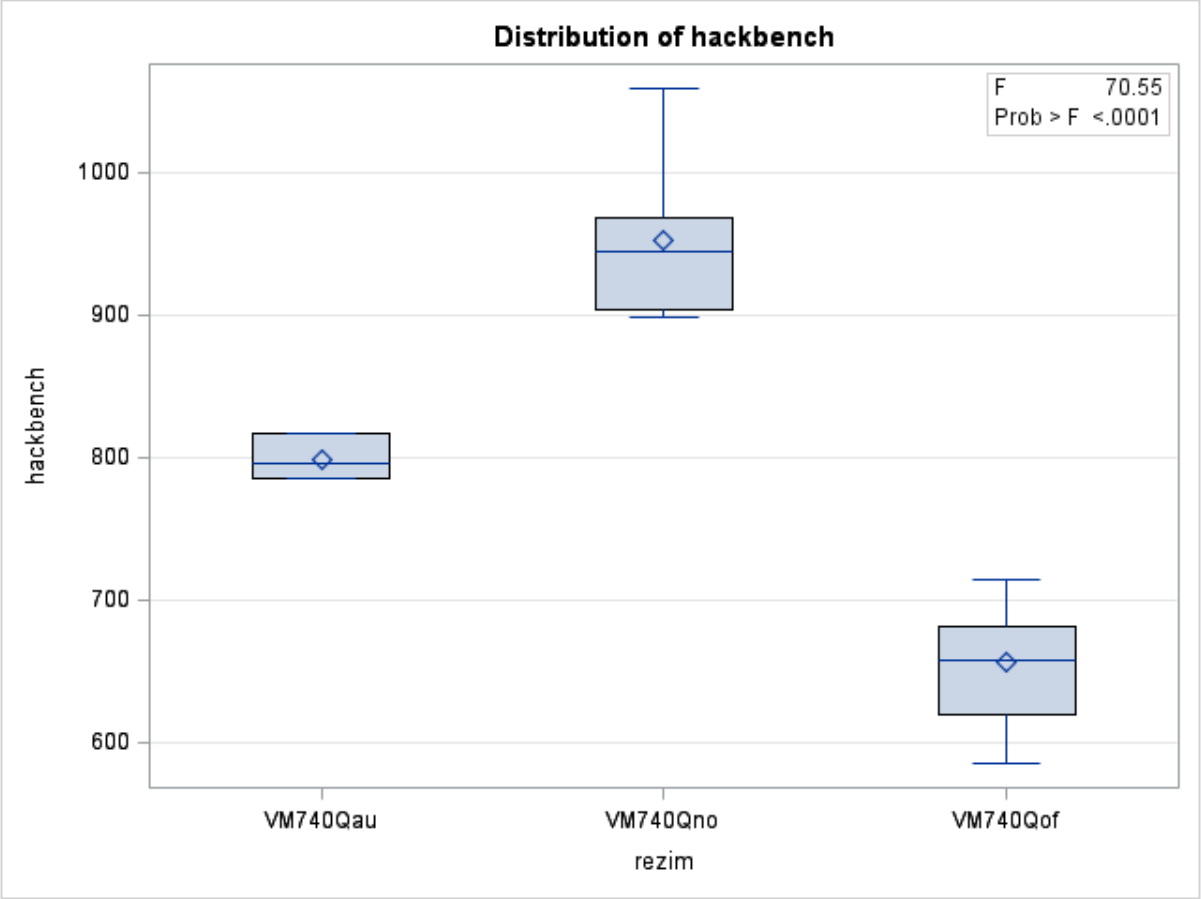
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	4526.1	1508.7	0.84	0.4886
Error	17	30386.5	1787.4		

Obrázek č. 20 – Leveneův test pro fyzickou část testu Hackbench i5-7267U [autor]

Konf.	16 Thread

CPU	i7-740Q			i7-4790K			i5-7267U		
Třída	**auto**	**nosmt**	**off**	**auto**	**nosmt**	**off**	**auto**	**nosmt**	**off**
Pozorované hodnoty	795,179	903,126	676,506	229,784	264,596	101,395	565,305	622,208	210,860
	816,720	897,849	585,149	227,106	270,299	106,881	609,939	630,967	207,613
	785,224	953,691	610,567	229,730	267,742	79,017	603,568	640,405	214,427
		967,843	681,388			97,943	577,065		
		1059,045	657,118			109,271	604,267		
		936,169	649,847			105,420	591,838		
			712,608			108,792			
			713,737			70,223			
			619,787			90,368			
						84,097			
						110,502			
						77,994			
						89,542			
						109,231			
						83,268			
Rozsah	3	6	9	3	3	15	6	3	3

Tabulka č. 34 – Pokusný plán pro virtualizovanou část testu Hackbench [autor]



Obrázek č. 21 – Procedura Boxlpot pro virtualizovanou část testu Hackbench i7-740Q [autor]

The GLM Procedure

Dependent Variable: hackbench

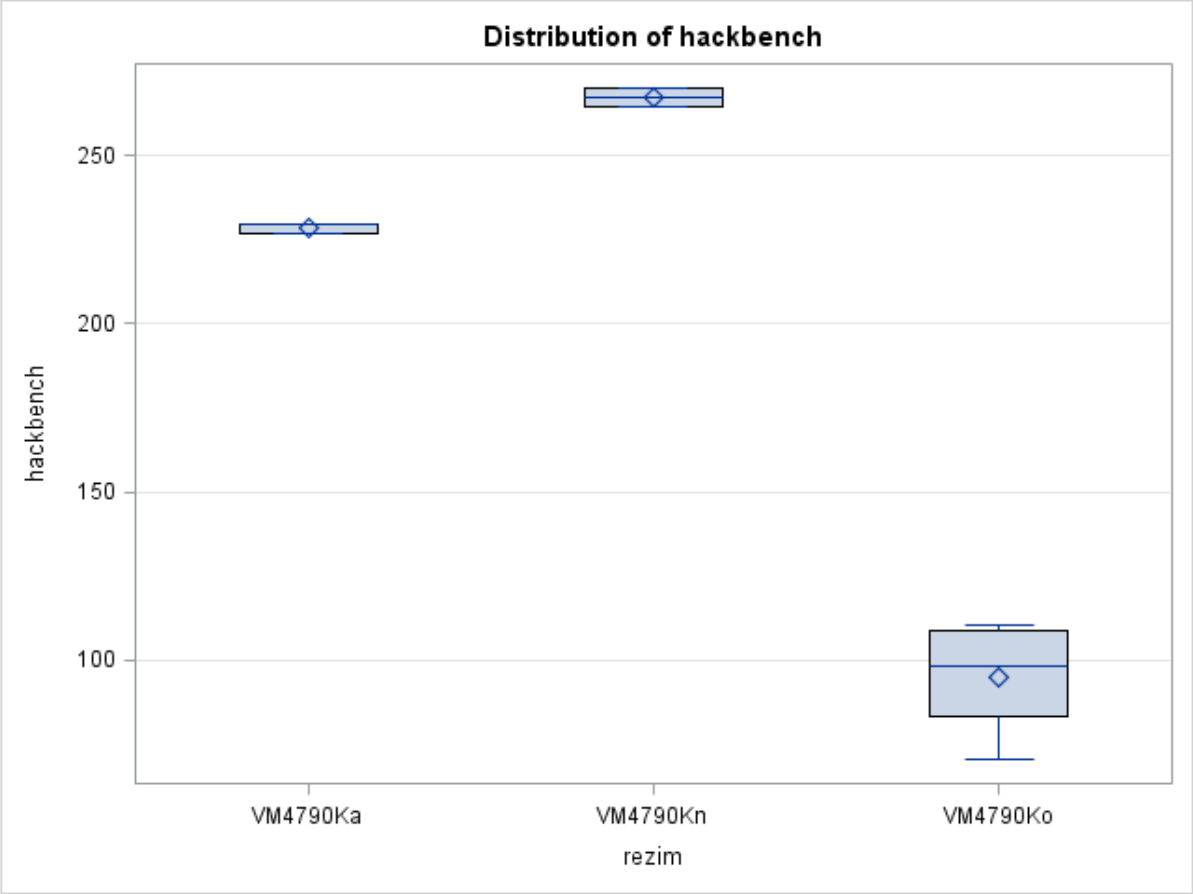
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	318260.4229	159130.2115	70.55	<.0001
Error	15	33833.7958	2255.5864		
Corrected Total	17	352094.2187			

Obrázek č. 22 – Procedura GLM pro virtualizovanou část testu Hackbench i7-740Q [autor]

Levene's Test for Homogeneity of hackbench Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	14827673	7413836	0.95	0.4092
Error	15	1.1719E8	7812551		

Obrázek č. 23 – Leveneův test pro virtualizovanou část testu Hackbench i7-740Q [autor]



Obrázek č. 24 – Procedura Boxlpot pro virtualizovanou část testu Hackbench i7-4790K [autor]

The GLM Procedure

Dependent Variable: hackbench

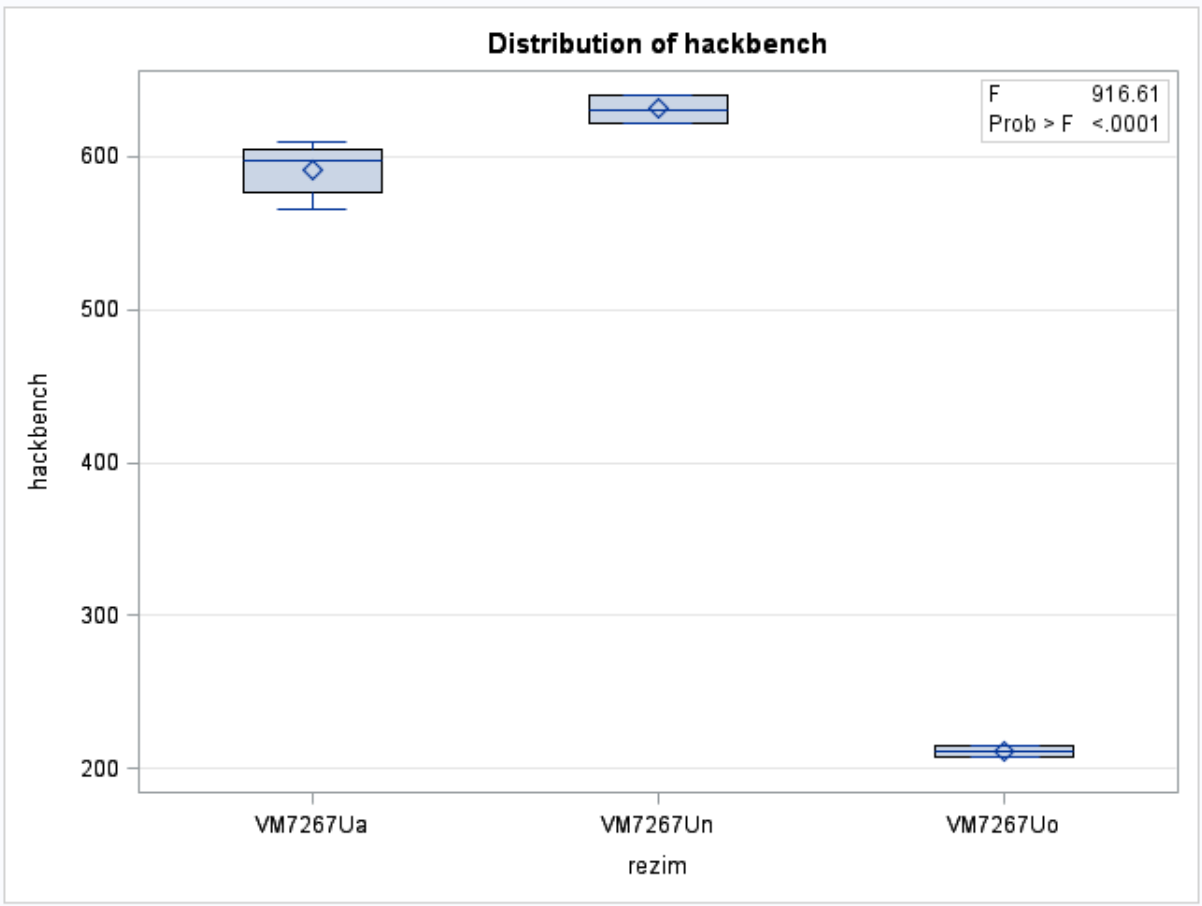
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	102935.0144	51467.5072	353.15	<.0001
Error	18	2623.2586	145.7366		
Corrected Total	20	105558.2730			

Obrázek č. 25 – Procedura GLM pro virtualizovanou část testu Hackbench i7-4790K [autor]

Levene's Test for Homogeneity of hackbench Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	123854	61927.0	3.53	0.0507
Error	18	315337	17518.7		

Obrázek č. 26 – Leveneův test pro virtualizovanou část testu Hackbench i7-4790K [autor]



Obrázek č. 27 – Procedura Boxlpot pro virtualizovanou část testu Hackbench i5-7267U [autor]

The GLM Procedure

Dependent Variable: hackbench

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	352523.5397	176261.7699	916.61	<.0001
Error	9	1730.6837	192.2982		
Corrected Total	11	354254.2234			

Obrázek č. 28 – Procedura GLM pro virtualizovanou část testu Hackbench i5-7267U [autor]

Levene's Test for Homogeneity of hackbench Variance
ANOVA of Squared Deviations from Group Means

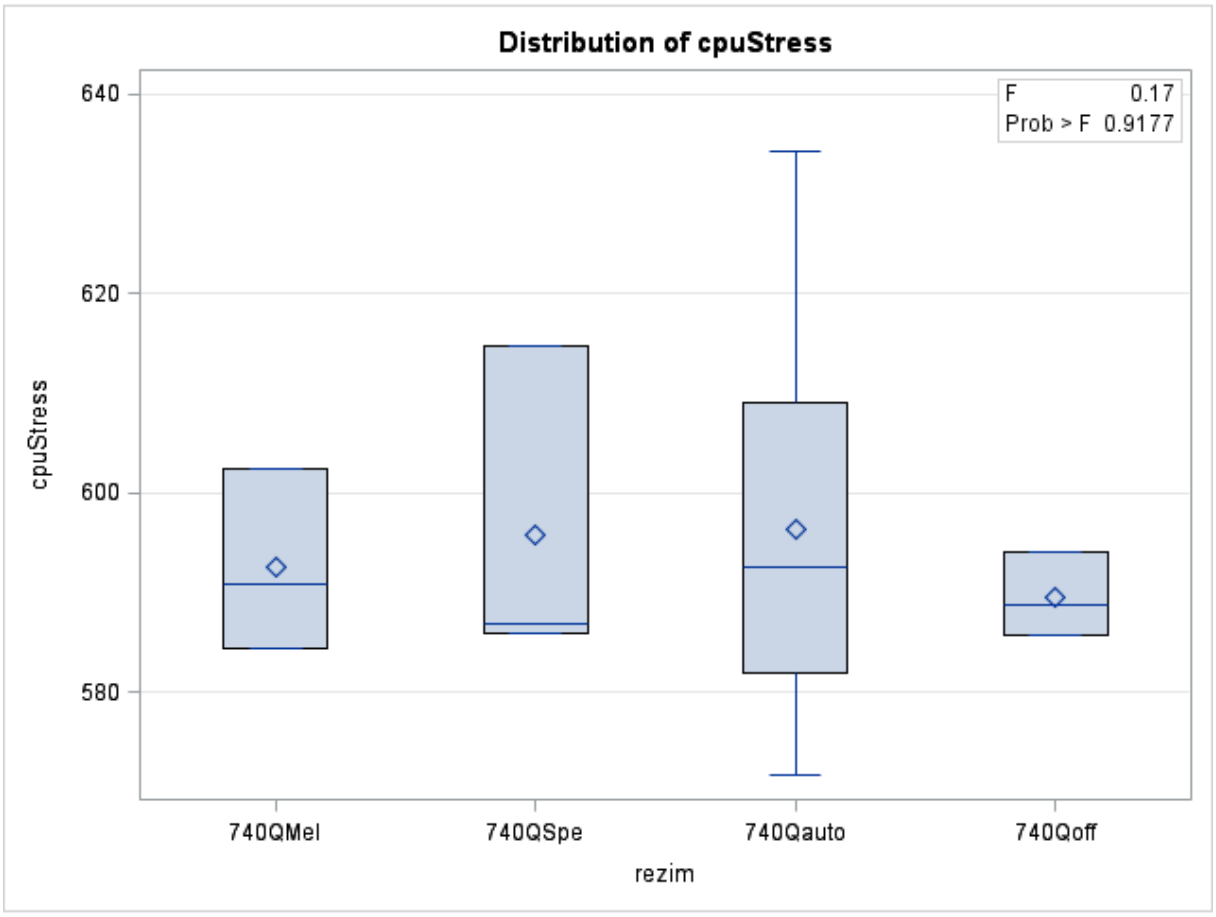
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	155916	77958.0	2.26	0.1599
Error	9	310005	34445.0		

Obrázek č. 29 – Leveneův test pro virtualizovanou část testu Hackbench i5-7267U [autor]

8.2 Příloha B – Statistická testování Stress-NG – Stress CPU

Konf.	CPU Stress											
CPU	i7-740Q				i7-4790K				i5-7267U			
Třída	auto	Spe=on	Mel=on	off	auto	Spe=on	Mel=on	off	auto	Spe=on	Mel=on	off
Pozorované hodnoty	634,3	586,9	584,4	588,8	1951,0	1951,7	1957,8	1931,1	931,4	909,2	900,6	904,0
	621,6	614,8	590,8	594,1	1972,8	1930,1	1985,9	1942,0	912,8	910,1	913,1	910,4
	571,7	585,8	602,3	585,8	1974,8	1970,0	1985,9	1939,8	934,2	907,0	903,2	915,8
	609,0											
	587,2											
	595,5											
	578,6											
	595,2											
	602,7											
	581,9											
	580,8											
	592,6											
	624,8											
	584,6											
	585,4											
Rozsah	15	3	3	3	3	3	3	3	3	3	3	3

Tabulka č. 35 – Pokusný plán pro fyzickou část testu Stress CPU [autor]



Obrázek č. 30 – Procedura Boxlpot pro fyzickou část testu Stress CPU i7-740Q [autor]

The GLM Procedure

Dependent Variable: cpuStress

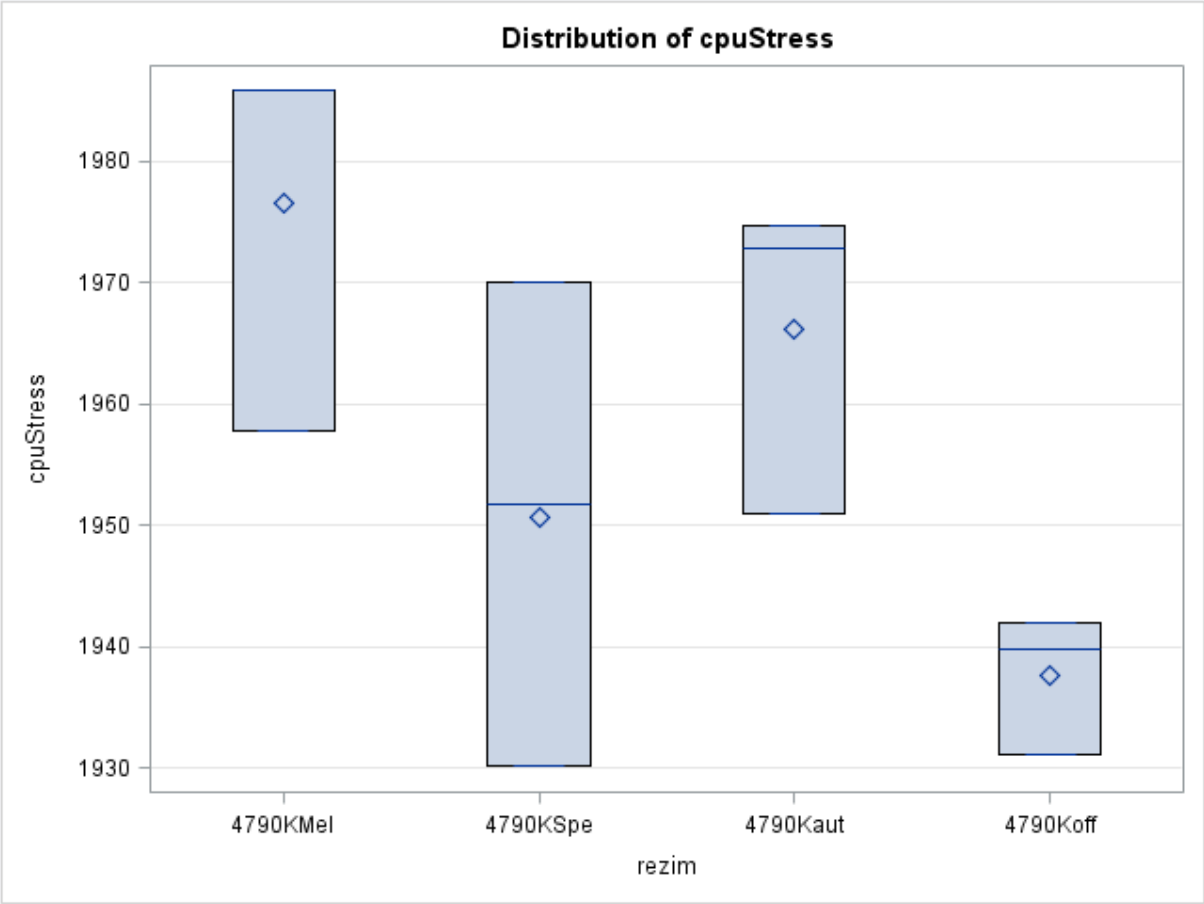
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	138.838050	46.279350	0.17	0.9177
Error	20	5558.948333	277.947417		
Corrected Total	23	5697.786383			

Obrázek č. 31 – Procedura GLM pro fyzickou část testu Stress CPU i7-740Q [autor]

Levene's Test for Homogeneity of cpuStress Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	367214	122405	1.08	0.3803
Error	20	2267514	113376		

Obrázek č. 32 – Leveneův test pro fyzickou část testu Stress CPU i7-740Q [autor]



Obrázek č. 33 – Procedura Boxlpot pro fyzickou část testu Stress CPU i7-4790K [autor]

The GLM Procedure

Dependent Variable: cpuStress

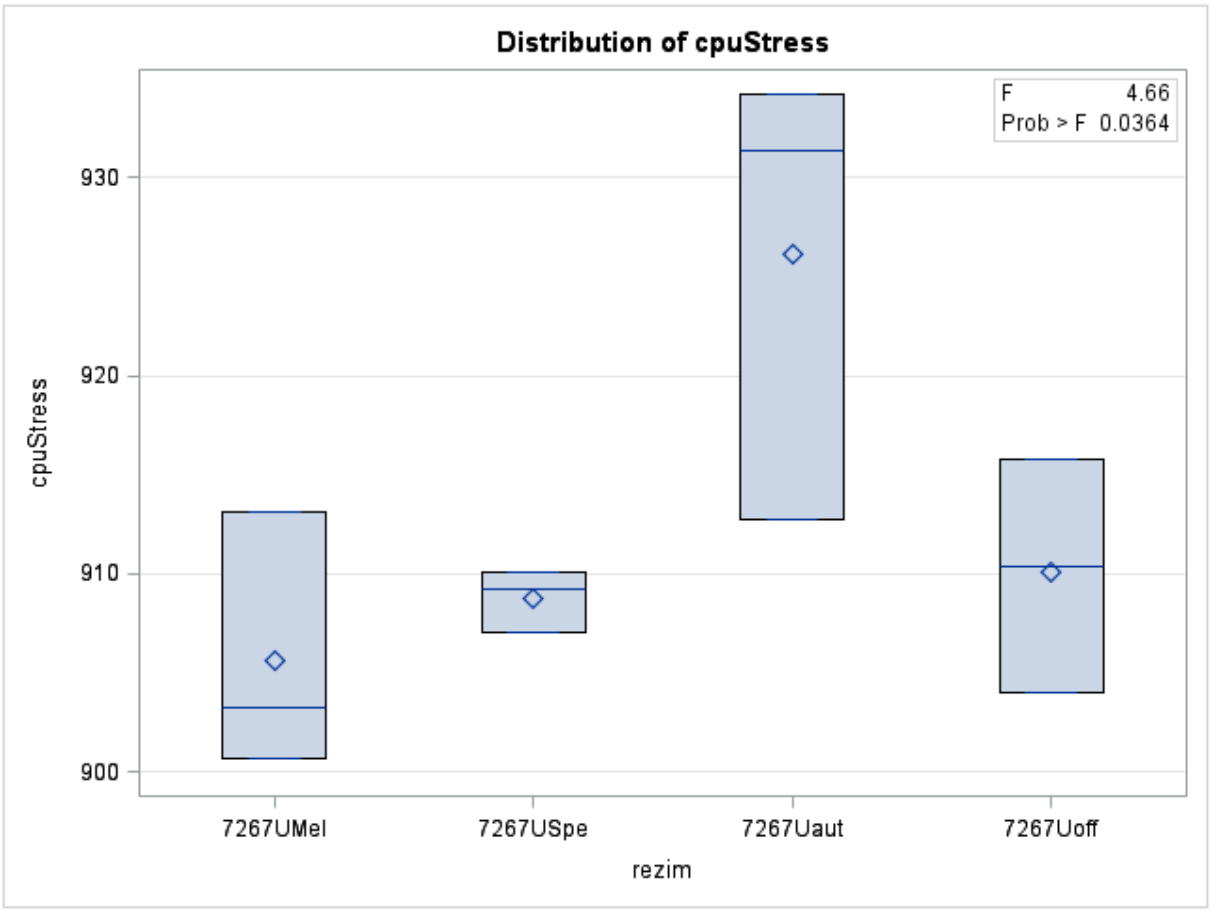
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	2638.163467	879.387822	4.04	0.0507
Error	8	1740.053733	217.506717		
Corrected Total	11	4378.217200			

Obrázek č. 34 – Procedura GLM pro fyzickou část testu Stress CPU i7-4790K [autor]

Levene's Test for Homogeneity of cpuStress Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	94245.3	31415.1	1.45	0.2990
Error	8	173280	21660.1		

Obrázek č. 35 – Leveneův test pro fyzickou část testu Stress CPU i7-4790K [autor]



Obrázek č. 36 – Procedura Boxlpot pro fyzickou část testu Stress CPU i5-7267U [autor]

The GLM Procedure

Dependent Variable: cpuStress

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	757.259133	252.419711	4.66	0.0364
Error	8	433.507267	54.188408		
Corrected Total	11	1190.766400			

Obrázek č. 37 – Procedura GLM pro fyzickou část testu Stress CPU i5-7267U [autor]

Levene's Test for Homogeneity of cpuStress Variance
ANOVA of Squared Deviations from Group Means

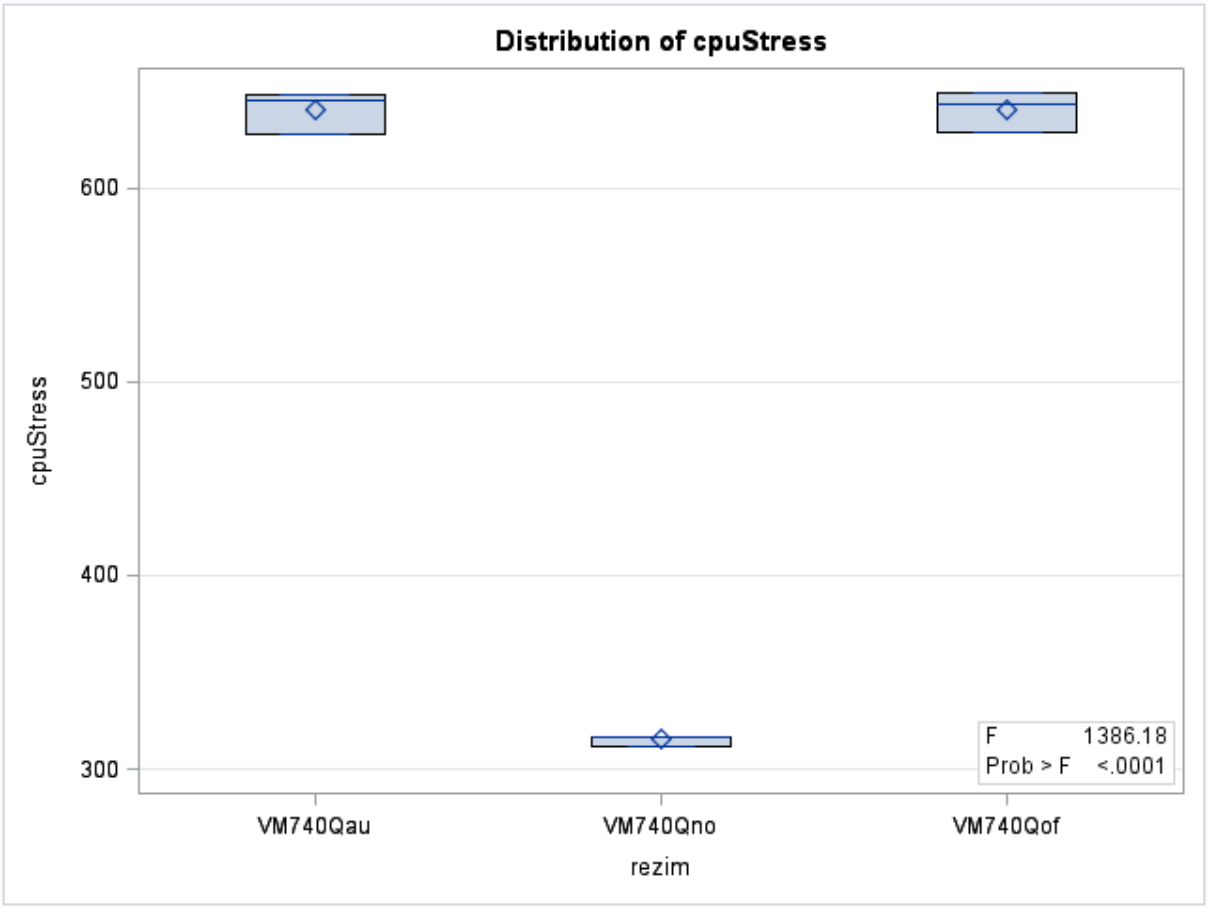
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	13169.5	4389.8	2.44	0.1397
Error	8	14415.1	1801.9		

Obrázek č. 38 – Leveneův test pro fyzickou část testu Stress CPU i5-7267U [autor]

Konf.	**CPU Stress**

CPU	i7-740Q			i7-4790K			i5-7267U		
Třída	auto	nosmt	off	auto	nosmt	off	auto	nosmt	off
Pozorované hodnoty	628,10	316,68	643,18	1969,23	1332,15	1966,67	930,23	636,74	919,07
	647,89	312,07	649,04	1949,15	1335,72	1962,48	919,44	631,97	919,98
	644,92	317,06	628,92	1950,67	1333,75	1963,32	917,06	636,63	917,87
Rozsah	3	3	3	3	3	3	3	3	3

Tabulka č. 36 – Pokusný plán pro virtualizovanou část testu Stress CPU [autor]



Obrázek č. 39 – Procedura Boxlpot pro virtualizovanou část testu Stress CPU i7-740Q [autor]

The GLM Procedure

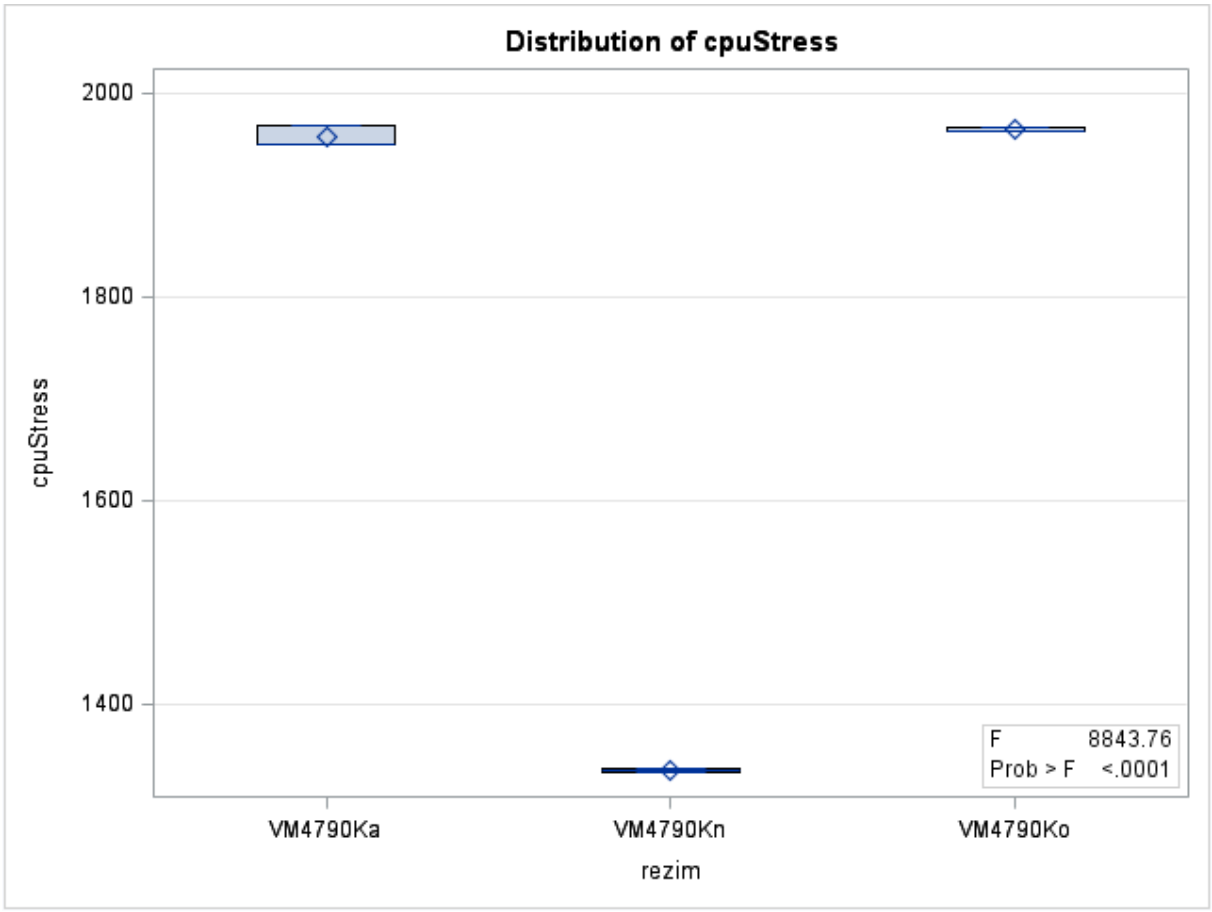
Dependent Variable: cpuStress

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	211343.1858	105671.5929	1386.18	<.0001
Error	6	457.3919	76.2320		
Corrected Total	8	211800.5776			

Obrázek č. 40 – Procedura GLM pro virtualizovanou část testu Stress CPU i7-740Q [autor]

Levene's Test for Homogeneity of cpuStress Variance ANOVA of Squared Deviations from Group Means					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	9419.8	4709.9	1.73	0.2551
Error	6	16332.5	2722.1		

Obrázek č. 41 – Leveneův test pro virtualizovanou část testu Stress CPU i7-740Q [autor]



Obrázek č. 42 – Procedura Boxlpot pro virtualizovanou část testu Stress CPU i7-4790K [autor]

The GLM Procedure

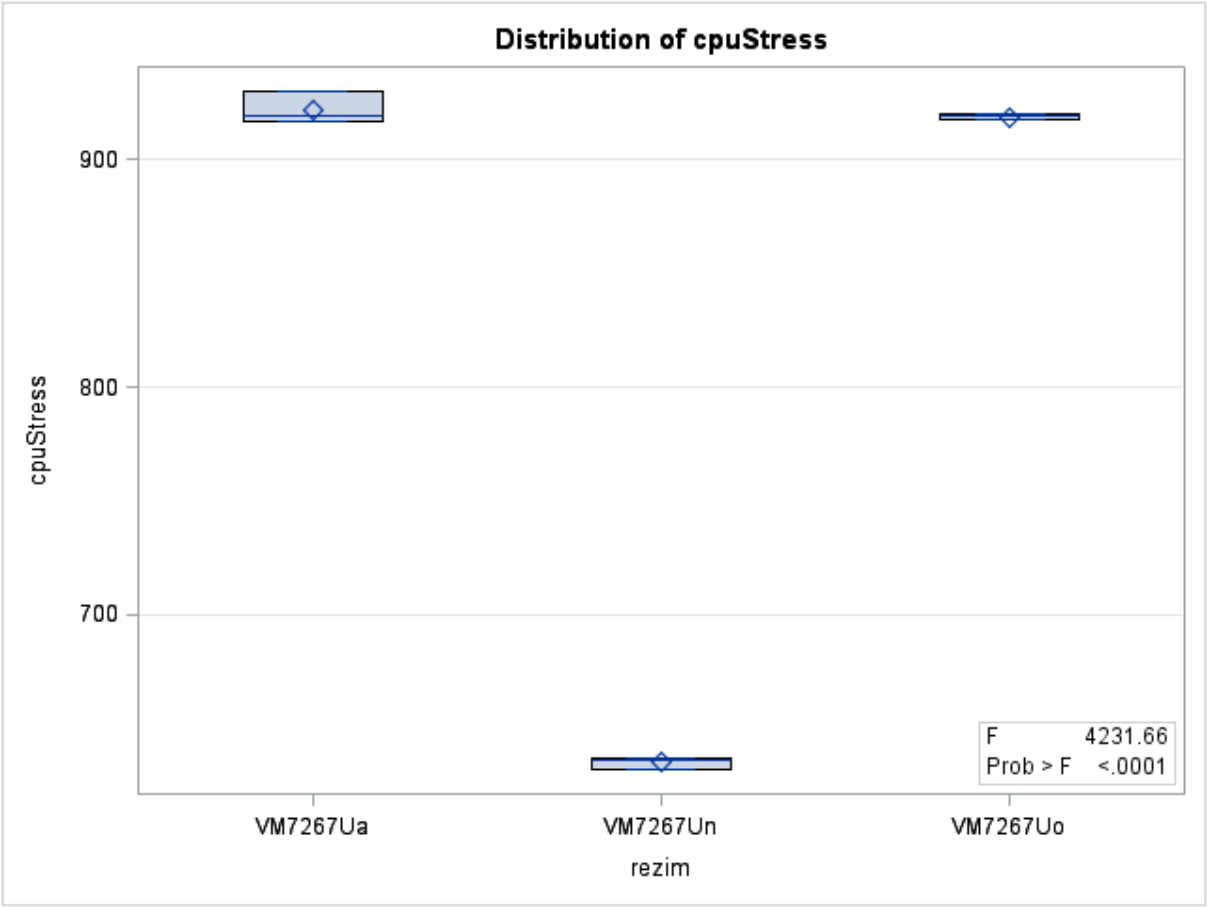
Dependent Variable: cpuStress

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	784795.2249	392397.6124	8843.76	<.0001
Error	6	266.2201	44.3700		
Corrected Total	8	785061.4450			

Obrázek č. 43 – Procedura GLM pro virtualizovanou část testu Stress CPU i7-4790K [autor]

Levene's Test for Homogeneity of cpuStress Variance ANOVA of Squared Deviations from Group Means					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	13003.8	6501.9	3.74	0.0883
Error	6	10439.3	1739.9		

Obrázek č. 44 – Leveneův test pro virtualizovanou část testu Stress CPU i7-4790K [autor]



Obrázek č. 45 – Procedura Boxlpot pro virtualizovanou část testu Stress CPU i5-7267U [autor]

The GLM Procedure Dependent Variable: cpuStress					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	163030.8294	81515.4147	4231.66	<.0001
Error	6	115.5794	19.2632		
Corrected Total	8	163146.4088			

Obrázek č. 46 – Procedura GLM pro virtualizovanou část testu Stress CPU i5-7267U [autor]

Levene's Test for Homogeneity of cpuStress Variance ANOVA of Squared Deviations from Group Means					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	1825.6	912.8	3.31	0.1075
Error	6	1654.9	275.8		

Obrázek č. 47 – Leveneův test pro virtualizovanou část testu Stress CPU i5-7267U [autor]

8.3 Příloha C – Statistická testování Stress-NG – Context Switching

Konf.	Context Switching			
CPU	**i7-740Q**			
Třída	**auto**	**Spe=on**	**Mel=on**	**off**
Pozorované hodnoty	485271,46	540586,07	492184,24	569784,24
	494211,20	538247,48	530908,12	564567,78
	518978,19	531573,98	527703,91	555200,16
	478845,19		533297,97	
	502438,61		502533,67	
	506155,10		532856,11	
			538185,19	
			549056,32	
			519922,35	
			499366,82	
			530294,01	
			551747,57	
			552482,82	
			512271,10	
			529426,39	
Rozsah	6	3	15	3
CPU	**i7-4790K**			
Třída	**auto**	**Spe=on**	**Mel=on**	**off**
Pozorované hodnoty	1444464,36	2080539,37	2547860,59	1857558,69
	1446395,42	3341808,68	3285532,76	2661293,93
	1460512,93	3611710,73	3949001,74	3109962,63
		2291259,89	2620683,98	2675475,54
		2546202,50	3604700,27	2491374,93
		2967163,01	3368131,31	3479149,81
		2622751,96	2848473,06	2264562,68
		3075889,12	2600880,59	2456626,95
		2900658,41	2822099,43	3923003,68
		3751814,38	3338929,06	3010033,78
		2598731,50	3270662,49	2321147,94
		3155853,32	3487096,01	3522289,54
		2405227,87	3482958,30	2668230,58

		2718992,26	3359446,82	3834714,46
		2604020,52	2895072,56	3542887,27
Rozsah	3	15	15	15
CPU	**i5-7267U**			
Třída	**auto**	**Spe=on**	**Mel=on**	**off**
Pozorované hodnoty	602788,38	785757,01	775098,02	826378,34
	599322,86	780091,66	754540,97	806351,26
	597457,04	778777,66	766144,20	809451,22
Rozsah	3	3	3	3

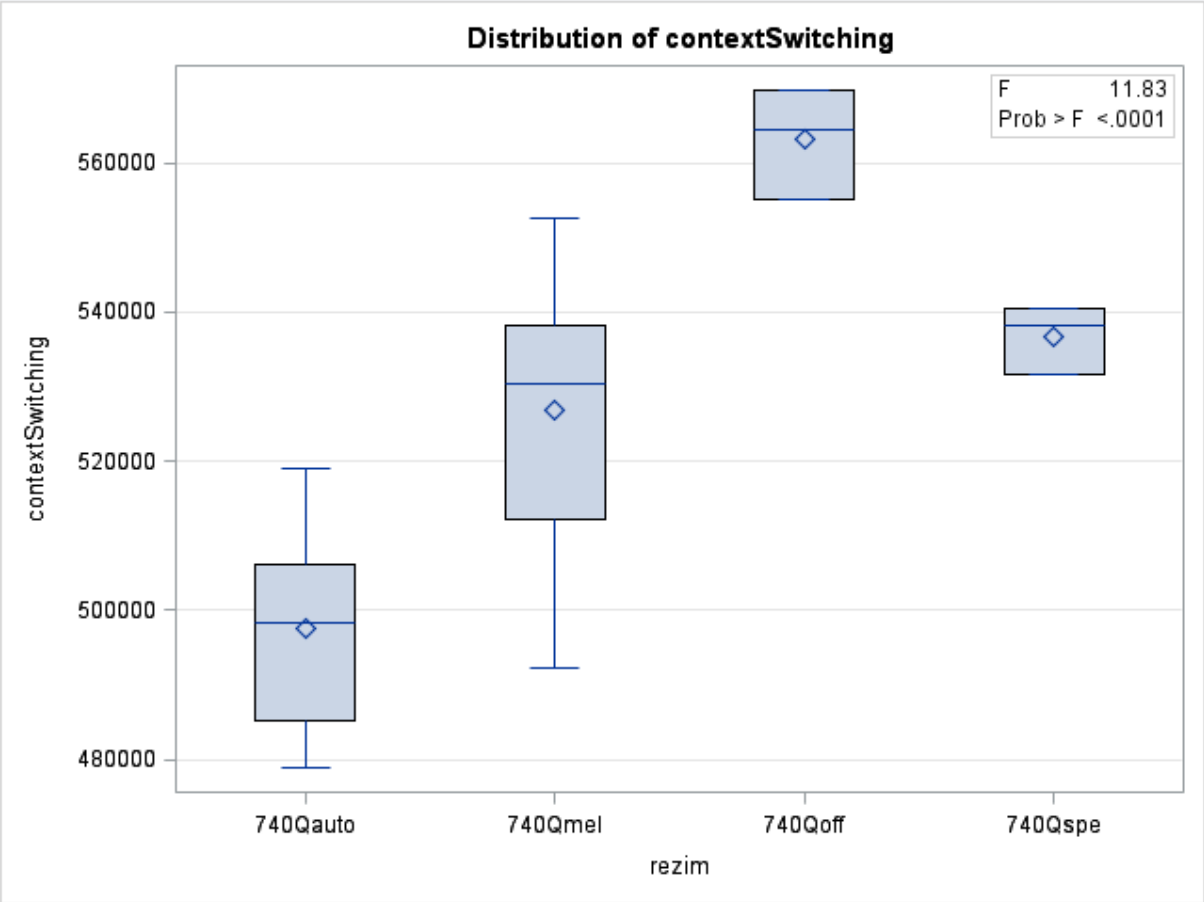
Tabulka č. 37 – Pokusný plán pro fyzickou část testu Context Switching [autor]

The GLM Procedure

Dependent Variable: contextSwitching

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	9323217078	3107739026	11.83	<.0001
Error	23	6042521074	262718308		
Corrected Total	26	15365738152			

Obrázek č. 48 – Procedura GLM pro fyzickou část testu Context Switching i7-740Q [autor]



Obrázek č. 49 – Procedura Boxlpot pro fyzickou část testu Context Switching i7-740Q [autor]

Levene's Test for Homogeneity of contextSwitching Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	3.92E17	1.307E17	1.42	0.2613
Error	23	2.11E18	9.174E16		

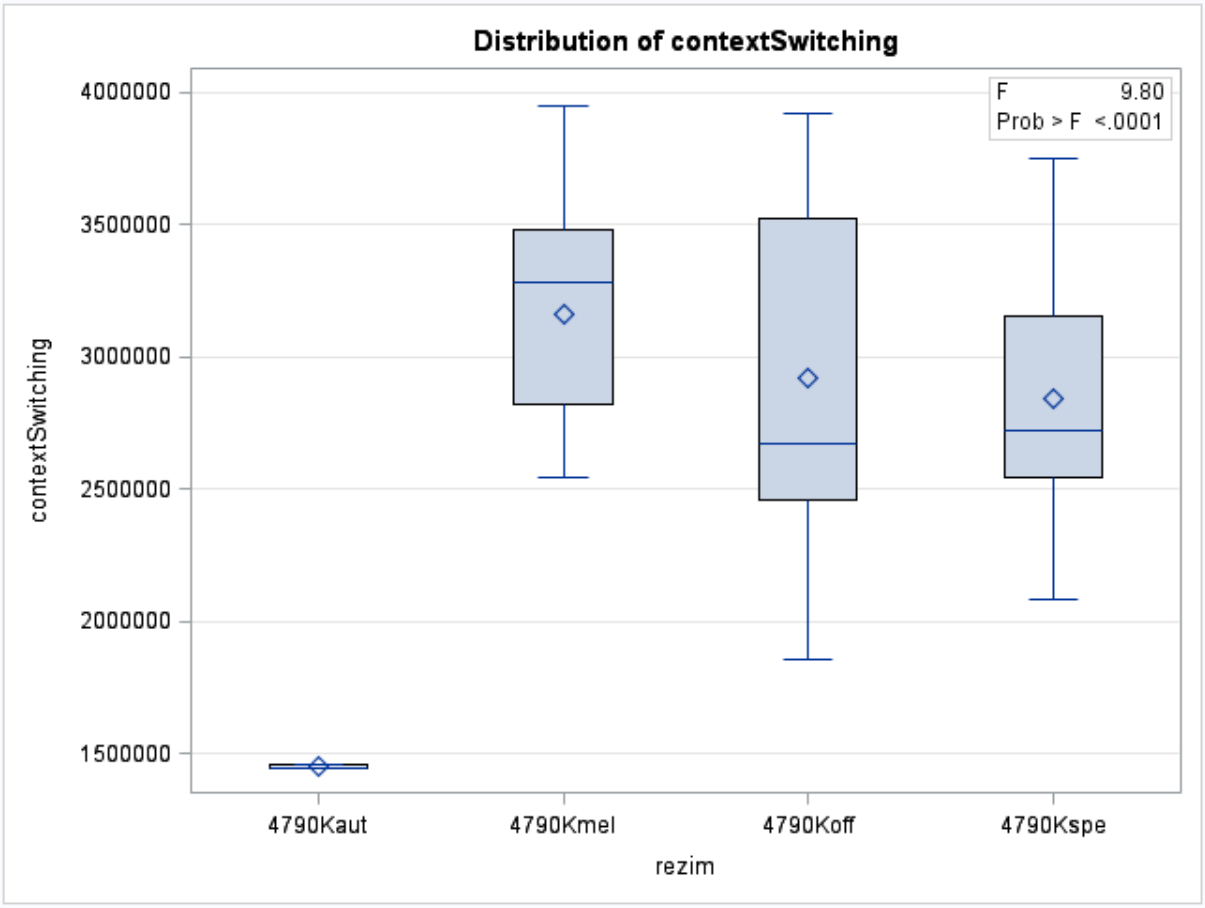
Obrázek č. 50 – Leveneův test pro fyzickou část testu Context Switching i7-740Q [autor]

The GLM Procedure

Dependent Variable: contextSwitching

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	7.3967567E12	2.4655856E12	9.80	<.0001
Error	44	1.1072469E13	251647013063		
Corrected Total	47	1.8469225E13			

Obrázek č. 51 – Procedura GLM pro fyzickou část testu Context Switching i7-4790K [autor]



Obrázek č. 52 – Procedura Boxlpot pro fyzickou část testu Context Switching i7-4790K [autor]

Levene's Test for Homogeneity of contextSwitching Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	5.004E23	1.668E23	2.36	0.0841
Error	44	3.106E24	7.058E22		

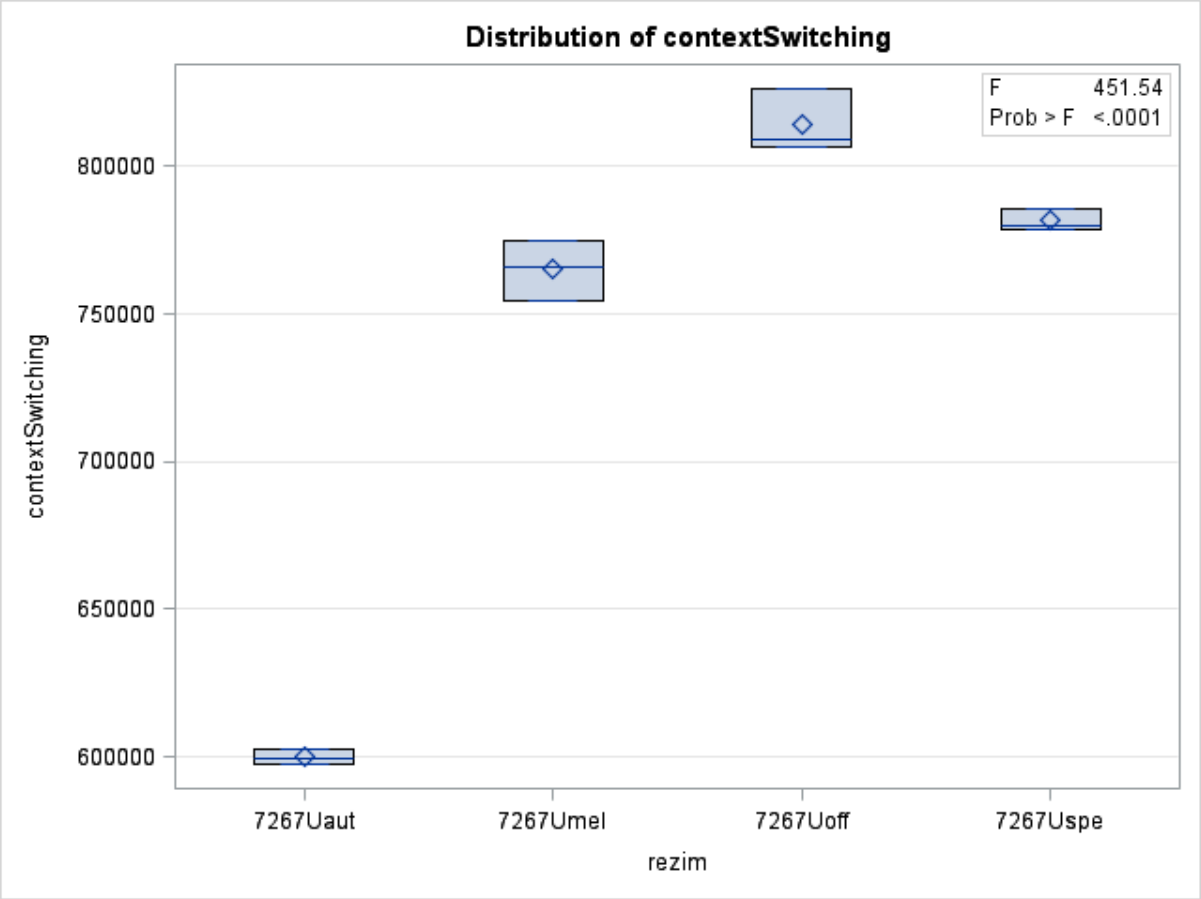
Obrázek č. 53 – Leveneův test pro fyzickou část testu Context Switching i7-4790K [autor]

The GLM Procedure

Dependent Variable: contextSwitching

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	82466932355	27488977452	451.54	<.0001
Error	8	487022544	60877818		
Corrected Total	11	82953954899			

Obrázek č. 54 – Procedura GLM pro fyzickou část testu Context Switching i5-7267U [autor]



Obrázek č. 55 – Procedura Boxlpot pro fyzickou část testu Context Switching i5-7267U [autor]

Levene's Test for Homogeneity of contextSwitching Variance
ANOVA of Squared Deviations from Group Means

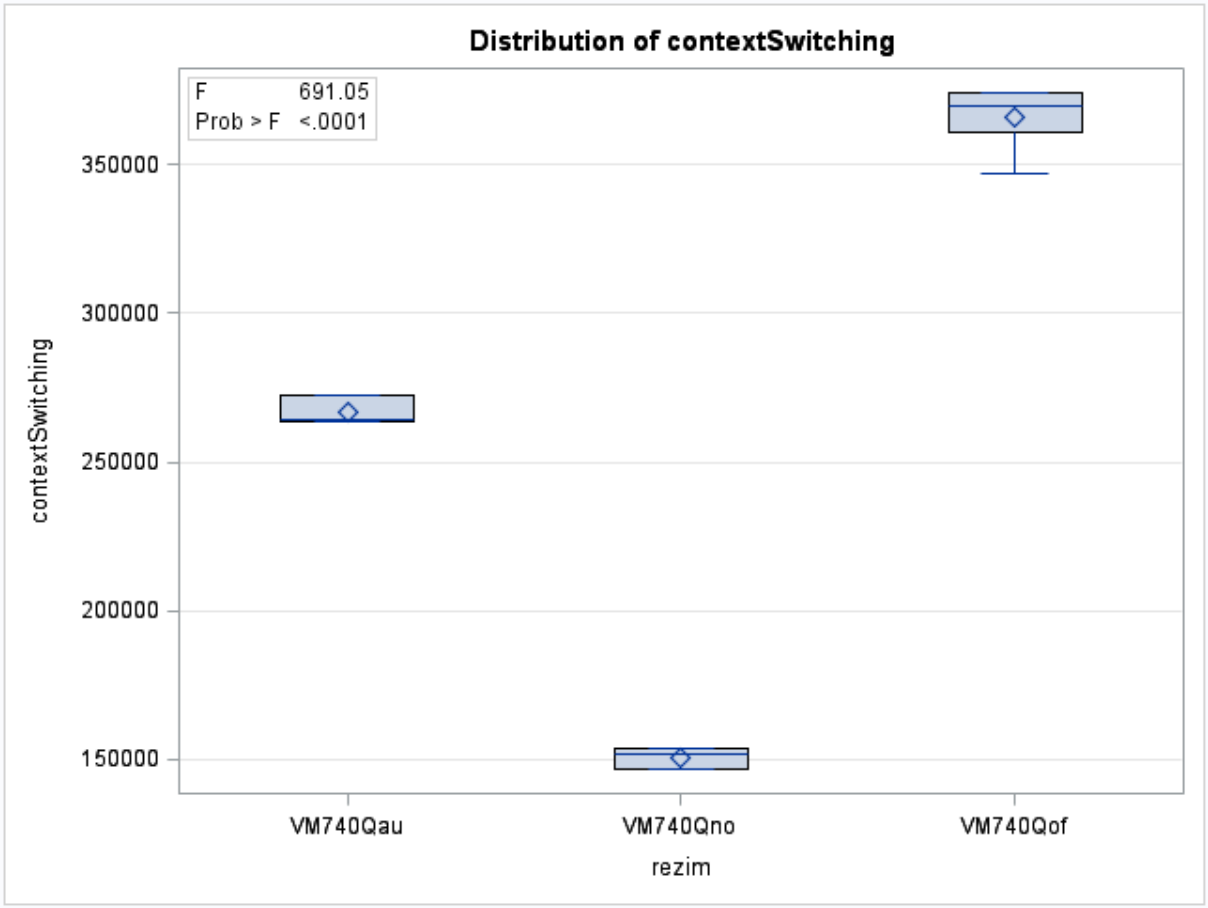
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	3	1.361E16	4.536E15	2.17	0.1689
Error	8	1.669E16	2.086E15		

Obrázek č. 56 – Leveneův test pro fyzickou část testu Context Switching i5-7267U [autor]

Konf.	**Context Switching**		
CPU	**i7-740Q**		
Třída	**auto**	**nosmt**	**off**
Pozorované hodnoty	272403,16	153457,64	347234,62
	263368,52	146651,59	374297,97
	264158,16	151881,92	374207,99
			370581,42
			361210,81
			368756,64
Rozsah	3	3	6
CPU	**i7-4790K**		
Třída	**auto**	**nosmt**	**off**

Pozorovaná hodnoty	901428,59	751136,34	1575157,55
	923655,00	748343,01	1594736,74
	929421,96	742113,06	1576900,32
Rozsah	3	3	3
CPU	**i5-7267U**		
Třída	**auto**	**nosmt**	**off**
Pozorované hodnoty	403039,65	315844,04	757513,85
	405952,78	312775,13	754330,11
	409461,92	311854,08	754054,38
Rozsah	3	3	3

Tabulka č. 38 – Pokusný plán pro virtualizovanou část testu Context Switching [autor]



Obrázek č. 57 – Procedura Boxlpot pro virtualizovanou část testu Context Switching i7-740Q [autor]

The GLM Procedure

Dependent Variable: contextSwitching

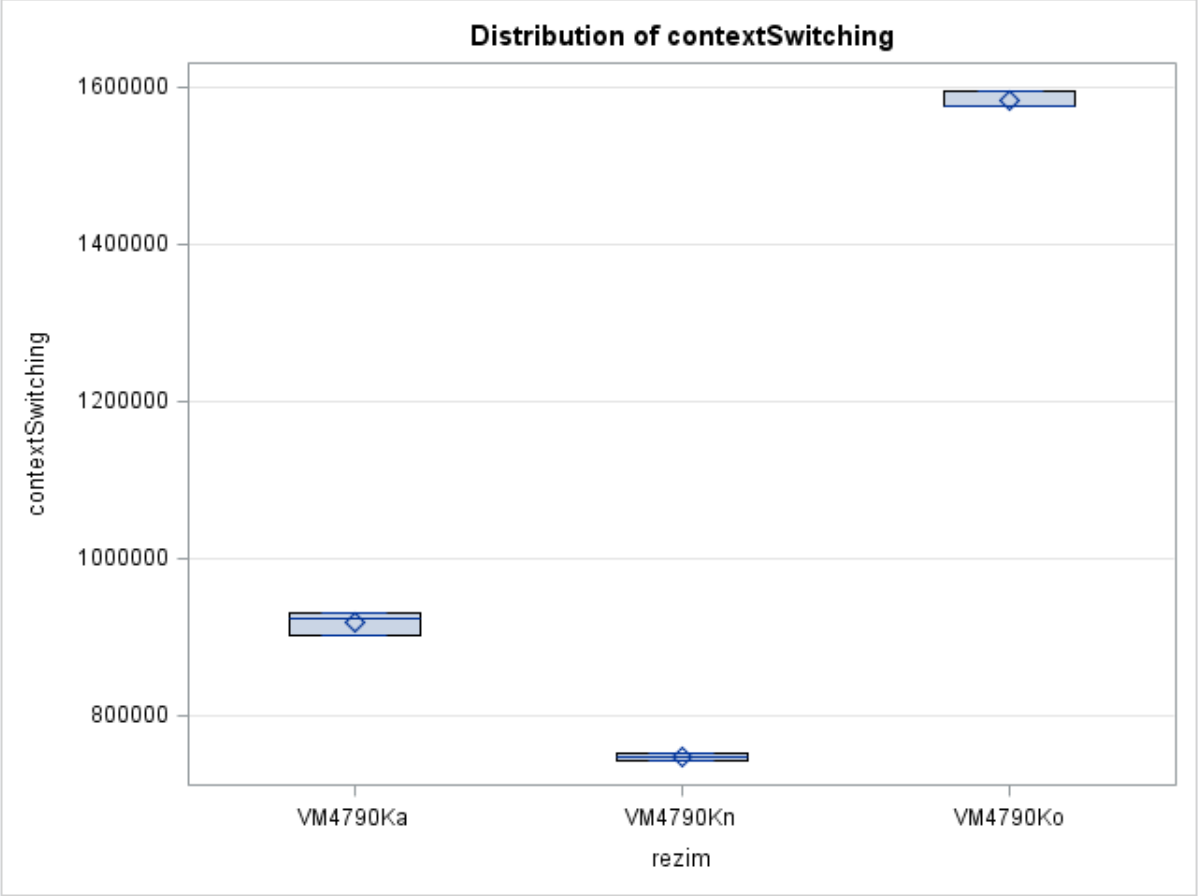
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	94496204353	47248102177	691.05	<.0001
Error	9	615341023	68371225		
Corrected Total	11	95111545376			

Obrázek č. 58 – Procedura GLM pro virtualizovanou část testu Context Switching i7-740Q [autor]

Levene's Test for Homogeneity of contextSwitching Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	1.807E16	9.037E15	0.93	0.4289
Error	9	8.732E16	9.702E15		

Obrázek č. 59 – Leveneův test pro virtualizovanou část testu Context Switching i7-740Q [autor]



Obrázek č. 60 – Procedura Boxlpot pro virtualizovanou část testu Context Switching i7-4790K [autor]

The GLM Procedure

Dependent Variable: contextSwitching

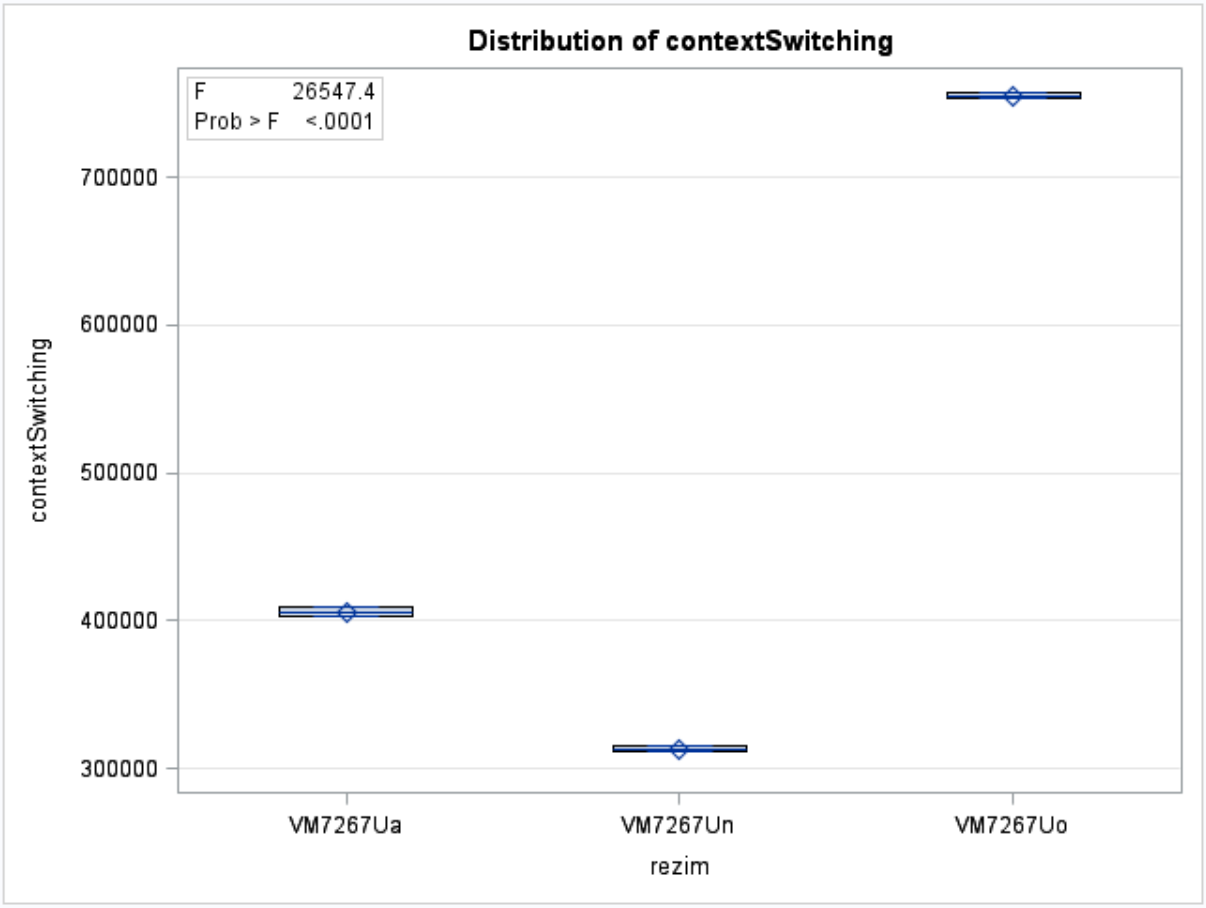
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	1.1675926E12	583796313926	4902.52	<.0001
Error	6	714484750.43	119080791.74		
Corrected Total	8	1.1683071E12			

Obrázek č. 61 – Procedura GLM pro virtualizovanou část testu Context Switching i7-4790K [autor]

Levene's Test for Homogeneity of contextSwitching Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	2.592E16	1.296E16	1.88	0.2321
Error	6	4.132E16	6.886E15		

Obrázek č. 62 – Leveneův test pro virtualizovanou část testu Context Switching i7-4790K [autor]



Obrázek č. 63 – Procedura Boxlpot pro virtualizovanou část testu Context Switching i5-7267U [autor]

The GLM Procedure

Dependent Variable: contextSwitching

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	325684897038	162842448519	26547.4	<.0001
Error	6	36804141.494	6134023.5824		
Corrected Total	8	325721701180			

Obrázek č. 64 – Procedura GLM pro virtualizovanou část testu Context Switching i5-7267U [autor]

Levene's Test for Homogeneity of contextSwitching Variance
ANOVA of Squared Deviations from Group Means

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
rezim	2	3.569E13	1.785E13	1.15	0.3777
Error	6	9.31E13	1.552E13		

Obrázek č. 65 – Leveneův test pro virtualizovanou část testu Context Switching i5-7267U [autor]

8.4 Příloha D – Přehled změn výkonu procesorů při zapnutých mitigacích

		i7-740Q				
Část testování		Fyzická			Virtualizovaná	
Název testu	Název testování	auto	Spe=on	Mel=on	auto	auto,nosmt
Flexible IO Tester	Random Read	-8,8%	1,3%	-6,2%	-3,0%	-2,6%
	Random Read	-8,6%	1,9%	-6,0%	-3,1%	-2,6%
	Random Write	-20,5%	1,0%	-13,3%	-37,8%	-36,3%
	Random Write	-20,2%	1,2%	-13,0%	-37,6%	-36,1%
	Sequential Read	-7,9%	-0,4%	-5,6%	-25,8%	-13,1%
	Sequential Read	-8,1%	-0,7%	-5,6%	-14,5%	-14,9%
	Sequential Write	-14,9%	-6,4%	-8,9%	-14,9%	-18,3%
	Sequential Write	-8,8%	0,6%	-5,7%	-14,6%	-18,0%
Compile Bench	Compile	3,0%	0,3%	1,2%	-3,3%	5,6%
	Initial Create	-12,1%	-1,0%	-11,2%	-14,0%	-13,4%
	Read Compiled Tree	-22,4%	-0,7%	-21,7%	-35,3%	-39,1%
Postmark - Disk Transaction Performance		-17,4%	-4,9%	-13,4%	-26,2%	-26,5%
TTSIOD 3D Renderer - Phong Rendering WithSoft-Shadow Mapping		-0,6%	-0,3%	-0,2%	0,5%	-46,4%
Timed Linux Kernel Compilation - Time To Compile		-3,5%	-0,2%	-2,8%	-9,4%	-103,8%
Hackbench	4 Thread	-20,1%	-3,3%	-16,5%	-19,3%	-91,1%
	8 Thread	-21,0%	-0,2%	-19,9%	-17,5%	-58,2%
	16 Thread	-32,5%	-0,2%	-36,8%	-21,7%	-45,2%
	4 Process	-19,2%	-1,8%	-17,5%	-16,2%	-51,3%
	8 Process	-23,8%	-1,7%	-19,1%	-10,2%	-39,4%
	16 Process	-23,2%	-2,5%	-22,4%	-35,4%	-63,0%
OpenSSL - RSA 4096-bit Performance		-1,0%	-0,6%	-0,2%	0,6%	-40,6%
Glibc Bench	ffs	-10,7%	-1,1%	-8,9%	-24,6%	-23,4%
	sqrt	-4,9%	-0,6%	-2,8%	-3,4%	-3,9%
	ffsll	-11,1%	-1,0%	-9,0%	-23,8%	-23,4%
	pthread_once	-11,1%	-1,3%	-9,3%	-22,0%	-21,1%
PostgreSQL pgbench Scaling Buffer Test	Normal Load Read Only	-9,0%	-0,8%	-7,9%	-12,0%	-56,9%
	Normal Load Read Write	-5,3%	-0,3%	-4,4%	-9,8%	-56,6%
	Single Thread Read Only	-3,9%	-1,9%	-3,6%	-14,3%	-48,5%
	Single Thread Read Write	-4,9%	-1,6%	-3,7%	-4,5%	-27,3%
	Heavy Contention Read Only	-6,5%	-0,6%	-6,5%	-15,8%	-60,0%
	Heavy Contention Read Write	-3,8%	0,1%	-3,7%	-8,8%	-51,8%
Redis	LPOP	-39,2%	-2,8%	-37,7%	-68,5%	-69,3%
	SADD	-39,4%	-3,4%	-38,0%	-67,2%	-68,0%
	GET	-39,7%	-3,7%	-37,5%	-69,6%	-70,4%

Stress-NG	Crypto	0,1%	0,3%	0,1%	-0,1%	-36,6%
	CPU Stress	-0,8%	1,1%	0,5%	0,0%	-50,8%
	Socket Activity	-19,6%	-3,8%	-15,7%	-40,1%	-67,3%
	Context Switching	-13,9%	-4,7%	-6,5%	-27,2%	-58,8%
NGINX Benchmark - Static Web Page Serving		-15,0%	-2,7%	-11,1%	-26,4%	-24,7%
Apage Benchmark - Static Web Page Serving		-13,9%	-2,6%	-11,8%	-21,1%	-20,9%

Tabulka č. 39 – Přehled změn výkonu procesorů při zapnutých mitigacích i7-740Q [autor]

		i7-4790K				
Část testování		**Fyzická**			**Virtualizovaná**	
Název testu	**Název testování**	**auto**	**Spe=on**	**Mel=on**	**auto**	**auto,nosmt**
Flexible IO Tester	Random Read	-1,2%	-24,2%	-23,0%	11,9%	-4,5%
	Random Read	-0,3%	-23,5%	-23,1%	10,7%	-5,8%
	Random Write	-7,4%	1,1%	1,6%	-16,3%	-12,8%
	Random Write	-7,4%	1,0%	1,5%	-16,4%	-13,0%
	Sequential Read	-3,0%	-0,8%	-1,4%	-5,1%	-3,3%
	Sequential Read	-3,1%	-0,8%	-1,5%	-5,0%	-3,4%
	Sequential Write	0,5%	-2,4%	-1,0%	-9,4%	-14,9%
	Sequential Write	0,4%	-2,2%	-1,2%	-9,4%	-15,1%
Compile Bench	Compile	-7,3%	-6,7%	-3,5%	-10,9%	-8,2%
	Initial Create	-9,5%	-0,6%	-3,8%	-22,7%	-22,1%
	Read Compiled Tree	-20,7%	-5,7%	-10,5%	-27,7%	-66,2%
Postmark - Disk Transaction Performance		-20,6%	-10,3%	-9,4%	-38,3%	-38,0%
TTSIOD 3D Renderer - Phong Rendering WithSoft-Shadow Mapping		0,9%	0,6%	1,2%	-4,6%	-22,2%
Timed Linux Kernel Compilation - Time To Compile		-3,0%	-1,5%	-1,2%	-15,7%	-39,4%
Hackbench	4 Thread	-10,1%	22,4%	13,2%	-54,0%	-90,5%
	8 Thread	-40,8%	3,5%	-1,9%	-67,9%	-116,5%
	16 Thread	-61,6%	-2,6%	-11,7%	-141,1%	-181,8%
	4 Process	-45,6%	4,2%	-1,7%	-58,0%	-98,6%
	8 Process	-56,2%	1,3%	-9,2%	-109,1%	-150,9%
	16 Process	-73,6%	-14,9%	-20,4%	-151,7%	-227,5%
OpenSSL - RSA 4096-bit Performance		0,1%	0,0%	0,1%	-0,1%	-1,8%
Glibc Bench	ffs	-19,1%	-1,7%	-7,4%	-30,4%	-29,3%
	sqrt	-9,7%	-0,8%	-3,7%	-16,0%	-15,6%
	ffsll	-19,1%	-1,5%	-7,3%	-33,8%	-29,9%
	pthread_once	-17,9%	-0,6%	-6,0%	-30,8%	-29,1%
PostgreSQL pgbench Scaling Buffer Test	Normal Load Read Only	-9,2%	-1,3%	-2,5%	-25,7%	-47,3%
	Normal Load Read Write	2,4%	9,5%	4,0%	-17,6%	-34,9%

	Single Thread Read Only	-6,5%	-2,3%	-2,7%	-24,1%	-35,0%
	Single Thread Read Write	-2,3%	3,1%	-6,4%	-11,9%	-15,5%
	Heavy Contention Read Only	-7,2%	-0,6%	-2,0%	-22,8%	-45,1%
	Heavy Contention Read Write	-1,0%	0,6%	-2,0%	-18,1%	-39,2%
Redis	LPOP	-6,6%	-2,6%	-3,2%	-21,0%	-24,0%
	SADD	-2,1%	-0,2%	3,2%	-17,2%	-16,7%
	GET	-8,8%	-4,2%	-3,5%	-23,7%	-22,5%
Stress-NG	Crypto	-0,1%	0,01%	-0,1%	-0,3%	-3,5%
	CPU Stress	1,5%	0,7%	2,0%	-0,4%	-32,1%
	Socket Activity	-25,6%	-3,7%	-5,7%	-44,6%	-56,0%
	Context Switching	-50,3%	-2,6%	8,4%	-42,0%	-52,8%
NGINX Benchmark - Static Web Page Serving		-20,3%	-8,3%	-7,0%	-40,2%	-38,2%
Apage Benchmark - Static Web Page Serving		-17,0%	-5,5%	-3,3%	-39,2%	-4,6%

Tabulka č. 40 – Přehled změn výkonu procesorů při zapnutých mitigacích i7-4790K [autor]

		i5-7267U				
Část testování		**Fyzická**			**Virtualizovaná**	
Název testu	**Název testování**	**auto**	**Spe=on**	**Mel=on**	**auto**	**auto,nosmt**
Flexible IO Tester	Random Read	20,8%	1,3%	8,7%	6,5%	-2,6%
	Random Read	21,0%	1,4%	8,7%	6,0%	-3,1%
	Random Write	-8,5%	4,0%	2,1%	-14,5%	-9,5%
	Random Write	-8,6%	4,2%	2,0%	-14,7%	-9,6%
	Sequential Read	-4,0%	-0,5%	-1,3%	-2,4%	-1,3%
	Sequential Read	-3,9%	-0,3%	-1,4%	-1,4%	-0,8%
	Sequential Write	-6,6%	-2,1%	-5,3%	-8,2%	-12,6%
	Sequential Write	-6,5%	-2,0%	-5,2%	-8,1%	-12,5%
Compile Bench	Compile	-6,3%	-3,3%	-0,4%	-2,4%	1,2%
	Initial Create	-11,2%	-7,4%	-3,9%	-14,6%	-12,4%
	Read Compiled Tree	11,1%	2,2%	7,7%	-50,8%	-48,6%
Postmark - Disk Transaction Performance		-32,2%	-18,7%	-10,8%	-44,3%	-44,4%
TTSIOD 3D Renderer - Phong Rendering WithSoft-Shadow Mapping		-2,0%	2,4%	0,1%	-4,1%	-22,9%
Timed Linux Kernel Compilation - Time To Compile		-7,3%	1,2%	0,6%	-18,9%	-49,5%
Hackbench	4 Thread	-14,8%	6,8%	2,9%	-97,8%	-156,6%
	8 Thread	-33,3%	12,7%	0,2%	-139,0%	-158,1%
	16 Thread	-44,6%	9,9%	-0,6%	-180,6%	-199,2%
	4 Process	-28,8%	24,7%	13,1%	-126,9%	-177,5%
	8 Process	-64,7%	1,8%	-8,2%	-161,9%	-210,4%
	16 Process	-92,3%	-20,1%	-30,6%	-169,6%	-201,0%

OpenSSL - RSA 4096-bit Performance		-1,2%	0,6%	-0,5%	0,2%	-4,9%
Glibc Bench	ffs	-18,9%	-1,2%	-7,4%	-38,0%	-38,3%
	sqrt	-16,6%	-1,2%	-6,6%	-34,3%	-34,1%
	ffsll	-19,0%	-1,3%	-7,4%	-37,4%	-38,1%
	pthread_once	-19,3%	-1,7%	-8,0%	-37,8%	-38,0%
PostgreSQL pgbench Scaling Buffer Test	Normal Load Read Only	-13,3%	-2,4%	-3,7%	-30,6%	-43,8%
	Normal Load Read Write	-7,2%	0,7%	-0,5%	21,8%	-34,5%
	Single Thread Read Only	-9,8%	-1,3%	-3,6%	-28,1%	-33,5%
	Single Thread Read Write	7,1%	-0,1%	-2,4%	-13,5%	-13,0%
	Heavy Contention Read Only	-15,0%	-2,9%	-4,1%	-27,7%	-48,1%
	Heavy Contention Read Write	-11,2%	-0,7%	-3,2%	0,8%	-36,8%
Redis	LPOP	-8,5%	0,4%	-0,7%	-21,2%	-20,9%
	SADD	-7,6%	-6,0%	2,1%	-20,2%	-16,6%
	GET	-14,1%	-9,8%	-12,2%	-22,8%	-21,0%
Stress-NG	Crypto	-3,7%	0,5%	-0,4%	0,0%	-2,8%
	CPU Stress	1,8%	-0,1%	-0,5%	0,4%	-30,9%
	Socket Activity	-35,8%	-9,9%	-11,3%	-49,5%	-61,0%
	Context Switching	-26,3%	-4,0%	-6,0%	-46,2%	-58,5%
NGINX Benchmark - Static Web Page Serving		-26,8%	-10,0%	-8,2%	-43,0%	-39,3%
Apage Benchmark - Static Web Page Serving		-20,9%	-7,8%	-5,7%	-44,6%	-57,2%

Tabulka č. 41 – Přehled změn výkonu procesorů při zapnutých mitigacích i5-7267U [autor]