TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MARIIA KOMAR

ASSESSMENT OF DATA RATES ON THE INTERNAL AND EXTERNAL CPU INTERFACES AND ITS APPLICATIONS FOR WIRELESS NETWORK-ON-CHIP DEVELOPMENT

Master of Science thesis

Examiner: Prof. Yevgeni Koucheryavy,
Dr. Dmitri Moltchanov
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 27th September 2017

ABSTRACT

MARIIA KOMAR: Assessment of data rates on the internal and external CPU interfaces and its applications for Wireless Network-on-Chip development
Tampere University of Technology
Master of Science thesis, 56 pages
November 2017
Master's Degree Programme in Information Technology
Major: Information Technology
Examiner: Prof. Yevgeni Koucheryavy, Dr. Dmitri Moltchanov
Keywords: central processing unit, CPU, CPU performance, Wireless Networks-on-Chip, WNoC, data rates assessment

Nowadays central processing units (CPUs) are the major part of the personal computers, and usually their progress defines personal computers (PCs) progress. However, modern CPU architecture has a set of limitations mentioned in this thesis. As a result, new CPU architectures are now under development. Most prospective solution in this field are based on a proposed concept of Wireless Networks-on-Chip (WNoCs), where part of wired connections is changed into wireless links. However in order to design and develop this kind of system, information about data rates on the internal and external CPU interfaces of modern CPUs is needed. Main goals set in the beginning of working on this thesis were to get this data rates assessment and give an assessment of suitable wireless technologies for milticore CPUs with different number of cores.

In this thesis CPU evolution is described and peculiarities of modern CPU architectures are mentioned. Besides state-of-the-art overview for Wireless Networks-on-Chip is provided. Moreover, full methodology of measuring intra-CPU counters and getting data rates on cache bus between second and third level caches and third level cache and random access memory (RAM) controller bus are provided. Dependencies of data rates on interfaces of interest on the number of active CPU cores and CPU clock frequency are studied and provided in a form of plots. Also differences in the traffic for different types of CPU load are provided as bar diagrams. For testing we used several real-life tasks that are typical for CPUs and artificial tests which are represented as programs written in C programming language. In addition, extrapolation model for CPUs with bigger amount of cores is provided and assumption about suitable wireless technologies for different number of CPU cores is made.

PREFACE

This work has been done in the faculty of Computing and Electrical Engineering and especially in the Nano Communications Center at Tampere University of Technology.
I would like to thank Prof. Yevgeni Koucheryavy and Dr. Dmitri Moltchanov for giving me an opportunity to work in such a great place and under the guidance of such great people. Their help, endless support, ideas, suggestions, corrections and belief in me helped me a lot not only in my current research, but also in all my life tasks and searching for future research directions.
I would like to thank Vitaly Petrov, Pavel Boronin and Anna Volkova for support, criticizing of my ideas, great questions, help in both research and personal tasks. I hope these connections wouldn't cease with my defense and relocation to another country.
I would like to thank Alexander Pyattaev for teaching me how to think and solve problems that look unsolvable. I would like to thank Roman Florea for his patience and tact in explaining practical networking issues and answering my silly questions.
I would like to thank Alexey Ponomarenko-Timofeev for his help, expertise, opening of new horizons and support in hard times.
Certainly, I would thank all my colleagues with whom I was lucky to work for great company, work and non work conversations and useful advice.
I would like to thank V. Sokolov and E. Sokolova for their help with my previous studies and PhD opportunities, wise advice and guidance.
Moreover, I would like to express great gratitude to my beloved husband who was supporting me during all these hard times and made my thesis defense possible. I really appreciate all that he is doing for me, his endless support, believing in my ideas, care, understanding, patience and giving me an ability to be who I am.
Besides, I would like to thank my parents, family and friends for their care and support during my whole life. Without them I can't reach any of my achievements.
Additionally, I would like to thank Coreboot and Grub projects communities as they helped me a lot in my research giving advice and pointing me to the desired directions in very technical part of research.
I dedicate this Thesis to my parents who did everything possible (and sometimes even impossible) to give me a good start in my life.

Zurich - Tampere, 07.11.2017

Maria (Mariia) Komar

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

2D	3-dimensional
3D	3-dimensional
aarch64	ARM Architecture 64
AES	Advanced Encryption Standard
ALU	Arithmetic logic unit
AMD	Advanced Micro Devices
Apple IIgs	Apple II-graphics, sound
ARM	Advanced RISC Machine
BIOS	Basic Input/Output System
BUNCH	Burroughs, UNIVAC, NCR, Control Data Corporation, and Honeywell
CISC	Complex Instruction Set Computing
CPU	Central Processing Unit
CVD	ConVert to Decimal
DDR4	Double Data Rate 4th-generation
DEC	Digital Equipment Corporation
ENIAC	Electronic Numerical Integrator and Computer
GB	Gigabyte
Gbps	Gigabit per second
GHz	Gigahertz
GPU	Graphics Processing Unit
HDD	Hard Disk Drive
IBM	International Business Machines
IC	Integrated Circuit
I/O	Input/Output
ISA	Instruction Set Architecture
KHz	Kilohertz
KiB	Kibibyte
L1	Level 1
L2	Level 2
L3	Level 3
LED	Light-Emitting Diode
MAC	Medium Access Control
MB	Megabyte
Mbps	Megabit per second
MHz	Megahertz

MMU	Memory Management Unit
mmWaves	Millimeter waves
MOS	Metal Oxide Semiconductor
NCR	National Cash Register
NoC	Network-on-Chip
OOK	On-Off Keying
OpenSSL	Open Secure Sockets Layer
OS	Operating System
PC	Personal computer
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PCM	Performance Counter Monitor
RAM	Random Access Memory
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
SIMD	Single Instruction, Multiple Data
SoC	System-on-Chip
SSD	Solid-State Drive
THz	Terahertz
TRS	Tandy/Radio Shack
UNIVAC	Universal Automatic Computer
US	United States
USD	United States Dollar
WNoC	Wireless Network-on-Chip
Z3	Zuse 3

S_{L2-L3}	data rate on interface between second and third level cache [Gbps]
$L2_{miss,av}$	average number of second level cache misses [millions]
S_{L3-RAM}	data rate on interface between third level cache and random access memory controller [Gbps]
$L3_{miss,av}$	average number of third level cache misses [millions]

1. INTRODUCTION

In this chapter state-of-the-art overview of central processing unit is provided and current issues in CPU development are mentioned. Besides, thesis outline and full description of solvable problems are provided.

1.1 Central processing units state-of-the-art overview

Looking back to the computer science history we can note that progress in computers development was really significant in 80s and 90s[18]. However during last two decades it is not so rapid as users would like to have. Main reason for this fact is a limitations connected with performance of central processing units (CPUs). Their development was so fast that every 18 month productivity has doubled[18]. It basically determined the progress of computers of that era in general. After several decades the situation has changed. Looking on the dynamics of development of different CPU characteristics, such as clock speed, power or number of transistors, we can notice, that during 80-s and 90-s their progress was advancing at constant pace[10].
However from the beginning of 00-s, both clock speed and power are bounded and the only thing that still increases is the number of transistors[10]. Due to this limitations multi-core CPU concept appeared[10]. It gave a significant gain for CPU development, but overall progress was worse than expected[10]. After the introduction of multi-core CPUs the only significant changes were increasing the number of cache levels from 2 to 3 on most CPUs and increase of a cache size[6]. However, it does not give the desired effect and currently every new generation of CPUs gives an improvement of 15 percent at most[30]. Nowadays most typical desktop CPU has several cores and several cache levels with hierarchical architecture[31]. Number of cores in the CPU increases not so rapidly, so first CPU with two cores made by Intel appeared in 2005[32], and nowadays most popular CPUs called as "bestseller" on many computer shops has just 4 cores.
Now new ways of increasing CPU performance are being explored. Most promising concepts for future CPU development are Network-on-Chips (NoCs) and especially Wireless Network-on-Chips (WNoCs) architectures[60], [66], [48], [23], [46]. They

combine advantages of CPU concept and networking advances in modern technologies. Main idea of both concepts is to make a network of computing cores with a routing scheme using wired and/or wireless links. There are many proposals of architectures and routing schemes exist, but for applicability assessment of all proposed solutions it is necessary to take into account real life CPUs peculiarities and especially data rates on the internal and external CPU interfaces to understand buses and/or wireless channels load. Of course usually it is expected that number of cores would be large enough, but to estimate channels load for this amount of cores we should know real load of buses in modern multi-core CPUs and dependencies between number of cores and buses load. Also dependence between clock frequency and buses load as well as comparison of different types of CPU loads might be helpful.

1.2 Problem Definition

As it was mentioned before, in order to assess applicability of proposed NoCs and WNoCs as well as design new CPU architectures it is required to know at least an approximate traffic on the internal and external CPU buses and also be able to extrapolate obtained results for more massive cores CPUs. There are many solutions for CPU traffic estimation received through simulation, but it might be not so precise and covers all CPU features, so real life measurements are required.
It is necessary to choose proper program tool for CPU measurements and propose a measurements methodology. Moreover, it is important to distinguish most typical CPU loads and develop tests based on that analysis. Besides it is important to produce an algorithm of calculation of buses load if the measurements tool doesn't provide data rate in some reasonable unit. Results of measurements should be provided in intuitive and easy-to-perceive form. In addition, it is required to propose an extrapolation of existing results algorithm and explain what kind of wireless communication technology could be used for CPUs with different number of cores.

1.3 Thesis Outline

The main goal of this thesis is to assess data rates on L2-L3 cache interface inside CPU and L3 cache — Random Access Memory (RAM) controller CPU bus, which could be considered as internal CPU bus as well as part of external CPU bus (e.g. Peripheral Component Interconnect (PCI) bus in some CPUs) as these two interfaces are most common candidate for replacement of wired interface onto wireless. So the following tasks within this thesis were posed:

1. Propose and describe in details methodology of real life measurements on modern x86-64 CPU aimed to get assessment of data rates on L2-L3 cache bus and L3—RAM controller bus.

2. Provide results of data rates assessment on the buses of interest and describe dependencies of traffic on number of CPU cores and CPU clock frequency.

3. Compare different CPU loads in terms of data rates on the buses of interest.

4. Provide method for extrapolation of obtained results for bigger number of cores.

5. Analyze obtained extrapolation results and special requirements and propose suitable wireless technology to be used for CPU WNoCs.

The following part of the thesis looks like follows: in Section 3.1 technical stand used for testing described, Section 3.2 is devoted to software tool used for research description as well as specification of different types of CPU load, both real-life and artificial. Precise measurements methodology, formulas for calculation of traffic on buses of interest and software tool designed for measurements processing are described in Section 3.3. Obtained results in the form of plots and bar plots as well as their full description are demonstrated and discussed in Sections 3.4 — 3.5. Final conclusions on the measurements methodology and results are made in Section 3.6. Extrapolation of existing results methodology is discussed in Section 4.1. Review of suitable wireless technologies depending of number of cores in WNoC is provided in Section 4.2. Additional requirements for WNoC development obtained during research are listed in Section 4.3. Final conclusions and future research directions are discussed in Chapter 5

2. CPU DEVELOPMENT AND FUTURE TRENDS

In this chapter concept of central processing unit or CPU is explained, CPU evolution history is detailed and typical architecture of modern CPU with several cores is described. Moreover, current research direction in CPU development are mentioned.

2.1 CPU definition and commonly used notions

Any book that is somehow connected with computer architecture mentions a central processing unit or CPU for short as core element of the computer. The most common definition of CPU is an electronic circuit that carries out instructions provided by computer program using basic operations of logic, arithmetic, control and input/output (I/O for further usage)[43], [24]. This term has been used at least from 1960s. The concept and design of CPU has changed a lot during the history, but the term CPU refers to processor or even more specific part of it that is responsible for data control and processing. Principal scheme of CPU remains pretty much unchanged since the beginning of development in the middle of last century. CPU contains arithmetic logic unit (ALU) aimed to perform logic and arithmetic operations[24]. Also there are processor registers that provide operands for ALU functioning and store results of ALU operations. Also there is a control unit that is responsible for fetching instructions from memory and their execution, coordination operations of ALU registers and all other components[24]. Usually modern CPUs are made as a single integrated circuit (IC) on chip. They might have other devices integrated on the same chip, usually they are called System-on-Chip or SoC[63]. Most of the up-to-date computers have several CPUs in the same chip called cores and several levels of cache memory[6], [24]. In the rest of the thesis we would call several cores on the same chip with several cache levels as CPU. This notation is now commonly used in research papers and all the technology world.
Modern CPU has a long history with great set of ancestors, that people used for automation of calculations. Overview of it is provided in the following section.

2.2 Automation of calculations — from abacus to microprocessor

Performing calculations is a well-known task, discovered in ancient times. To make the process faster and more efficient, people tried to create some devices that help to calculate values easier. One of the first tools used for that purpose was abacus[78]. Usually it looks like frame with several lines of beads that can be easily moved on the strings or sticks[78]. Designs might vary (e.g. Russian abacus called schety and Japanese abacus called soroban looks a bit different), and algorithms used for each exact type of abacus differ as well. However, it was used from ancient times till the present time (at least in post-Soviet countries) to perform calculations, teach children and automate the calculation process.

In medieval times there was a couple of attempts to make some more advanced devices, but all efforts were put on astronomical calculations, so they made a good foundation for future research and plenty of new ideas, but no significant technical solutions for the future use.

In Renaissance times it was discovered that that division and multiplication can be performed with subtraction and addition. After that John Napier designed a tool named "Napier's bones" that simplifies calculations with divisions and multiplications[74]. Also idea of representation of real numbers as intervals leaded to invention of slide rule and its modifications that allow to perform division and multiplication extremely fast in terms of that times[74]. After that several attempts to design a calculating machine were made and in 1640s Blaise Pascal invented a calculator also known as Pascaline[74]. One more well-known mechanical calculator is Stepped Reckoner by Gottfried Wilhelm von Leibniz[74]. Both inventions by Pascal and Leibniz continued in the machine of Charles Xavier Thomas de Colmar called Arithmometer, built in 1820s. It was a reliable, portable enough and easy-to-use mechanism, so it was quite wide-spread especially in offices[77].

But all the machines described above had one common feature that might be annoying enough - it had to be operated by a human. However technical progress didn't stay still and with the development of technology first attempts of making an automatically calculating machines were made. Surprisingly, one of important ideas for future computer development came from weaving. Joseph Marie Jacquard in 1804 invented a Jacquard machine that was aimed to reduce expenses while manufacturing textiles with complex patterns. It used a long chain of punched carts looped in one uninterrupted sequence, and one cart represented one row of design.[3]

One more idea appeared during Charles Babbage's work on the Analytical Engine - a mechanical computer that was never built during Babbage's life, but had plenty of revolutionary ideas inside, that are used in modern machines (e.g. ALU, conditional

branching and I/O)[8]. Ada Lovelace, who was keen on Babbage's ideas, proposed an algorithms of calculating Bernoulli numbers using punch cards instruction and is considered as first computer programmer[8]. She also suggested to use Jacquard loops instead of punch cards. After that in 1880s, Herman Hollerith proposed to use punched cards for storing data[65].

The 1890s have seen an introduction of comptometer. Comptometer is essentially a Pascaline but it used buttons instead of input wheels. This has approached it to modern calculators.[47].

Early computers were all build for a single task and their algorithms were hardwired into the device. In 1936 Alan Turing formalised the notion of algorithm by using a novel concept or "universal" (Turing) machine. This formalization has paved the road to separation of program and hardware and so a single machine would be able to handle a whole class of tasks efficiently[26].

During World War II for deciphering Enigma ciphers Poland, Great Britain and US created a series of specialized electromechanical calculators called "cryptographic bombes"[69]. Those while being impressive were still not programmable other than inputting a ciphertext and possible plaintext.

In the same war, the need to break Lorenz cipher used by German high command lead to development of Colossus with involvement of Alan Turing[13]. It was the first electronic computer. It didn't need any mechanical parts to function and used vacuum tubes instead. This significantly sped up its operation. However this effort was classified and not known to the public until 1970's. Independently electromechanical programmable computers were created in Germany by Konrad Zuse in 1941 and at Harvard university in 1944. They were named Z3 (short from Zuse 3)[50] and Harvard Mark I[12] respectively. The first fully electrical computer known by public was ENIAC (abbreviation from Electronic Numerical Integrator and Computer) created in 1945[73]. However in those machines the programming was still done by physically connecting the wires which required considerable effort. First computer to have its program on a general purpose storage was "Manchester Small-Scale Experimental Machine" at Victoria University of Manchester in 1948[20].

1950s were characterized as beginning of commercial computers and increase in quantities. For UNIVAC I (Universal Automatic Computer) from 1951 46 units were sold[9]. For IBM 650 (model made by IBM - International Business Machines) from 1954 already 2000 units were sold [27]. The other big change in 1950s is the switch from valves to transistors. First transistor-based computer dates to 1953 at university of Manchester[4]. In 1959 IBM made a transistor variant of 650 and called them IBM 7070[62]. Increased speed, programmability and availability has extended the range of tasks performed by the computers. As different models had different applications in mind and compatibility was rarely considered this has let

to having a lot of incompatible machines. However, experience obtained during this period allowed to create a more designed system.

2.3 CPU evolution — from beginning to current days

History of modern CPU concept begins in 1950s. In that time there was no commonly used computer design. Each kind of machine was unique and no one cared about compatibility. All software written for one special type of machine was not suitable for other ones made by other company (and often even with different models of the same manufacturer). Also in that period there were no ideas of a "correct way" of making a computer, so each one had his own opinion and the only limitation were production and components costs. Also the machines of that time were extremely big, the computers of the end of 1950s were at least truck-sized. In that decade almost all computers were aimed to perform very specific tasks. Due to that fact many of them had specific features, e.g. worked on systems based on 10 instead of modern computers working on system of base 2.
However in 1960s it became apparent that part of the software could benefit from being reusable on different machines. Hence IBM decided to create different-purpose computers that would be binary compatible. They showed ingenuity and created a new concept of Instruction Set Architecture (ISA), where instruction set was detached from exact hardware.The new ISA received a name S/360 or System/360[28], [29]. Hence different machines can be compatible even if their hardware implementations are vastly different, and different implementations can be optimized for different purposes. When someone need a system optimized for speed, the whole ISA would be implemented in hardware. In case when budget version of system is needed, hardware would provide only minimal instruction subset and the rest would be implemented by a layer called microcode which would be completely transparent to the programmer.
At that time programs were often developed directly in assembly, so having complex instructions in the CPU increased developer's productivity. The S/360 was meant to be a successor of several earlier IBM machines meant for different usecases and hence has incorporated instructions for both scientific and business usecases. Hence IBM included a large amount of complex instructions. So it had, e.g. "CVD" (convert to decimal) as a single instruction[71]. As a side effect this also decreased the number of instructions CPU had to fetch from slow and cheap memory. On the other hand microcode was placed in a very fast memory as even in the most low-end machines microcode would still be small for the price of the storage not to matter much and microcode speed was very important. In retrospect the practice of including operations doing pretty complex tasks became known as CISC (complex

instruction set computing). Despite being one of very early architectures it was very successful and even to this day its modern descendant z/Architecture is still fully compatible with S/360 and is emulated by a cross-platform emulator Hercules[14], [25].

S/360 has its competitors. Most known are so called BUNCH: Burroughs, UNIVAC, NCR (NCR Corporation, originally National Cash Register), Control Data Corporation, and Honeywell. However S/360 was a market leader[51].

In 1970 and 1971 first personal computers "Datapoint 2200"[76] and "Kenbak-1"[75] were created. Their CPU weren't a single chip but a collection of large number of separate small chips.

The next breakthrough occurred in 1971 when Intel corporation manufactured first CPU Intel 4004 on a single integrated circuit (IC). Previously computers were made out of miniature discrete transistors[21].

Intel 4004 is a 4-bit CPU running at 740 KHz, could execute 90000 operations per second and has only 16 pins and 2300 transistors[37]. This is unimpressive by today standards but was enough for its use in electronic calculators and pinball games. Next year Intel followed up with an upgrade to 8-bit CPU: Intel 8008. This sparked the creation of first personal computer with microprocessor: Micral N[70]. 3 years later Intel released 8080. 8080 was based on 8008 but lacked compatibility[15].

In 1975 this chip was used by the first desktop/home computer ever: Altair 8800[49]. It had only few LEDs as output and switches as input. It costed 439 USD as a kit for assembly by customer or 621 USD pre-assembled. In 2017 it's equivalent to 2014 USD resp 2849 USD. Yet it was a first time an individual could own a computer and first home computer to achieve limited popularity.

Intel 8080 wasn't without competitors. Notable ones include Z80 by Zilog which is binary-compatible with 8080 and Metal Oxide Semiconductor (MOS) technology 6502 with an independent instruction set.

Those 3 CPUs are responsible for kickstarting booming home computers market. TRS-80 (Tandy/Radio Shack — 80) Model I[56], Apple II[53], Apple IIgs (Apple II-graphics, sound)[54], Commodore line of computers[55], ZX spectrum[57] all use some variant of those 3 CPUs.

In 1978 Intel release 16-bit modification: Intel 8086[15]. 8086 architecture is the basis of the x86 architecture and modern CPUs found in most of desktops, laptops and servers still start in binary-compatible mode with 8086[19].

In 1985 the first ARM (Advanced RISC Machine) chip was manufactured by Acorn. This started another ISA widely used to this day. From the beginning ARM starts as a 32-bit CPU. Unlike x86 which is CISC, ARM is RISC (reduced instruction set computing). The idea of RISC is that instead of trying to express difficult operation in a single instruction, make every instruction simple to execute and execute

more of them even if they are simpler. This approach allows to reduce die size and energy consumption. Nowadays many of application of ARM are tied to its energy efficiency and it's used in i.a. phones, embedded applications and some laptops[11].
The same year Intel releases 80386[15] which is 32-bit upgrade for 8086 line. Compared to 8086 this CPU also has MMU (Memory Management Unit) integrated on chip. MMU is a component which maps virtual addresses to physical ones. This allows different processes to have independent memory spaces and implement protection of one process from the other and swap memory. With MMU one would typically have a multitasking OS which in turn increases utilisation of all components of the computer.
In 1989 Intel releases 80486[15]. It introduces pipelining and on-chip L1 cache. For every instruction CPU has to do potentially a lot of things: fetch the instruction, decode it, read data, perform operations and write data to memory. Without pipelining CPU doesn't begin processing new instruction before the previous one is fully completed. The idea of pipelining is to have components to do their job in parallel on different instructions like a pipeline. E.g. instruction decoder can decode the next instruction when ALU is handling previous one and instruction fetcher can fetch yet next instruction. Traditionally a way to increase CPU performance was to increase CPU frequency. With higher frequencies less can be done per cycle. The solution is to have longer pipelines with every cycle pipeline advancing one step and the amount of work to be done per cycle stays small. The cache is much faster than the RAM memory and so it allows to fetch commonly used instructions (e.g. in functions or loops) and data much faster[39].
In 1995 DEC (Digital Equipment Corporation) released Alpha 21164 which introduced 96 Kibibyte (KiB) of $L2$ cache[42]. L2 is a type of memory that is cheaper than L1 while still being very fast [2] . Having 2 levels of cache allows to decrease number of time CPU has to go to slow memory for either instructions or data. The same year Pentium Pro by Intel also gets a 128 KiB L2 cache[33].
Optimal instruction set would be different for every CPU architecture. In order to keep backwards compatibility while being able to change underlying architecture Intel CPUs starting from Pentium Pro use microcode. In this case microcode allows to improve pipelining and takes care about the complexity of x86 instruction set without increasing the complexity of CPU. Pentium Pro also has a so-call instruction-level parallelism, where different components of CPU are used for different instructions. So in naive implementation a large chunk of CPU will be unused. To improve efficiency several instructions are executed in parallel on different parts of CPU as long as they are not dependent[1].
Yet another feature added in Pentium Pro are so called Single Instruction, Multiple Data (SIMD) instructions. While the concept existed since 60s, it was the first time

it was brought to x86. SIMD allows to perform the same operation on multiple data points greatly speeding up the computations on arrays and computations needed for 3D simulations and rendering[64].
Instruction-level parallelism comes at a cost of a sophisticated scheduling circuitry which increases cost and in some cases creates bugs like lockup in Loongson (Godson) 2F[22]. Intel decided to remedy it in their new architecture Itanium which was also meant as 64-bit successor to x86[40]. Their solution was to let compiler encode parallel instructions and hence the result would be fast and cheap chip. Unfortunately compilers lack necessary sophistication to create a performant code with explicit parallelism. Also compiler lacks critical information like which data is in cache and which not. E.g. load operation may take several times more time if it fetches from memory than when it fetches from cache greatly changing optimal parallelisation. At the end of the day Itanium was a failed experiment. Intel has abandoned it and instead adopted in 2004 AMD's (Advanced Micro Devices) 64-bit successor to x86: amd64, also known as x86-64[68]. Currently the most common CPU architecture for desktops and laptops is 64-bit variant of x86, also named x86-64[30]. For phones and tablets the most common one is 64-bit variant of ARM called aarch64 (short from ARM Architecture 64).
Currently used CPUs concept appeared in 2005, when it became clear that CPU frequency can't continue to increase significantly as by then even smallest step needed by longest pipeline was already within the order of magnitude of what silicon could achieve in theory[41]. New instructions could barely increase the speed as most useful instructions are already added. Hence the focus shifted to fitting several cores inside a single CPU. AMD Athlon 64 X2 and Intel Pentium D were one of the first CPUs of that kind[32]. This trend has continued in subsequent years with manufacturers putting more cores in the single CPU.
L1 and L2 caches significantly speed up the cores but they don't improve the case of memory used by several cores significantly. For this reason on-chip cache L3 used by all cores was introduced, and it began to be widely used around 2006. This allows to make better tradeoff between cache latency and hit rate, but still this improvement is not fundamental and adding several more cache levels wouldn't change the situation. Current CPUs has several limits to the CPU performance, such as physical limits on reducing the size of transistors, physical limits on increase of CPU frequency and thermal considerations for increase of number of cores.
Typical modern CPU architecture that appeared around 2005 and still stays actual is described in the following section.

2.4 Multi-core CPU as commonly used approach

As it was mentioned before, Intel x86-64 is now commonly used architecture for personal computers (PCs). Several more architectures uses multi-core approach, but to explain main idea typical Intel x86-64 CPU would be used[30].
Main components of the modern CPU are computational core (or several cores), cache memory of different levels, different buses and controllers[6]. Computational core is the main part of the CPU itself and of the whole computer. CPU could be named as computer's brain, and CPU core is CPU's brain. It performs all arithmetic and logic operations and also cares about all interrupts for fast and frequent task switching, commands receiving, decoding and processing, handling all eventual troubles and storing obtained results[24], [6]. Each core has own registers — small pieces of a very fast memory. They allow to store data that CPU uses for current operations. It is the fastest memory in the CPU, but its size is quite small[24], [6].
Usually there are more than one core in the modern CPUs. In this case CPU is called "multi-core". Multi-core CPUs allows to perform several tasks at the same time using different cores in parallel[24]. It allows to increase CPU performance. Most typical CPU architecture with 8 cores and 3 cache levels is shown on Fig. 2.1
One more part of a CPU that is really important is a cache memory. Commonly there are several levels of cache memory (most often there are three levels of CPU cache)[6]. It is aimed to store data that most probably would be needed by CPU in the nearest future. This allows to decrease delays as CPU shouldn't request data from RAM, that is really slow from the CPU's point of view, so delay becomes significantly smaller, and thus CPU performance increases. It can be considered as an intermediate buffer between CPU and RAM. CPU cache access time is much smaller that RAM access time, but significantly bigger, that registers response time. Size of all cache levels in total is much smaller that RAM size, usually tens of megabytes (MB) in total[6].
Generally, first level of cache memory (L1 cache) is smallest and faster and also most expensive in comparison with other cache levels. Usually each core has its own L1 cache, that is sometimes split into instructions and data cache. Second level cache also named L2 cache is also own for each core. It is bigger than L1 cache, but is a bit slower and less costly. That last level cache so-called L3 cache is biggest and slowest one, usually it is shared between all cores[6]. Cores and all cache levels are connected with buses. There is also a bus between CPU and RAM. Besides there are set of buses and controllers, e.g. RAM controller. They are aimed to search needed data, update them and get up-to-date versions, transfer them between cache levels and to or from RAM, and do all the service and maintenance tasks[24].
Introduction of several cache levels was a principal improvement of CPU architec-

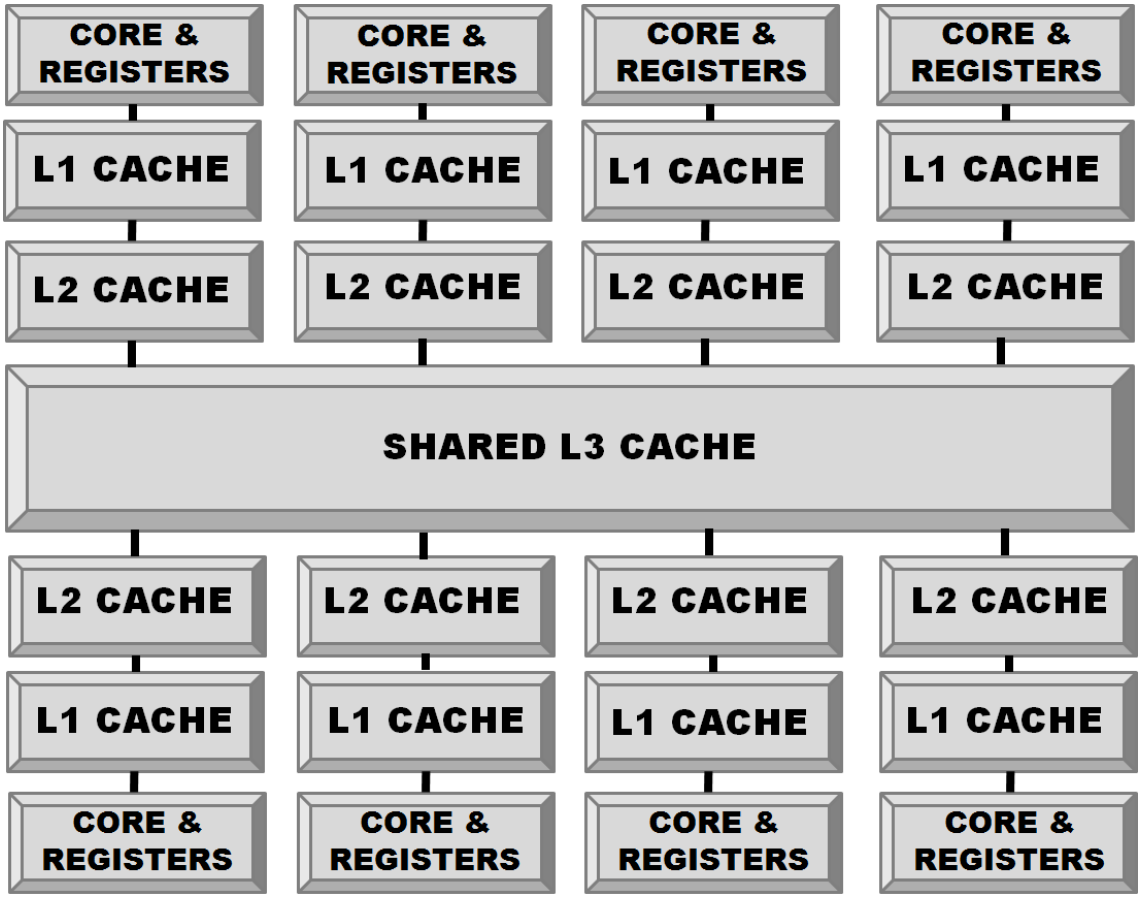

Figure 2.1 *Most common CPU architecture block diagram for 8 core CPU with three cache memory levels and shared last level cache.*

ture and it gave a huge gain for CPU progress[6]. Making this kind of memory and achieving a huge level of its reliability required a lot of significant efforts and costs. Modern architecture allows not only to decrease data access time, but also organize core-to-core communication using last level cache.Thereby data processed by one core could be immediately accessed by other one[6]. It leads to decrease each core's idle time and increases overall CPU performance.

Most important tasks for any multi-core CPU architecture are organization of shared access to stored data and timely update of each data instance to get the most actual copy accessible for each core. To solve it cache coherence protocols are used[31]. That's a set of very complex algorithms optimized for each CPU architecture and each CPU model. This issues complicates a lot adding more cores to the existing CPU architecture. So looks like in terms of scalability modern commonly used CPU architecture has several issues. As a result, new ways of core-to-core communication and CPU architecture organization are now in development. They would be discussed in the following section.

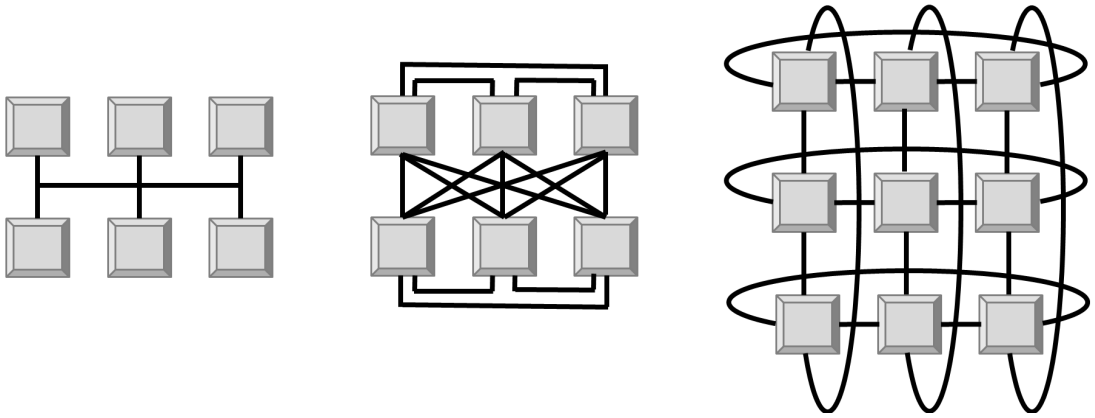

***Figure** 2.2 Networks-on-Chip most popular architectures: bus (left), mesh (middle) and 2D torus (right)*

2.5 Current research directions on CPU improvement

One of the most simple solutions of the CPU limitations problems is to build a new architecture of many cores connected with each other into some complex network topology. This idea bears a name of Network-on-Chip[60],[66]. There are a huge variety of proposed topologies such as bus or hierarchical bus, ring, 2D torus, mesh, fat-tree, cross bar, but single bus, 2-D mesh and torus covers more than 50% of cases[66]. These three architectures are shown on the Fig. 2.2. These architecture changes go hand in hand with proposals of efficient routing schemes on the selected architectures. For this purposes usage of small routers is proposed. Main goal for all NoC research proposals is to provide an energy-efficient solution with fast data routing and possibility of good integration into existing PC architecture and especially CPU and/or GPU (Graphics Processing Unit) architectures. "Classical" routing solutions such as using of lookup tables, is not suitable for this type of networks, so protocols using solutions with hardware logic are used[66]. Also complexity of all network elements, e.g. routers should be as low as possible. Also size and power limitations for these devices are very important. One more issue that is usually addressed in papers is an efficient cooling of proposed systems, as heating is one more trouble of modern CPUs[17]. In consideration of all aforementioned, one can conclude that it is a complicated task to connect many small scale devices with wires and provides delays that would be small enough to satisfy CPUs requirements, cooling issues and power consumption limitations. One more issue is making a die and fabricating a circuit with the required number of cores, that could be hard and expensive task.

Concept of Wireless Networks-on-Chip, where wired interfaces are proposed to be replaced on wireless links, could help with this issues[48], [23], [46]. Instead of wires and routers it is proposed to use miniaturized transceivers. There are many different

proposals, part of them suggests to use only wireless links for cores interconnect, other solutions offer to replace part of wired links onto wireless connections and leave some wired connections[23], [46]. Most solutions propose to use computation core with several cache levels as a single unit of WNoC. For example, in several papers it is proposed to replace wired links between L2 caches of each core and shared L3 cache to the wireless link[59]. Of course, in this case security issues should be taken into account. Moreover, using of this concept allows to make more complex spatial structures regardless of the wired connections (except the power ones that would be required in any case). Also space issues as well as cooling issues are not so acute, as more complicated 3D architectures could be used. Moreover, WNoC concept allows to make each core or small amount of cores on the same die and use a combination of that dies as a single CPU. It means that defective die wouldn't screw up the whole multi-core CPU and can be easily replaced with similar die. Besides, one-to-all and all-to-all communication becomes really easy as well as looking for the actual copy of data inside the CPU. Of course this concept offers new challenges such as interference mitigation, frequency reuse peculiarities and necessity of a very special medium access protocol (MAC) protocols designed for each exact type of WNoC. It is necessary to take into account all CPU peculiarities to make a proper CPU WNoC-based design.

To make a realistic assumptions about CPU NoC and WNoC design, it is important to get at least the order of magnitude of the expected buses load. To make any suggestions for massive multi-core CPUs , it is necessary to understand data rates on the internal and external interfaces in modern CPUs. Usually this data are taken from the CPU simulations. However, modern CPUs are very complicated, so make a simulator that takes into account all CPU features is almost impossible. Moreover, usually any documentation about internal structure of the CPU is under non-disclosure agreement, that makes any precise simulation and analysis impossible. Due to this fact is is important to make real life measurements to provide more accurate values of traffic. Next chapter is devoted to the measurements of the load of buses of interest and analysis of obtained results.

3. MEASUREMENTS AND ANALYSIS

This chapter is aimed to explain the full process of CPU real life measurements from the very beginning of the tool choice to the plotting of the dependencies of interest. Moreover, all obtained results are provided in a visual form and discussed.

3.1 Test stand technical description

The main reason for this research is to get a realistic data rates assessment on L2 – L3 cache bus and L3 cache — RAM controller bus on modern desktop CPUs that are commonly used. To obtain it we need a multi-core CPU that can be easily obtained from the shop or web site, so it should not be any specific-purpose CPU that is hard to get. As it was mentioned in the previous chapter, we decided to use CPUs with Intel x86-64 architecture, as it is most commonly used architecture. We started from Intel Core i7-2600K with regular CPU frequency of 3.4 GHz and maximal CPU frequency of 3.8 GHz and 4 cores[34]. After the methodology was tested we moved forward and rerun all our tests on Intel Core i7-5960X with CPU frequency range from 1.2 GHz to 3 GHz (without using TurboBoost mode with any cores), 8 cores and Haswell architecture[35].
While thinking about operating system (OS) for doing the CPU tests we had a huge variety of choices - almost all versions on Windows and Linux were suitable at first sight. After some research we ended up with understanding that all Windows options are not very convenient for our case as it is really hard to force Windows to minimize the background load while tests are run, so it would affect the results. Our aim was to get as pure tests load as possible, so OS load should be as small as possible. In a contrast with Windows, Linux allows to do it easily in almost all distributions, so we were able to choose any distribution that is familiar and convenient to use. Our choice was Kubuntu Linux as it is simple to use and efficient in terms of our tasks. We used Kubuntu 14.10 as the most stable Kubuntu version in the moment of the beginning of our research[45].
One really important part of our stand was an amount of RAM we have in the test computer. It is not so obvious from the beginning, but RAM can be a huge bottleneck for CPU efficiency. If the size of the RAM is too small the OS would be

forced to drop disk caches or to swap out some pages to disk. When the page in question is needed, time to load it from disk would create a significant delay and hence increase CPU idle time. So it would decrease total system efficiency. Someone can argue that we need to know typical load of buses inside the CPU, so it is not a significant part of testing stand and usually people don't want to add as much RAM to the system as possible. However, one of our purposes is to estimate the highest possible load on CPU buses, and in this case amount of RAM is important. Besides, while estimating CPU buses load we should keep in mind that the aim of the whole research is to understand this values and use it in future CPUs development. It means that the amount of data going through the buses inside the CPU per time unit would probably be not smaller than the same value for modern high-end CPU. For the same purposes we used Solid-State Drive (SSD) instead of Hard Disk Drive (HDD) as usually SSD provides smaller delays while processing data.
We ended up with the following configurations of the test stand:

- CPU: Intel Core i7-5960X with Haswell architecture and CPU frequency range from 1.2 GHz to 3 GHz[35]
- Motherboard: Asus X99-A LGA2011-3[5]
- RAM: Kingston HyperX Predator DDR4 (Double Data Rate 4th-generation), 16 GB (4x4GB)[44]
- Video Card: NVIDIA 750GTX[52]
- SSD: Samsung 850 256 GB[67]
- OS: Linux Kubuntu 14.10[45]

The configuration remained the same during the whole testing process, no software and/or hardware changes were made. Aforementioned desktop configuration was easy to find in the stores, so no unique equipment was used.
The next step after test stand setup is to choose a proper software tool and measure data rates on buses of interest. Both issues as well as full description of chosen test are discussed in the following chapter.

3.2 Software tools and tests used for research

There are several options of getting internal and external CPU buses load. For example, we can just analyze datasheets for modern CPUs provided by Intel and

try to make some mathematical model of CPU operations. It might be an option, but the complexity of this approach is too high. Also it is possible to use some software that can simulate of CPU and get data rates from there. It also might work, but either it would take almost infinite time to learn about all the peculiarities of CPU operations and take it into account or simulation model would be not quite precise. The much promising option is to measure real load of an existing CPU. However at least in a moment when the research started, there were no tools allowing to get any exact load on L2—L3 cache bus and L3 cache — RAM controller bus in any unit that is used for data rates assessment. Usually this kind of tools gives some ideas about behavior of internal CPU counters, e.g. cache misses and cache hits. Some tools also allow to get a load on a PCIe (Peripheral Component Interconnect Express) bus, but that is the total load, which includes a traffic from many different devices that use PCIe bus without ability to demarcate traffic for different parts of the system.

After some search we ended up with using Intel Performance Counter Monitor (Intel PCM)[36]. That is a special tool provided by Intel and allows to get internal CPU counters values and save them to csv file. Of course there are couple of analogs that can be used, e.g. Perf utility[61]. The main reason for using Intel PCM was the fact that it is an official software tool provided by Intel for its own CPUs and they have a set of forums, where Intel employees usually provide a help in case of any troubles. Intel PCM allows user to choose if he would like to save the results to the file or not. If the answer is yes, than user should choose a name and path for the future file. No warning about rewriting of existing files wouldn't be provided, so it is necessary to be careful with writing data to file not to loose something. Also user can chose how often the measurements should be recorded and how many measurements points are needed. The most common call of an Intel PCM tool we used looks like follows:

```
./pcm.x $frq -i=$lng -ns -nc -csv=$filepath/$filename.csv
```

`$frq` there is a step in seconds with that Intel PCM performs measurements. Measurements frequency can be easily obtained from this value - we just need to divide 1 by the `$frq` to get measurements frequency per second. `-i=$lng` defines the number of points we would like to get data in. The amount of points should be big enough to get the realistic data from CPU counters especially if the measurements frequency is high enough. Using of `-ns` and `-nc` is aimed to make output easy readable and hide information that would be not useful with very high probability. `-csv=$filepath/$filename.csv` allows us to save all the data into csv file with specified name to specified location. Usually call of Intel PCM during the measurements was looking like follows:

```
./pcm.x 0.2 -i=12000 -ns -nc
```

```
-csv=./logs_core/pcm_2_Background_5_2100.csv
```

With this call Intel PCM will make measurements 5 times per second, the measurements would be taken 12000 times, so test would last for 40 minutes and all the results wold be saved to the folder named "`logs_core`" with the name "`pcm_2_Background_5_2100.csv`". The files names notation would be discussed in the section 3.3.

It is important to measure buses load for different types of testing programs and both high and low load are important. To generate a reasonable load we had two groups of tests. First one was aimed to get a load of real life CPU applications e.g. working with a complex computational task, etc. The second one was aimed to get the highest possible data rate and the average data rate of the well optimized program. For the first group of tests set of typical tasks for CPU was chosen. Initially, the following tests were proposed:

- OS background load
- Encryption
- Linux kernel compilation
- CPU video decoding
- Computer game
- Video call
- Archivation

After the proof of concept testing on the first test stand with 4 active cores, we realized that some tests are not suitable for our case. To be as precise as possible we decided to have long runs for each of the test, however the testing time should be reasonable. After set of experiments we ended up with the following conclusion:

- Measurements frequency should be 5 times per second or less. If the frequency is higher than 5 measurements per second, the Intel PCM working time would be significantly higher than it should be. E.g. when we run a test with 10 measurements per second and 1000 measurements points, it should end in less than 2 minutes. However in real life it ended in almost 4 minutes. So all the measurements should be performed with frequency 5 or less measurements per second.

- Optimal test length is 12000 measurements and higher. After this number of points average values of counters of interest (that would be discussed further) changes insignificantly.
- During the first couple of seconds Intel PCM generates additional load, so some small amount of measurements from the beginning should be removed. We decided to remove first hundred of measurements to make sure that we have as pure counters value as we can.
- According to the aforementioned values, it took us 40 minutes to have 1 log file for one set of parameters.

In case of Linux kernel compilation we faced a trouble with its operating time. As it was explained before, the test lasted 40 minutes. However, on both CPUs we've used for testing, Linux Kernel compiled with any set of parameters during the half of hour at most. We had to abandon this test due to time issue.
Also archivation test was removed from the test set. While processing the results we realized that it is very uneven in terms of buses load. At some time periods load is very small, after that it becomes several times bigger for couple of seconds and than drops back, so average values are not applicable for this test.
In case of one active core or single core CPU testing process is obvious. It is just needed to run the test and then run Intel PCM to get a csv file with all counters that you need. In case of multi-core CPU it is not fair to compare buses load for different number of active cores with each other when the only instance of the testing program works. In this case often only one core would perform all the computations and all other would stay idle. That is especially true for tasks that are hard to solve in parallel. To make testing more fair, we decided to run as many instances of testing program as the number of cores currently active in the CPU. That works perfectly for all the tests except the video call. In case of video call several instances of any video call program e.g. Skype or Hangouts should run. They all must have different user accounts and be active on the same computer. And of course they all should somehow share microphone and camera. Usually this kind of programs don't support two or more calls simultaneously, so this test was rejected as well. Finally, we ended up with the following real-life tests:

- OS background load — Linux Kubuntu 14.10 working in an idle mode and performing only those computations that needed for basic functioning of the system.
- Encryption — encryption of a huge file of several gigabytes with AES-256 (Advanced Encryption Standard)[16]. We used OpenSSL (Open Secure Sockets

Layer) utility for AES encryption and all the encryption results were sent to /dev/null to save the space and reduce delays[58].

- CPU video decoding — playing of a high-resolution video, that couldn't be decoded on a GPU and decodes on a CPU instead

- Computer game — playing a non primitive game. While running games on Linux we faced a trouble as usually modern ones requires additional software to run on it. We had to chose older game called Total Annihilation with the highest possible settings[72]. To have an option of running several games on the same computer, we forced each instance to use autoplayer , so all instances were fully workable and generating a load on internal and external CPU buses.

Second group of tests is one with artificial tests. To estimate the highest possible load we made a computer program, that was written in C language. It uses drawbacks of cache prediction mechanism (prefetching), that works as follows: any data going from RAM to CPU on the blocks of 64 bytes[31]. If some core needs a byte of data, RAM would give this byte and 63 next because they might be used in the future computations.
In the beginning of our test we allocated a memory and create an 1 GB array of char symbols. In C language each char symbol has a size of 1 byte. This array is definitely bigger that CPU cache size, so it would be stored in a RAM. After that, we start an infinite loop, inside which we are reading each 64th element of the array. In this case cache prediction mechanism works badly and with the byte we are asking for we get 63 more, that are useless in our case. So each time when we would like to read the data, we have to ask new piece of it on the RAM. This process generates high load on the internal and external CPU buses. We call this test "load maximization".
Opposite to the previous test we constructed one more aimed to get a load of well-optimized program. It uses benefits of cache prediction mechanism and present a program on a C language, that creates 1 GB array of char symbols in the beginning of operation. After that in the infinite loop each element reads consistently. So cache prediction mechanism works pretty well and with the byte we need next 63 bytes would come from RAM. They would be used on the next 63 steps, so the next query to RAM would appear only after 63 more cycle iterations. It got a name "load optimization".
So all the tests are described above in details and we can proceed to the methodology of measurements description, which is explained in the following section.

3.3 Measurement methodology for Intel PCM

We would like to understand how buses load changes depending on the CPU frequency and number of CPU cores. To achieve this goal we did two sets of measurements. In one of them the number of cores should be fixed and the CPU frequency changes. In other CPU frequency remains the same and the number of active cores changes.
It is very important to keep the room as cool as possible to avoid CPU overheating that might results in decrease CPU frequency or limitation of CPU performance.
To perform the set of measurements describing dependence of internal and external CPU buses load on CPU frequency, the following steps should be done:

1. Enable number of cores that was chosen for the set of measurements and disable all the rest. It should be remembered about differences between number of physical cores and logical cores in modern CPUs. Usually there are twice more cores in the OS's point of view, than the number of physical cores is. It means that there are two logical cores per one physical while working with the OS. So if we need to run some tests with one active core on a CPU with 4 physical cores, we most probably need to disable 7 logical cores to leave one core alive. In case of a CPU with 8 physical cores that we used for testing, we have 16 logical cores. We decided to make this set of measurements for 1, 4 and 8 active cores and for all available CPU frequencies for each amount of active cores. We can use the following command to enable logical core with number `$X` (number varies from 0 to 15 in our case):

   ```
   echo 1 > /sys/devices/system/cpu/cpu$X/online
   ```

   To disable logical core with number `$X` we can use the following instruction, where `$X` varies from 0 to 15, as it was mentioned before:

   ```
   echo 0 > /sys/devices/system/cpu/cpu$X/online
   ```

   It should be mentioned that one logical core always stays alive, so the tools doesn't allow us to disable all the cores.

2. Set CPU frequency equal to the lowest one. In case of our CPU it was a frequency of 1.2 GHz. It can be done with the following command, where `$A`, `$B` and so on are the numbers of all active cores that are enabled for the test:

   ```
   cpupower --cpu $A,$B frequency-set --freq 1200MHz
   ```

3. Wait for couple of seconds and check that all changes were successful and the number of active cores and their frequencies are set properly. It can be used e.g. with the following command:

   ```
   cat /proc/cpuinfo
   ```

4. Run as many instances of the testing program as number of active cores. This step is not suitable for measuring OS background load, as we can't run several copies of OS simultaneously on the same machine and make them fully functional.

5. Run Intel PCM with measurements frequency of 5 measurements per second or less and number of measurements equal to 12000 or more, as it was described and explained in the previous section. One test run would last for 40 minutes. As it was mentioned before, the following command could be used, where we specify the location of the csv file with results using `$filepath` and `$filename`.

   ```
   ./pcm.x 0.2 -i=12000 -ns -nc -csv=$filepath/$filename.csv
   ```

6. Wait until all the measurements are finished.

7. Repeat items 2—6 from this list for all possible CPU frequencies. Note, that not all CPU frequencies are available on different CPU models and BIOS (Basic Input/Output System) implementations. Usually frequencies that are the multiple of 100 and being within CPU frequencies range can be used as CPU frequencies. However, in our test stand this rule was not always true. Operating frequencies available on our stand were 1.2 GHz, 1.3 GHz, 1.5 GHz, 1.6 GHz, 1.7 GHz, 1.8 GHz, 2 GHz, 2.1 GHz, 2.2 GHz, 2.4 GHz, 2.5 GHz, 2.6 GHz, 2.7 GHz, 2.9 GHz, 3 GHz, so we can see that some points, e.g. 1400 MHz were missing. Usually frequencies of e.g. 1201 MHz are not possible to set on the CPU.

8. Repeat items 1—7 from this list for all chosen numbers of active cores.

One more dependence that probably is even more interesting than the dependence between buses load and CPU frequency is a dependence between buses load and number of cores in the CPU. We can enable and disable cores to perform the measurements and get an idea about this dependence. To perform the set of measurements describing dependence of internal and external CPU buses load on CPU frequency, the following steps should be done:

1. Enable chosen number of cores. Usually we started from 1 active core and then increase it after tests for all frequencies are finished. It worth remembering about the fact that usually we disable and enable logical cores instead of physical, and the amount of logical cores is twice higher the amount of physical one. Enabling and disabling cores can be done using the following commands respectively, where `$X` is a number of the logical core and in our case lying in a range between 0 and 15 inclusive.

   ```
   echo 1 > /sys/devices/system/cpu/cpu$X/online
   echo 0 > /sys/devices/system/cpu/cpu$X/online
   ```

2. Set chosen CPU frequency for all active cores. We did this set of measurements for CPU frequencies of 1.2 GHz, 2.1 GHz and 3 GHz. The following command could be use for this purpose:

   ```
   cpupower --cpu $A,$B frequency-set --freq 1200MHz
   ```

   `$A`, `$B`, etc. are number of all enabled logical cores.

3. Wait for several seconds and check that the test stand is set up properly.

4. Run number of testing programs equal to he number of active cores.

5. Run Intel PCM with measurements frequency of 5 or less measurements per second and number of points in measurements exceed 12000 or equal to it.

6. Wait until measurements would be finished. It takes approximately 40 minutes to complete the measurements with 12000 points and measurements interval of 0.2 seconds.

7. Repeat points 2—6 for other CPU frequencies if needed. We did it for CPU frequencies of 1.2 GHz, 2.1 GHz and 3 GHz.

8. Repeat point 1—7 for all possible number of active (physical) cores.

Also it is important to give proper names for files with measurements result. We tried several options and ended up with the following rules:

- Different measurements sets should be stored in different folders named after measurements aims with added date of starting this set of measurements. Also it is better to use underscore instead of space in the folder name as it helps while processing files and folders on different systems and OSes. For example, if we would like to run the set of tests to get a buses load dependency on CPU frequency, and this set of measurements was started on 12th April 2016, we can name the folder e.g. “`frequency_logs_12042016`”.

- File names should have a logic and allow to understand all the parameters used for the measurements from the file name. We start all the file names from "`pcm_`" to make further parsing of files easier. Then we add number of active cores and underscore, then the name of the test consists of one word started with uppercase letter. Test name starts from uppercase to obtain easy search for people working with this logs, as uppercase letter shows the usually most important part of a filename - a test type. Then we add one more underscore and a measurements frequency in times per second. This parameter is not needed when we use the only measurements frequency but it is really convenient for further logs processing. After that goes underscore and currently used CPU frequency in MHz. In the end we add file extension that is ".csv". As it was mentioned above, the typical name of a file with results could look as "`pcm_2_Background_5_2100.csv`". It means that this file stores all the data we got from the OS background load measurements with 2 active cores operating on 2.1 GHz and measurements were taken 5 times per second.

It should be mentioned that getting the final data for further research takes a lot of time. Just a test without any preparations for it lasts for 40 minute in our case. To get all the tests with different frequencies we should run 270 tests (6 types of load, each run for 3 different number of active cores and 15 different CPU frequencies). So it takes 180 hours to get the intermediate results for the first set of tests. To get all the test results for different number of cores for 3 different frequencies and 6 load types we need 144 runs of the test, i.e. 96 hours of measurements. It is too long to operate all these tests manually, so several bash scripts were written. They allow to automate almost all tests except of CPU video decoding test. This test was carried out manually as often during 40 minutes of video decoding run something went wrong and video freezes for very long period. As we didn't want to make any changes on the testing stand and remeasure all the values we got, this test was conducted with a full human supervision.

After carrying out all aforementioned tests we would like to somehow obtain the buses of interest loads from the results we got. In the saved files there are a lot of columns with values, but none gives an actual data rate. However we can estimate it using the data we got. There are two columns named L2MISS and L3MISS. They contains a values of cache misses of 2nd and 3rd level in millions. When the L2 cache miss happens, query with the address of data we are looking for, goes to the L3 cache. From the core's point of view after some time piece of data with size of 64 bytes comes back. The address size in modern CPU differs from model to model, but it is now bounded by 64 bits, so when we transmit address we can think that we transmit 64 bits (8 bytes) at most. So after the each L2 cache miss 8+64 bytes

go through L2-L3 cache bus. So the data rate on L2-L3 cache interface in gigabits per second (Gbps) could be estimated using the following formula:

$$S_{L2-L3} = \frac{L2_{miss,av} \cdot 10^6 \cdot 8 \cdot 72}{10^9} \tag{3.1}$$

$L2_{miss,av}$ in this formula is an average value of L2 cache misses per second in millions. Multiplication by 8 gives us value in bits per second, multiplication by 10^6 added due to the fact that $L2_{miss,av}$ is in millions, 72 bytes is a total packet size consists of 8 bytes for address and 4 bytes for data. Division by 10^9 used to get a the value in Gbps.

When the data of interests were not found in L3 cache, query with address goes to RAM and after some time response comes back with 64 bytes of data. The query to the RAM possible only after unsuccessful search in last level cache, so the amount of queries goes to RAM is equal to the number of L3 cache misses. According to this fact, the following formula can be used to calculate L3 cache - RAM controller bus data load in Gbps:

$$S_{L3-RAM} = \frac{L3_{miss,av} \cdot 10^6 \cdot 8 \cdot 72}{10^9} \tag{3.2}$$

$L3_{miss,av}$ is a number of L3 cache misses in million, all other multipliers' origin is the same as in previous equation.

It is important to note that equation 3.2 doesn't give the total load of a bus between CPU and RAM as it is used also for several other purposes. We can estimate just a part of its load, that is directly related with CPU and RAM only and can be denoted as a load of an intra-CPU bus between last level cache and RAM controller.

One more step in the way of getting results is data processing using aforementioned formulas to get them in easy to use form. Of course it is possible to handle all the data using any spreadsheets editors, but it is tiresome and senselessly to do it for more than 400 different files and then combine the results into plots or tables. It was decided to write some code for automatic processing of csv files obtained during the measurements to facilitate the task. Python programming language was chosen for this task as it is easy to understand and widespread language allows to solve many tasks e.g. reading data from csv fie, etc. easily. There were several classes made for this task. One of them is a serializer class, that allows to read data from files with specified name and specified folder, load all the files from the folder, and especially read columns of interest from the files, handle all the errors and possible troubles. The main aim of this class was to provide an array of several data arrays that can be further used for analysis.

During files parsing it was important also to force Python to use a proper delimiter

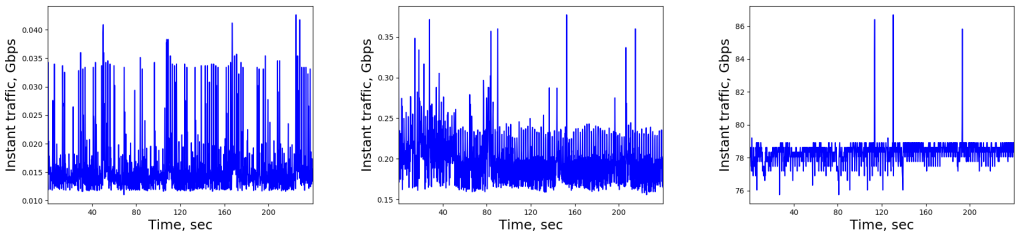

Figure 3.1 *Instant traffic on L2-L3 cache bus in Gbps for 4 minutes interval for OS background load (left), "load optimization" test (middle) and "load maximization" test (right). All graphs are for 2100 MHz and 8 active cores.*

while reading the csv file. Intel PCM saves results to the file with semicolon (";") delimiter, so it should be taken into account while writing the code. During all the experiments we realized, that in the beginning of the test there is an activity burst irrespective to the test type. It is connected with Intel PCM self setup in the beginning of each measurement. Due to this fact it is better to delete several tens of points from the resulting array to make data more "clean" and related only to the test's traffic. It was experimentally revealed that values around hundred and more are most suitable for amount of points to delete.

After we have all the data, we can compute mean for each set of parameters (test type, number of cores, clock frequency) and plot the dependencies of interest. It is important to remember that in the csv files both L2 and L3 cache misses values are in millions. Moreover, we had measurements for 5 times per second, so if final result for traffic values is needed e.g. in Gbps, it is compulsory to make a conversion between millions of misses in 0.2 seconds and millions of misses in seconds.

To plot all the results we got matplotlib library was used. It allows to plot Matlab-like plots with different line colors, types and markers, also with both axes labels and proper legends. Moreover, numpy library was used for all mathematical calculations. Results obtained using this software are presented in the following sections.

3.4 Analysis of L2-L3 cache bus load

This section is aimed to provide an overview of L2-L3 cache bus load dependencies from time, number of cores, CPU clock frequency and type of load.

First of all, it is important to understand how bus load changes with time and what kind of changes happens during the tests. It is important to note that all plots for L2-L3 cache bus shows total traffic for all links between L2 caches of several cores and shared L3 cache. Hereafter "traffic" for L2-L3 cache bus would be used for total traffic on this link designation.

In the figures 3.1 and 3.2 we have a plots of instantaneous traffic between L2 and L3

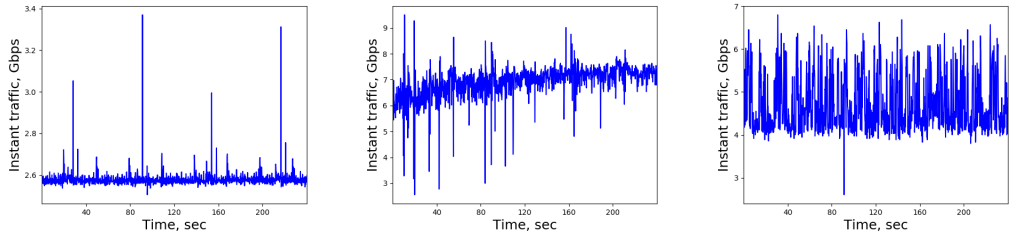

Figure 3.2 *Instant traffic on L2-L3 cache bus in Gbps for 4 minutes interval for encryption test (left), computer game test (middle) and CPU video decoding test (right). All graphs are for 2100 MHz and 8 active cores.*

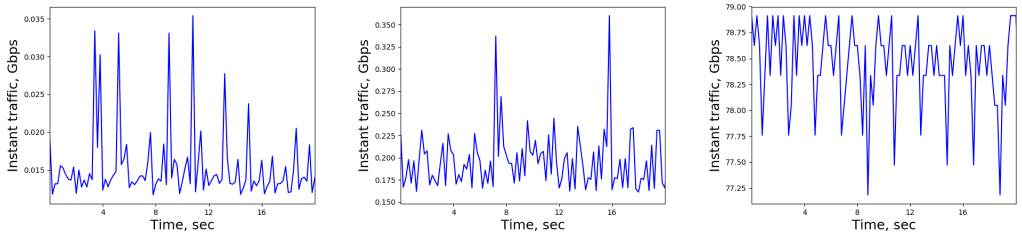

Figure 3.3 *Instant traffic on L2-L3 cache bus in Gbps for 20 seconds interval for OS background load (left), "load optimization" test (middle) and "load maximization" test (right). All graphs are for 2100 MHz and 8 active cores.*

caches for 4 minutes. The graphs are for OS background load, "load optimization" test, "load maximization" test, encryption, computer game and CPU video decoding. As the nature of the graph doesn't depend on number of cores and frequency, we decided to show only the graphs for 2100 MHz and 8 active cores. For OS background the graph is randomly oscillating between 0.012 Gbps and 0.04 Gbps with most of the graph being between 0.012 Gbps and 0.02 Gbps. "Load optimization" graph is oscillating mainly between 0.15 Gbps and 0.25 Gbps with occasional spikes up to 0.38 Gbps. "Load maximization" looks smoother than previous 2 graphs and is oscillating mainly between 77 and 79 Gbps with occasional spikes up to 87 Gbps or down to 75.5 Gbps. Encryption test graph is smooth and stays around 2.6 ± 0.05 Gbps with few spikes to 3.4 Gbps. Computer game test starts at around 6 Gbps and slowly increases to 7 Gbps with random oscillation of top with most of them being no more than ± 0.5 Gbps away from centerline, occasional spikes can deviate up to 3.5 Gbps away from centerline. CPU video decoding test unlike previous tests shows periodicity due to nature of CPU video decoding because CPU would go idle after decoding enough frames for display. Lower periods are at about 4 Gbps and higher at around 6.5 Gbps. There is a single downward spike of 2.5 Gbps. Additional figures 3.3 and 3.4 show the same dependency but zoomed into first 20 seconds. Those versions have the same tendency and ranges as full version but illustrate closer

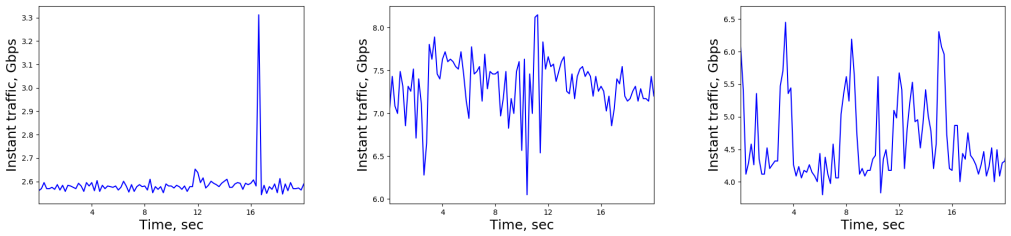

Figure 3.4 *Instant traffic on L2-L3 cache bus in Gbps for 20 seconds interval for encryption test (left), computer game test (middle) and CPU video decoding test (right). All graphs are for 2100 MHz and 8 active cores.*

the nature of oscillations. They're essentially random for graphs for OS background, ''load optimization" and computer game. Plot for encryption test is essentially 2.6 Gbps with tiny random oscillations and one single spike to 3.3 Gbps. Also ''load maximization" test shows some periodicity on the closer look. CPU video decoding also shows periodicity of around 4 –5 seconds.

It can be noticed that all tests shows random oscillation except the CPU video decoding tests which shows periodicity due to tests nature. Moreover, in some cases ''load maximization" test can also show periodicity. Also we can have ideas about the instant load for different tests.

On the graphs 3.5 and 3.6 we show the dependency of L2-L3 traffic depending on the CPU frequency. The graphs differ by the number of cores. They are for 1 active core (Fig. 3.5), 4 and 8 cores (Fig. 3.6 left and right respectively). The graphs are logarithmic to better illustrate values at different orders of magnitude. The bottom line (green) on all the graphs is OS background load at 10^{-2} Gbps for all 3 plots. It varies a bit in a noise-like level but essentially stays constant across all frequencies for all number of cores. It is the lowest line by far. Most of other lines show linear dependency of the throughput on the frequency. They are however at different levels. Red ("load optimization" test) line goes from 0.12 Gbps to 0.19 Gbps for 1 core. It is the second-lowest line other that OS background load. Cyan (encryption test) goes linearly from 1.35 Gbps to 3.52 Gbps for 1 core. Black (computer game test) goes from 3.90 Gbps to 8.45 Gbps for 1 core. Finally the blue one ("load maximization" test) goes from 46.53 Gbps to 106.21 Gbps for all number of cores and it is the highest line on the graph. Magenta (CPU video decoding test) is staying constant at 0.64 Gbps for 1 core. This is due to the nature of video decoding test: in our setup it decodes the same amount of video per seconds independently

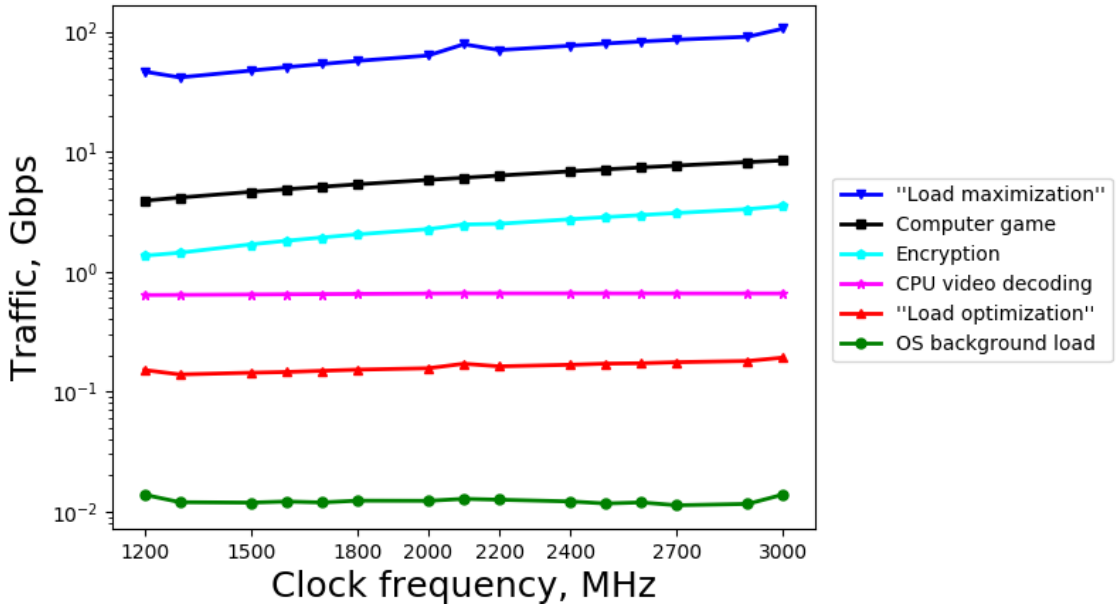

Figure 3.5 *Dependence of total traffic on L2-L3 cache bus on CPU clock frequency for one active core.*

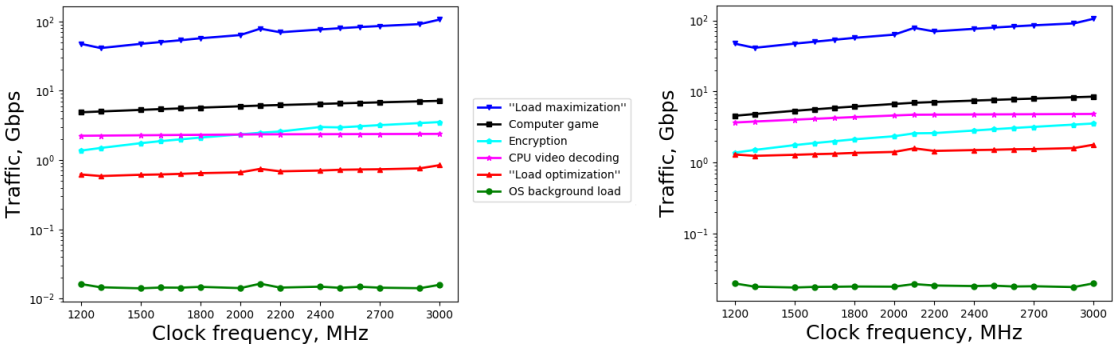

Figure 3.6 *Dependence of total traffic on L2-L3 cache bus on CPU clock frequency for four active cores (left) and eight active cores (right).*

of the CPU frequency. The graphs on Fig. 3.6 for 4 and 8 cores are similar to Fig. 3.5 for 1 core, just at higher levels. All the behavior tendencies are exactly the same.

We can conclude that load generated by all tests except CPU video decoding test increases linearly with CPU clock frequency increase. CPU video decoding is not linear due to peculiarities of our test stand setup. Moreover, OS background load stays almost constant and varies on a noise-like level.

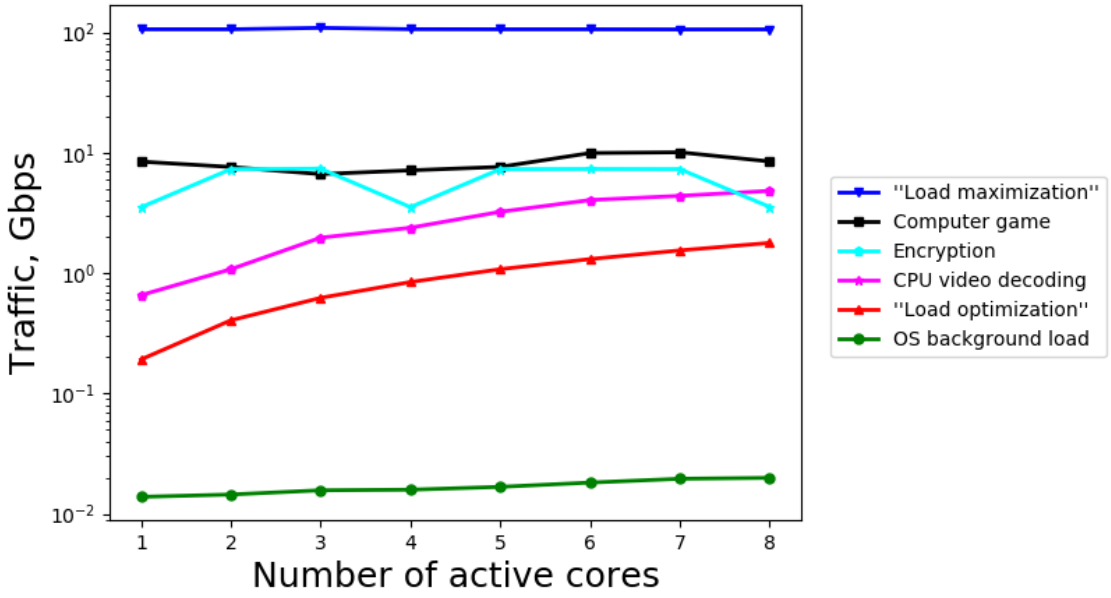

Figure 3.7 *Dependence of total traffic on L2-L3 cache bus on number of active cores for clock frequency of 3000 MHz.*

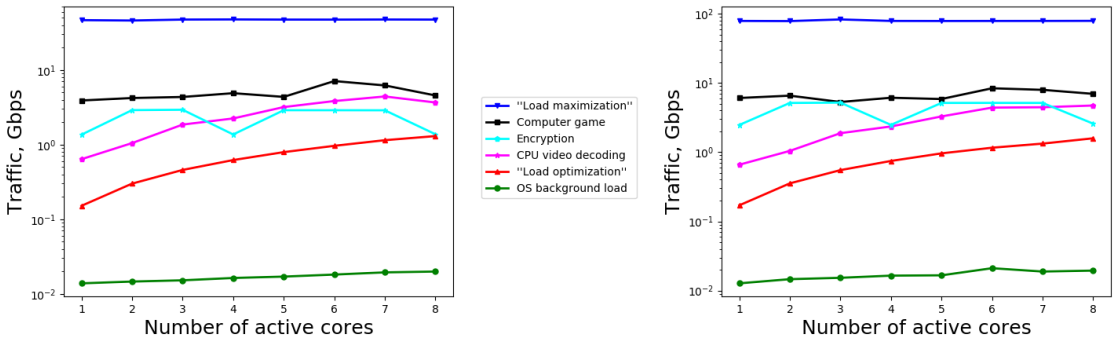

Figure 3.8 *Dependence of total traffic on L2-L3 cache bus on number of active cores for clock frequency of 1200 MHz (left) and 2100 MHz (right).*

In graph on Fig. 3.7 we show dependency of L2-L3 traffic on the number of cores. The lowest line is OS background load (green) which is at around 10^{-2} Gbps. The next line is red ("load optimization" test) going linearly from 0.19 Gbps to 1.78 Gbps. Magenta (CPU video decoding) shows similar dependency but is higher going from 0.65 Gbps to 4.83 Gbps. Black (computer game) shows the same dependency but the values are also higher and belong to the interval from 8.44 Gbps to 10.9 Gbps. Cyan line (encryption test) is in the almost same range but is more noisy, the highest value is around 7.38 Gbps. Blue line ("load maximisation" test) is staying almost constant at 109.17 Gbps. Plots depicted on Fig. 3.8 shows same dependencies for

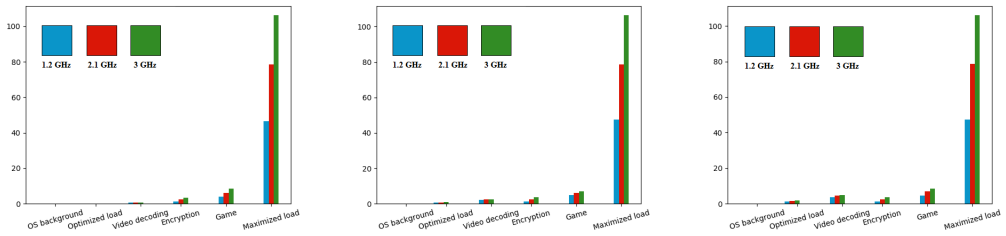

Figure 3.9 *Overall comparison of traffic on L2-L3 cache bus generated by tests for one active core (left), four active cores (middle) and eight active cores (right) for different clock speeds.*

lower CPU clock frequencies. They are essentially the same just at different levels. Due to the fact that levels are smaller than on Fig. 3.7, this data are not so important as for applicability assessment we usually need highest data rates from several possible. This plots are provided more for giving an idea of CPU behavior on other clock frequencies and proving that types of dependencies are exactly the same for all clock frequencies.

It is important to note that we can get idea of L2-L3 cache bus load changing with increasing number of CPU (active) cores. The type of dependency is linear for all tests except "load maximization" one, that faces RAM bottleneck. Also we can have an idea of different CPU loads disposition relative to each other. "Load maximization" test generates the highest load on L2-L3 cache bus that is up to 109.17 Gbps, after that goes computer game and encryption, with up to 10.9 and 7.38 Gbps load respectively, followed by video decoding test with the load up to 4.83 Gbps and "optimized load" test that generates load of 1.78 Gbps. The smallest load on the bus of interest is provided by OS background load, the values are around 0.02 Gbps."

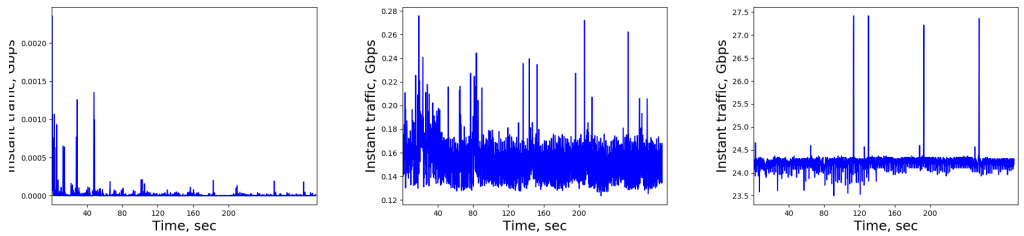

Figure 3.10 *Instant traffic on L3 cache — RAM controller bus in Gbps for 4 minutes interval for OS background load (left), "load optimization" test (middle) and "load maximization" test (right). All graphs are for 2100 MHz and 8 active cores.*

Overview of the whole situation in general in linear scale is provided on Fig. 3.9.

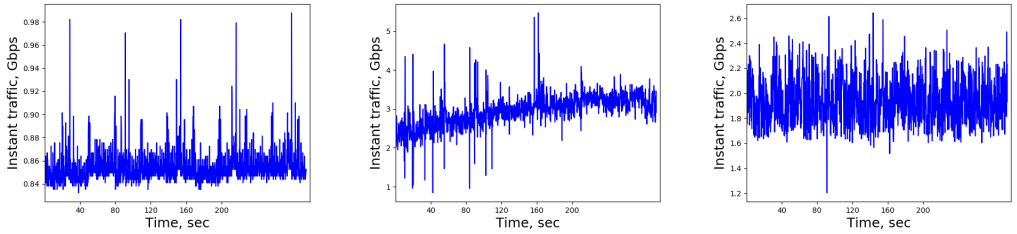

Figure 3.11 *Instant traffic on L3 cache — RAM controller bus in Gbps for 4 minutes interval for encryption test (left), computer game test (middle) and CPU video decoding test (right). All graphs are for 2100 MHz and 8 active cores.*

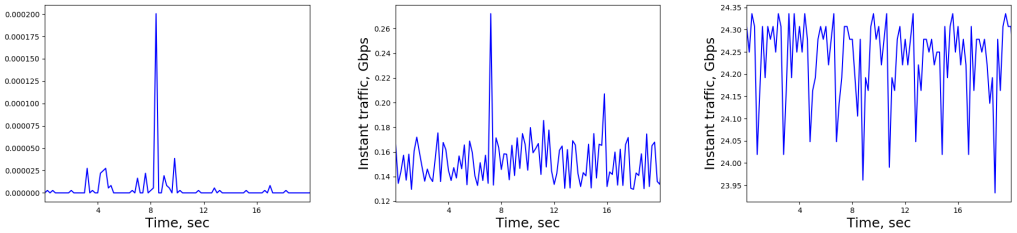

Figure 3.12 *Instant traffic on L3 cache — RAM controller bus in Gbps for 20 seconds interval for OS background load (left), "load optimization" test (middle) and "load maximization" test (right). All graphs are for 2100 MHz and 8 active cores.*

There are several bar plots for each test showing L2-L3 cache bus load for all types of tests for minimal (blue), maximal (green) and middle (red) clock frequencies.
We can notice that increasing clock frequency results in proportional increase of L2-L3 cache bus load for all types of tests except CPU video decoding and OS background load. Load on L2-L3 cache bus increases linearly with increasing number of cores for all tests except "maximized load" one as it faces RAM bottleneck and couldn't increase anymore. In terms of data rates on L2-L3 cache bus artificial test imitating "maximized load" condition works as intended and make greatest possible load. The smallest load is generated by OS background. Second artificial test imitating "optimized load" conditions also works as intended and shows up as smallest load we got except the OS background one. Game, encryption and CPU video decoding makes load of the same order that does not exceed 11 Gbps. OS background load is negligible in comparison with all other tests and stays around several dozens of Mbps for all tested cases. Load of "optimized load" test does not exceed 2 Gbps. The highest data rate provided by ''load maximization" test is 10 times higher that the most common ones and stays around 109 Gbps.

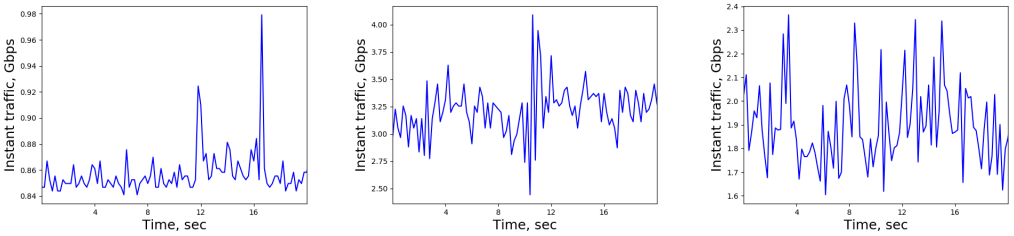

Figure 3.13 *Instant traffic on L3 cache — RAM controller bus in Gbps for 20 seconds interval for encryption test (left), computer game test (middle) and CPU video decoding test (right). All graphs are for 2100 MHz and 8 active cores.*

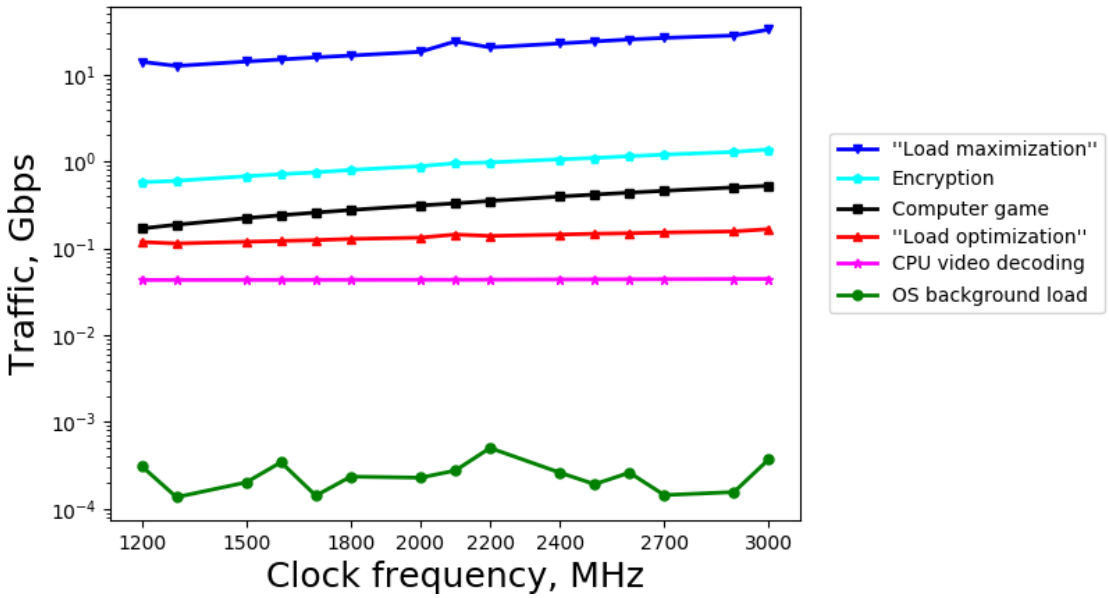

Figure 3.14 *Dependence of total traffic on L3 cache — RAM controller bus on CPU clock frequency for one active core.*

3.5 Analysis of L3 cache — RAM controller bus load

This section is aimed to provide an overview of L3 cache — RAM controller bus load dependencies from time, number of cores, CPU clock frequency and type of load. In the figures 3.10 and 3.11 we have plots of instantaneous traffic on a bus between L3 cache and RAM controller for 4 minutes. The graphs, as usual, provided for all 6 tests: OS background load, “load optimization”, “load maximization”, encryption, computer game and CPU video decoding. As the nature of the graphs doesn’t depend on number of cores and frequency, we decided to show only the graphs for 2100 MHz and 8 cores. For OS background load the graph is randomly oscillating between zero and 0.02 Gbps with most of the graph being between zero and 0.00025

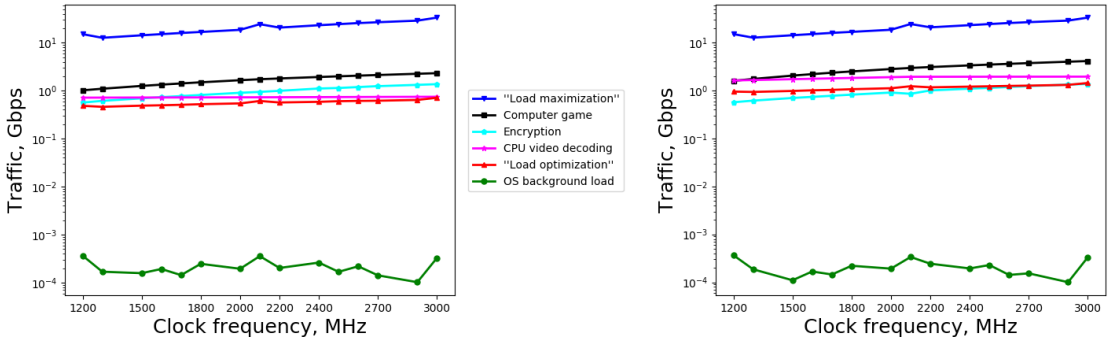

Figure 3.15 *Dependence of total traffic on L3 cache — RAM controller bus on CPU clock frequency for four active cores (left) and eight active cores (right).*

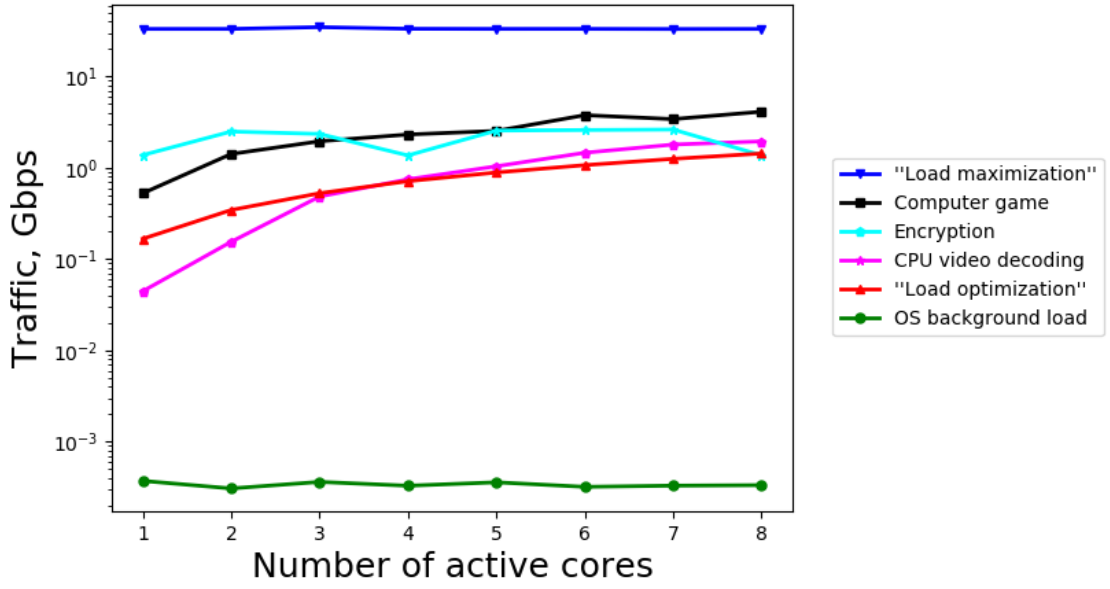

Figure 3.16 *Dependency of total traffic on L3 cache — RAM controller bus on number of active cores for clock frequency of 3000 MHz.*

Gbps. "Load optimization" test is oscillating mainly between 0.13 Gbps and 0.17 Gbps with occasional spikes up to 0.28 Gbps. "Load maximization" looks smoother than previous 2 graphs and is oscillating mainly around 24.25 with occasional spikes up to 27.5 Gbps or down to 23.5 Gbps. Encryption test graph is smooth and stays around 0.85 ± 0.05 Gbps with few spikes to 1 Gbps. Computer game test starts at around 2 Gbps and slowly increases to 3 Gbps with random oscillation of top with most of them being no more than ± 0.5 Gbps away from centerline, occasional spikes can deviate up to 2.5 Gbps away from centerline. For CPU video decoding test unlike the same test case on L2-L3 cache bus there is no periodicity to

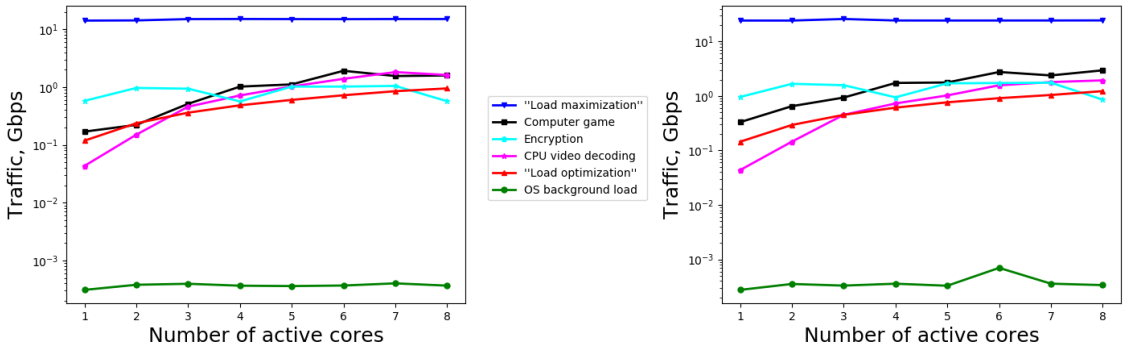

Figure 3.17 *Dependency of total traffic on L3 cache — RAM controller bus on number of active cores for clock frequency of 1200 MHz (left) and 2100 MHz (right).*

be seen as requests are smoothing out when going through several cache layers. It oscillates between 1.6 and 2.6 Gbps with a single downwards spike to 1.2 Gbps. Additional figures 3.12 and 3.13 show the same dependency but zoomed into first 20 seconds. Those versions have the same tendency and ranges as full version but illustrate closer the nature of oscillations. First graph stays mostly at zero with few occasional spikes. Other graphs essentially look like random oscillations except the ''load maximization" one, which shows some periodicity.

In case of L3 cache — RAM controller bus all tests except ''load maximization" show random oscillations. Both zoomed in and out plots give idea of instantaneous traffic behavior for all 6 tests.

On the graphs 3.14 and 3.15 shown the dependency of L3 cache — RAM controller traffic on the CPU frequency. The graphs 3.14 show dependencies for 1 active core, Fig. 3.15 shows same dependency for 4 and 8 active cores. The graphs are logarithmic to better illustrate values at different orders of magnitude, as difference between highest and smallest values is significant. The bottom line (green) is devoted to OS background load that values does not exceed 0.00037 Gbps. It varies in a noise-like level but anyway stays constant across all frequencies. It is the lowest line comparing with all other tests. Most of other lines show linear dependency of the data rates on the frequency but levels differ from test to test. Red line ("load optimization" test) goes from 0.10 Gbps to 0.18 Gbps for 1 core. It is the third-lowest line other that OS background and video decoding. Cyan line corresponding to encryption test goes linearly from 0.58 Gbps to 1.38 Gbps for 1 core. Black line (computer game test) goes from 0.16 Gbps to 0.52 Gbps for 1 active core. The blue one ("load

maximization" test) goes from 14 Gbps to 33.2 Gbps for all number of cores. It is the highest line on the graph. Magenta (CPU video decoding test) is staying constant at 0.04 Gbps for 1 core. This is due to the nature of video decoding test: in our setup it decodes the same amount of video per seconds independently of the CPU frequency. This is also second-lowest line. The graphs on Fig. 3.6 for 4 and 8 cores respectively are similar to 3.5 described one, the only difference is higher loads levels.

We can conclude that as well as on L2-L3 cache bus, load generated by all tests except CPU video decoding test increases linearly with CPU clock frequency increase. CPU video decoding is not linear due to peculiarities of our test stand setup. However, order of plots changed in comparison with L2-L3 cache bus: encryption swapped with computer game and video decoding is swapped with "load optimization" test.

In graphs on Fig. 3.16 and Fig. 3.17 shows dependency of traffic on L3 cache — RAM controller bus on the number of cores. Fig. 3.16 show the dependencies for CPU clock frequency of 3 GHz. Fig. 3.17 is devoted to same dependencies for CPU clock frequency of 1.2 GHz and 2.1 GHz. The most important for our purposed data are depicted on Fig. 3.16 as we need upper bound for our tests load, so all the following values are provided for 3000 MHz CPU frequency. The lowest line is one for OS background load (green) which oscillates around 0.0003 Gbps and bounded by 0.00037 Gbps. The next line is red ("load optimization" test) going linearly from 0.16 Gbps to 1.44 Gbps. Magenta (CPU video decoding) shows similar dependency in similar range from 0.04 Gbps to 1.94 Gbps. Black (computer game) shows the same dependency in the range between 0.52 Gbps for 1 active core and 4.1 Gbps for 8 active cores. Cyan line (encryption) changes in almost the same diapason as computer game, but it is more noisy and bounded by 2.62 Gbps. Blue line ("load maximisation" test) is staying almost constant at 33 Gbps.

Similar to the case of L2-L3 cache bus, load changes linearly for all tests except "load maximization" one with number of cores growth. Also we can get the data rates on L3 cache — RAM controller bus. The values are much smaller that on L2-L3 cache bus, but still significant ones, bounded by 33 Gbps for "load optimization" test.

Overview of the whole situation in general in linear scale is provided on Fig. 3.18. There are several bar plots for each test showing L3 cache — RAM controller bus

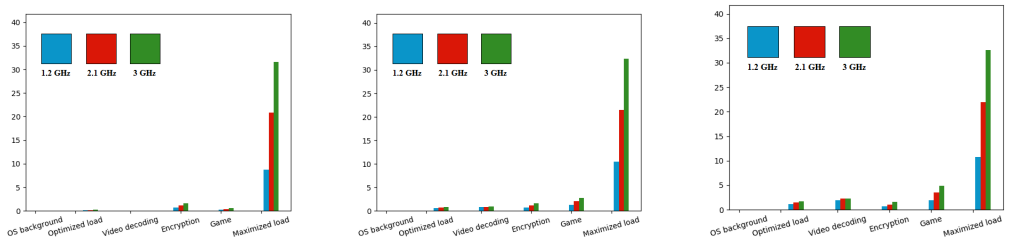

Figure 3.18 *Overall comparison of traffic on L3 cache — RAM controller bus generated by tests for one active core (left), four active cores (middle) and eight active cores (right) for different clock speeds.*

load for all types of tests for minimal (blue), maximal (green) and middle (red) clock frequencies.

It it noticeable that linear increase in clock frequency leads to linear increase of L3 cache — RAM controller bus load. The only exceptions is CPU video decoding due to our stand settings.

The highest data rate of 33 Gbps is generated by "load maximization" test and it is approximately 10 times higher than all other loads. OS background load is negligible in comparison with all other tests. "Load optimization" test together with CPU video decoding test and encryption have roughly similar data rates bounded by 2 Gbps. Computer game generate a bit higher load around 4 Gbps.

Moreover, load on L3 — RAM controller bus increases linearly for all tests that don't face RAM bottleneck. Overall conclusions about measurements in general and their analysis are made in the next section.

3.6 Discussion of obtained results

In this chapter test stand used for measurements is described and software tool called Intel Performance Counter Monitor is detailed. Moreover, its advantages and disadvantages are mentioned. The basics of working with Intel PCM are explained and all issues we faced with are clarified. Full description of all proposed tests is provided as well as our choice of final tests subset. Besides, for the rest of the tests that could be named as typical CPU load but were not chosen for final set of measurements reasons for this decision and tests' troubles explained. Measurements tips and tricks are also provided. Full step-by-step measurement methodology is given. Methodology of L2-L3 cache bus and L3 cache — RAM controller bus loads calculation using number of L2 cache misses and L3 cache misses respectively is provided. Processing of obtained during the measurements results is described.

On L2-L3 cache bus instant traffic oscillates randomly for all tests except video

decoding and ''load maximization" test. On L3 cache — RAM controller bus all tests excluding ''load maximization" oscillate randomly.
According to the obtained results, on both buses data rates increases linearly with increasing number of cores or CPU clock frequency. The only exceptions are "load maximization" test that faces bottleneck in RAM and is bounded due to this reason and CPU video decoding test with CPU clock frequency increase due to our stand settings peculiarities.
We got data rates estimation for L2-L3 cache bus for all tests. Traffic for L2-L3 cache bus is understood as total traffic on the link between own L2 caches of each core and shared L3 cache.
On L2-L3 cache bus the following greatest data rates were observed:

- "Load maximization" test: 109.17 Gbps
- Computer game: 10.9 Gbps
- Encryption: 7.38 Gbps
- CPU video decoding: 4.83 Gbps
- "Load optimization": 1.78 Gbps
- OS background load: 0.02 Gbps

On L3 cache — RAM controller bus the following greatest data rates were observed:

- "Load maximization" test: 33 Gbps
- Computer game: 4.1 Gbps
- Encryption: 2.62 Gbps
- CPU video decoding: 1.94 Gbps
- "Load optimization": 1.44 Gbps
- OS background load: 0.00037 Gbps

Thereby, data rates assessment methodology for internal CPU bus (L2-L3 cache bus) and external CPU bus (L3 cache — RAM controller bus considered as part of bus used for several devices communication including CPU-to-RAM communication) is provided and resulted data rates for both interfaces are mentioned. Dependencies of data rates on both interfaces on CPU clock frequency and number of active cores are

provided, explained and discussed. Obtained results can be used for WNoC development and improvement as well as existing WNoC proposals assessment. Study on assessment of wireless technologies for WNoCs is provided in the following chapter.

4. ASSESSMENT OF WIRELESS TECHNOLOGIES FOR WNOCS

This chapter is devoted to extrapolation of existing results and investigation of wireless communication technologies suitable for WNoC development. Besides, additional requirements for WNoC development found during the research are described.

4.1 Extrapolation of existing results

As it was mentioned before, it is compulsory to get at least the order of magnitude of the inter- and intra-CPU traffic to make a realistic assessment of WNoC proposals. To do that, we need to extrapolate obtained data to the bigger number of cores. Moreover, doing that we should remember about several CPU features.
First one is the fact that changes of clock frequency also affects cache frequency for all levels of CPU cache. As it was explained before, it has an influence on the traffic on both L2-L3 cache bus and L3—RAM (RAM controller) bus. The higher clock speed is, the higher traffic amount on both interfaces. To make more realistic assumptions and trying to find upper bound of inter- and intra-CPU traffic, we will use results for the highest frequency in our measurements that is equal to 3 GHz.
Taking into account results from chapters 3.4 and 3.5 we can notice that for all tests except "maximized load" the nature of dependence is linear. The "maximized load" test doesn't show significant growth with number of cores increase, it stays almost constant for any number of cores. That happens due to the cache bandwidth limitations. All cores works independently, so memory access from cores to L3 cache and respectively RAM is statistically independent, so total amount of cache requests is equal to the sum of requests from each core. However, RAM has limited bandwidths, so only some fixed amount of queries per time can be processed. It becomes a bottleneck and slows down the process. It explains this test's non-linear behaviour.
So finally, in modern CPUs for both L2-L3 cache bus and L3—RAM controller bus traffic on the bus is $min\{load\ generated\ by\ test,\ bus\ capacity\}$.
As we mentioned before, bus capacity and/or cache and RAM bandwidth limitations

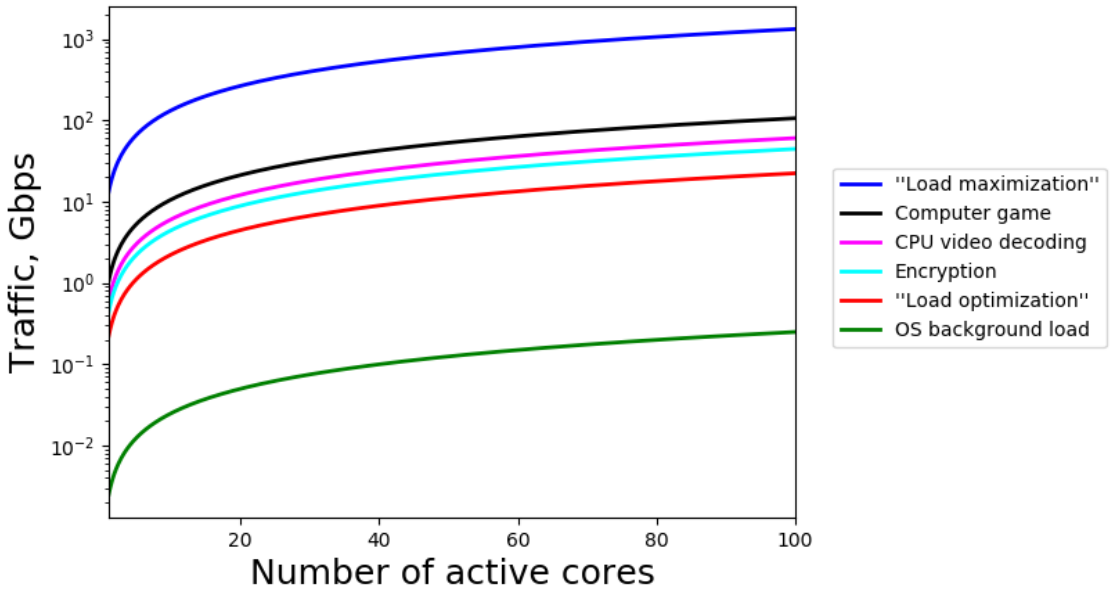

Figure 4.1 *Extrapolation results for 100 cores for L2-L3 cache bus*

make significant issues for CPU development. Thought, this restriction could be removed with using wireless channels for organization of core-to-core, core-to-cache and board-to-board communications. In this case, traffic of both buses of interested can be presented as ***min{load generated by test, wireless link capacity}***.
Due to the linearity nature of the growth of all tests except the "maximized load" one, we can use linear extrapolation to get a preliminary assessment of the load of buses of interest. It is important to mention that last level cache size should be increased proportionally to number of cores increase. It would add some additional costs, but they are negligible in comparison with manufacturing costs for very massive core CPUs. Also it would be important to use special kind of cache organization with fast and efficient search of data, e.g. address hashing.
For extrapolation we used bus loads from tests with 8 active cores and clock frequency of 3 GHz. Extrapolation results for all tests for amount of 1—100 cores and L2-L3 cache bus and L3—RAM controller bus respectively are provided on Fig. 4.1 and Fig. 4.2. It is important to note, that all plots in this and following chapter are presented in a logarithmic scale on axe y.
We can notice that on both interfaces of interest OS background traffic is still small enough and does not exceed several hundreds of megabits for 100 cores. Contrary to this, "load maximization" test gives estimated results over 1000 Gbps for L2-L3 cache bus and several hundreds of gigabits for L3—RAM controller bus. Both values are high enough and can not be reached in modern CPUs in terms of traffic

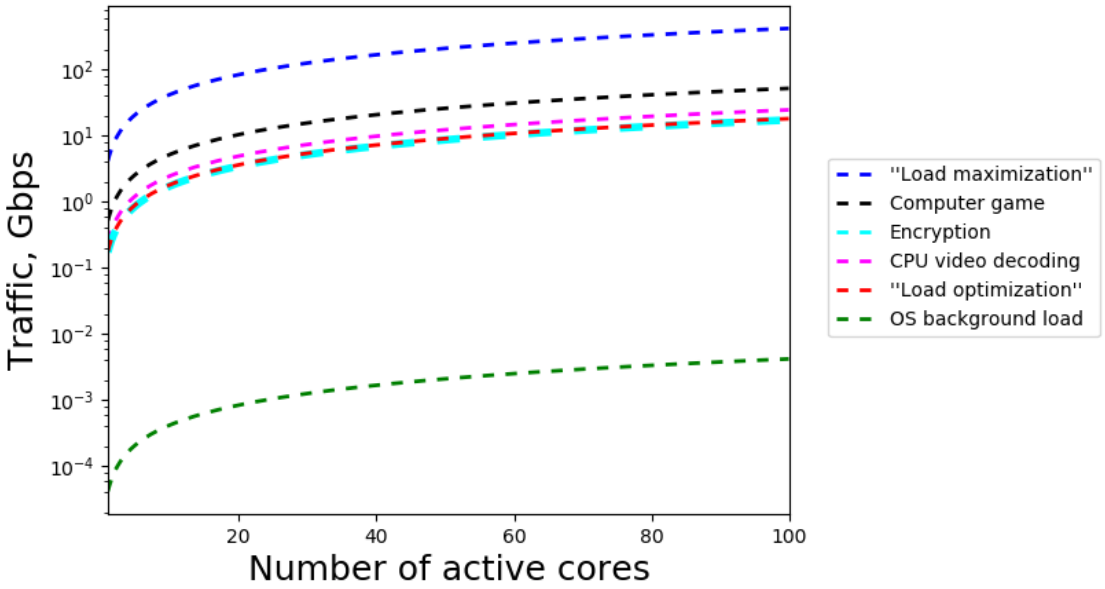

Figure 4.2 *Extrapolation results for 100 cores for L3–RAM controller bus*

amount on internal and external interfaces. All the rest tests' load does not exceed hundred of gigabytes per second for L2-L3 cache bus and several dozens of gigabytes for L3—RAM controller bus. So even modern buses are able to serve this loads level (but only with L3 cache size increase).

After the extrapolation of existing results it is possible to assess what kind of wireless technologies are suitable for WNoCs with different numbers of cores. The following section is aimed to evaluation of different wireless channels for WNoC development.

4.2 Review of suitable wireless channels

There are three communication technologies that are widely proposed to be used for WNoC. They are optics, RF wireless and Terahertz wireless[48], [23], [46], [59]. Use of optical interconnects is widely discussed in papers, especially addressing issues of implementation of transceivers that would be small enough to fit easily into CPU size. It is proposed to use optics (430-770 THz band) and claimed capacity of the link is up to 100 Gbps[59].

Another communication technology that is widely proposed for WNoC utilization is millimeter-wave diapason of radio frequencies, that is usually defined as 30–300 GHz band. It allows to achieve up to 10 Gbps channel capacity, but modulation and coding schemes proposed for achieving claimed rates are complex[59]. It leads to issues with making transceivers and complexity of the whole system.

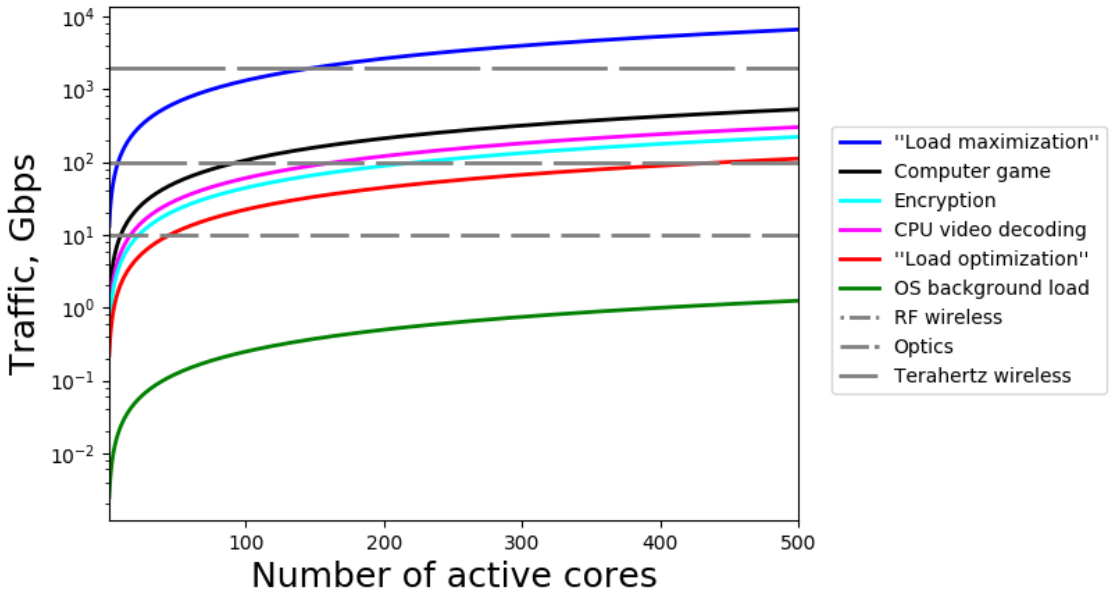

Figure 4.3 *Extrapolation results for 500 cores for L2-L3 cache bus with theoretical links capacity for different communication technologies*

Next step in using wireless high-frequency channels is to use Terahertz band (0.1–10 THz) for WNoCs. Antennas size in this case is small enough to be used in CPUs and claimed link capacity is high even with using simple modulation-coding schemes, e.g. on-off keying (OOK). According to the estimates, it is predicted that link capacity may achieve 2 Tbps with using band of 0.1–0.54 THz, and that results are valid for the distances up to several centimeters[7].

Using aforementioned link capacities for all three communication technologies, we can assess its applicability for WNoC development. All required links capacities made for ideal case scenario. If for some reason the capacity is reduced, the CPU may have lower bottleneck on some test but stays performant anyway. In that case traffic on the buses of interest can be considered as ***min{load generated by test, wireless link capacity}*** and plots behavior would be similar to ones presented on Fig. 3.7— 3.8 and Fig. 3.16— 3.17.

Extrapolation results for L2-L3 cache bus and 1—500 cores as well as links capacity for all three wireless communication technologies are shown on Fig. 4.3. It can be noticed that mmWaves are not suitable for L2-L3 cache bus replacement, as capacity of this technology is small enough even for currently existing CPUs. Optical links also doesn't fit well, but they could be used for special-purposes CPUs, e.g. for solving many tasks with small buses load in parallel and suitable approximately for CPUs with up to 100 cores. Terahertz band looks more promising. According to estimations, it can fulfill requirements on link capacity for all tests including "load

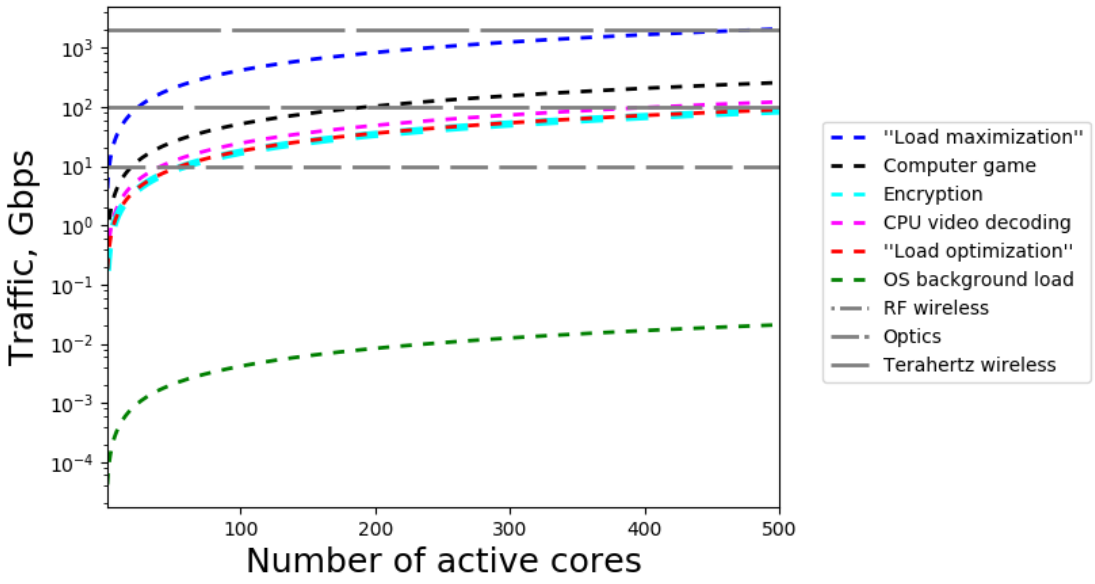

Figure 4.4 *Extrapolation results for 500 cores for L3–RAM controller bus with theoretical links capacity for different communication technologies*

maximization" for up to 150 cores. For more that 150 cores all tests except "load maximization" should work pretty well, but "load maximization" test would saturate wireless link. All other tests reach the limit of 2 Tbps after 500 cores. For higher amount of CPU cores architecture changes most probably would be needed, so it is senseless to estimate boundaries of further applicability of THz band for communication inside CPU with more than 500 cores.

Extrapolation results for L3 cache — RAM controller bus and 1—500 cores as well as links capacity for all three wireless communication technologies are shown on Fig. 4.4. We can notice that mmWaves are suitable for making cache-to-RAM communication for special-purpose CPUs that doesn't need a huge flow of data from RAM for processing many tasks simultaneously. For this kind of constraint CPUs with up to 18 cores can be done. Optic links are useful for CPU-to-RAM communication. For CPUs with up to 23 cores it fulfills capacity requirements for all tests. Also it can be used for special-purposes CPU with low load on L3 cache — RAM bus with up to 190 cores. Terahertz band is also suitable for organization of a wireless link between L3 cache and RAM and it can fit up to 480 cores without any limitations and more than 500 cores with link saturation for some tasks.

Finally, we can conclude that mmWaves are suitable only for CPU-to-RAM communication for CPUs with up to 18 cores. Optical links are useful for both L2-L3

cache and L3 cache — RAM communications. For L2-L3 cache link it can be used for CPUs with up to 100 cores with limitations from above for tests with high load. In case of L3 cache — RAM communication it can be used for CPUs with up to 23 cores without any limitations and for CPUs with up to 190 cores with saturation of links for tests with high link load. Terahertz band is extremely useful for both interfaces also. It can fit up to 150 cores for L2-L3 cache link and 480 cores for L3 cache — RAM link without any limitations for any test type. Moreover, it can fit more than 500 cores on either interfaces, but high load tests would be limited by channels capacity.
Besides, it is important to remember about additional CPU peculiarities that should be taken into account while designing WNoCs. They are discussed in the following section.

4.3 Additional requirements for WNoCs development

During our research we faced with several issues and peculiarities of modern CPUs that should be taken into account while designing WNoCs. Also they are useful for applicability assessment of existing proposals.
First one is a need of very efficient cooling. That might be obvious, but even our tests showed that it is still important. In our stand with 8 cores and proper cooling that was prepared for this kind of troubles we had faced overheating issues. During some tests with high CPU load there was a forced clock frequency decrease due to overheating forced by CPU itself despite of all settings. These tests were run again with temperature decrease in the testing room. It helped as weather was cool and we were able to set low enough temperature in the testing room, however in hot weather it might be not possible. So it is important to take into account proper cooling system while designing WNoCs architectures.
One more important requirement for WNoC design is access time of different cache levels and memory types. We didn't faced it during measurements directly, but it was found during related work study and testing methodology preparation.This criteria is very important for any NoC or WNoC system design, as for the extended CPU architecture based on modern one it is expected to have same or smaller access time for register, all levels of cache, RAM and hard drive. According to the information provided by Intel, access time for different levels of memory hierarchy are has approximately following values[6]:

- Registers — 1 ns
- L1 cache — 5 ns

- L2 cache — 10 ns
- L3 cache — 20 ns
- RAM — 100 ns
- Hard drive — 10 ms

This information should be taken into account while choosing proper modulation and coding scheme as well as MAC protocol for WNoCs development.
Moreover, it is really important to note, that buses load is not constant in time. It changes a lot and rapidly, with drops down and spikes. It is true for both buses, as we can see on Fig. 3.1— 3.4 from section 3.4 and Fig. 3.10— 3.13 from section 3.5. We chosen most stable tests to get an average buses load. Even in this tests there are peaks with buses load increase up to tens of Gbps. In other tests mentioned in section 3.2 that were not chosen for the final set of measurements load is either periodic with very high parts and rapid drops to the small values or has several spikes that can have up to 10 times difference comparing with average traffic. Due to the aforementioned reasons it is important to take into account type of expected buses load in case of special-purpose CPUs. In case of general-purpose CPUs it is important to have some margin in channel capacity that allows to cope with traffic peaks without any harm to the whole system functioning.
Thereby, it is important to take into account proper efficient cooling system for WNoCs to eliminate overheating and clock frequency decrease due to it. All wireless channels should be designed in accordance with resistance to appearance of peaks in instant traffic that can be up to dozens of Gbps. In addition, it is important to provide access time to all parts of memory hierarchy that would be same or smaller than values used in modern CPUs in case when basic CPU architecture wouldn't be changed.

5. CONCLUSIONS

Progress in modern computers development is directly related to the CPUs progress. However modern CPUs have a set of issues that make their development obstructed. These issues are addressed in the future research directions such as WNoC concepts development. However it is necessary to know expected data rates on interfaces eligible to be changed to wireless links. For this purpose, data rates assessment on modern CPUs as well as obtained data extrapolation is necessary.
In this thesis measurements methodology for data rates on internal and external CPU interfaces is provided with detailed explanations of all possible difficulties and unobvious moments. Besides, data rates on internal CPU interface between L2 and L3 cache and external CPU interface between L3 cache and RAM (RAM controller) are assessed, provided in easy to understand form and discussed. Dependencies between data rates on interfaces of interest and CPU clock frequency as well as dependencies between data rates on interfaces of interest and number of CPU cores are provided and explained. Based on it, extrapolation methodology for higher number of CPU cores is provided. Obtained results are applied for WNoC concept to get grounded dependencies between number of cores in WNoC and possible wireless communication technology. Results of the research are published in several papers.
The importance of this task is explained in Chapter 1 together with the considered problem definition and thesis outline. CPU evolution history, modern CPU architecture, open issues in CPU development and research direction on CPU improvement are reviewed in Chapter 2 . Measurements and data rates assessment methodology is described in Chapter 3, where also obtained results are provided in a form of plots and bar plots with full description and peculiarities explanation. Extrapolation results as well as review of suitable wireless channels and additional requirements for WNoC development are provided in Chapter 4.
The most important results of this thesis are the followings:

- Total traffic on L2-L3 cache bus can be up to 109.17 Gbps for 8 active cores
- Total traffic on L3 cache — RAM (RAM controller) bus can be up to 33 Gbps for 8 active cores

- Data rates on both buses of interest increase linearly with CPU clock frequency increase until facing RAM bottleneck (exclude several peculiar tests, e.g. CPU video decoding with special settings)
- Data rates on both buses of interest increase linearly with number of cores increase until facing RAM bottleneck
- mmWaves technology only suitable for CPU-to-RAM organization for small-scale CPUs (up to 18 cores)
- Optical links can be used for both interfaces with limitations above for tests with high loads
- Optical links also could be used for CPU-to-RAM communication for CPUs with up to 23 cores without any bandwidth limitations for all types of tests
- Terahertz band is very useful for wireless links organization on both interfaces and can fit real massive core CPUs without any bandwidth limitations

Possible research directions could be traffic model development using obtained results. This model can be applicable for NoCs and WNoCs development. Moreover, modulation and coding schemes as well as medium access control protocols development would be interesting.

REFERENCES

[1] A. Aiken,U. Banerjee, A. Kejariwal, A. Nicolau, Instruction Level Parallelism, Springer-Verlag US, 2016, 255 p.

[2] Alpha 21164 Microprocessor Data Sheet. Available (accessed on 07.11.17): https://www.cs.cmu.edu/afs/cs/academic/class/15740-f03/public/doc/alpha-21164-data-sheet.pdf.

[3] D. Anderson, J. Delve, Biographies [F.C. Williams; J. Vaucanson; J.M. Jacquard], IEEE Annals of the History of Computing, Vol. 29, Issue 4, pp. 90–102, 2007.

[4] D. Anderson, Tom Kilburn: A Pioneer of Computer Design, IEEE Annals of the History of Computing, Vol. 31, Number 2, pp. 82–86, 2009.

[5] ASUSTeK Computer Inc., X99-A. Available (accessed on 07.11.17): https://www.asus.com/us/Motherboards/X99A.

[6] Y. Baida, Introduction to Microprocessors. Available (accessed on 07.11.17): https://mipt.ru/drec/about/ilab/upload/38c/f_4vj743-arpgiu3hout.pdf.

[7] P. Boronin, V. Petrov, D. Moltchanov, Y. Koucheryavy, J. M. Jornet, Capacity and throughput analysis of nanoscale machine communication through transparency windows in the terahertz band, Nano Communication Networks, Vol. 5, pp. 72–82, 2014.

[8] A.G. Bromley, Charles Babbage's Analytical Engine, 1838, IEEE Annals of the History of Computing, Vol. 20, Issue 4, pp. 29–45, 1998.

[9] P. Ceruzzi, A history of modern computing, second edition, the MIT Press, 2003, 460 p.

[10] A. Chatterjee, Parallelism: In the Cloud, Cluster and Client, 18.12.10, Microsoft Developer Network. Available (accessed on 07.11.17): https://blogs.msdn.microsoft.com/amit_chatterjee/2010/12/18/parallelism-in-the-cloud-cluster-and-client.

[11] D. Chisnall, Understanding ARM Architectures, Pearson Education, Informit. Available (accessed on 07.11.17): http://www.informit.com/articles/article.aspx?p=1620207.

[12] B. Cohen, Howard Aiken, Portrait of a computer pioneer, The MIT Press, 1999, 412 p.

[13] J. Copeland, Colossus: The Secrets of Bletchley Park's Codebreaking Computers, Oxford: Oxford University Press, 2006m 480 p.

[14] J. Cortada, Second Bibliographic Guide to the History of Computing, Computers, and the Information Processing Industry, Vol. 2, Greenwood Publishing Group, 1996, 416 p.

[15] CPUShack.Net, The Life Cycle of a CPU.Available (accessed on 07.11.17): `http://www.cpushack.com/life-cycle-of-cpu.html`.

[16] J. Daemen, V. Rijmen, AES Proposal: Rijndael, 1999.

[17] B. Dang, M. S. Bakir, D. C. Sekar, C. R. Jr. King, And J. D. Meindl, Integrated Microfluidic Cooling and Interconnects for 2D and 3D Chips, IEEE Transactions on Advanced Packaging, Vol. 33, Issue 1, pp. 79–87, 2010.

[18] P. J. Denning, T. G. Lewis, Exponential Laws of Computing Growth, Communications of the ACM, Vol. 60 No. 1, pp. 54-65, 2017.

[19] B. Edwards, Birth of a Standard: The Intel 8086 Microprocessor. Available (accessed on 07.11.17): `https://www.pcworld.com/article/146957/components/article.html`.

[20] N. Enticknap, Computing's Golden Jubilee, Resurrection, the Bulletin of the Computer Conservation Society, Number 20, 1998.

[21] F. Faggin, T. Klein, Silicon-Gate Technology. Solid State Electronics, Vol. 13, pp. 1125–1144, 1970.

[22] Fixups of Loongson2F. Available (accessed on 07.11.17): `https://sourceware.org/ml/binutils/2009-11/msg00387.html`.

[23] A. Ganguly, S. Deb, B. Belzer, Scalable hybrid wireless network-on-chip architectures for multicore systems, IEEE Transactions on Computers, Vol. 60, Issue 10, pp. 1485–1502, 2011.

[24] J. L. Hennessy, D. A. Patterson, Computer Architecture, Fourth Edition: A Quantitative Approach, Morgan Kaufmann Publishers Inc., 2006, 704 p.

[25] The Hercules System/370, ESA/390, and z/Architecture Emulator. Available (accessed on 07.11.17): `www.hercules-390.org`

[26] A. Hodges, Alan Turing: The Enigma (The Centenary Edition), Princeton University Press, 2012, 609 p.

[27] IBM Corporation, Workhorse of modern industry: The IBM 650, IBM Archives. Available (accessed on 07.11.17): https://www-03.ibm.com/ibm/history/exhibits/650/650_intro.html.

[28] IBM Corporation, The S/360: A turning point in mainframe history. Available (accessed on 07.11.17): https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zmainframe/zconc_s360history.htm.

[29] IBM Corporation, IBM Mainframes — 45+ Years of Evolution. Available (accessed on 07.11.17): http://www.vm.ibm.com/devpages/jelliott/pdfs/zhistory.pdf

[30] Intel Corporation, 7th gen intel core and Intel Xeon processor brieng. Available (accessed on 07.11.17): https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/01/7th-gen-intel-core-january-product-brief.pdf.

[31] Intel Corporation, Intel 64 and IA-32 Architectures Software Developer's Manual. Available (accessed on 07.11.17): https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-3pdf.

[32] Intel Corporation, Intel Pentium D Processor. Available (accessed on 07.11.17): https://ark.intel.com/en/products/27520/Intel-Pentium-D-Processor-945-4M-Cache-3_40-GHz-800-MHz-FSB

[33] Intel Corporation, Intel Pentium Pro Precessor. Available (accessed on 07.11.17): https://ark.intel.com/products/49948/Intel-Pentium-Pro-Processor-150-MHz-256K-Cache-60-MHz-FSB.

[34] Intel Corporation, Intel Core i7-2600K Processor. Available (accessed on 07.11.17): https://ark.intel.com/products/52214/Intel-Core-i7-2600K-Processor-8M-Cache-up-to-3_80-GHz.

[35] Intel Corporation, Intel Core i7-5960X Processor. Available (accessed on 07.11.17): https://ark.intel.com/products/82930/Intel-Core-i7-5960X-Processor-Extreme-Edition-20M-Cache-up-to-3_50-GHz.

[36] Intel Corporation, Intel Performance Counter Monitor. Available (accessed on 07.11.17): www.intel.com/software/pcm.

[37] Intel Corporation, Intel 4004 Data Sheet. Available (accessed on 07.11.17): http://datasheets.chipdb.org/Intel/MCS-4/datashts/intel-4004.pdf.

[38] Intel Corporation, Introduction to the 80386 including the 80386 Data Sheet. Available (accessed on 07.11.17): ftp://bitsavers.informatik.uni-stuttgart.de/components/intel/80386/231746-001_Introduction_to_the_80386_Apr86.pdf.

[39] Intel Corporation, Embedded Intel 486 Processor Hardware Reference Manual. Available (accessed on 07.11.17): http://www.pld.ttu.ee/~prj/486dev.pdf.

[40] Intel Corporation, Massive Parallelism for Mission-Critical Applications. Available (accessed on 07.11.17): https://www.intel.com/content/www/us/en/processors/itanium/itanium-9500-massive-parallelism-mission-critical-computing-paper.html.

[41] Intel Corporation, Why has CPU frequency ceased to grow?Available (accessed on 07.11.17): https://software.intel.com/en-us/blogs/2014/02/19/why-has-cpu-frequency-ceased-to-grow.

[42] Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor. Available (accessed on 07.11.17): http://www.hpl.hp.com/hpjournal/dtj/vol7num1/vol7num1art9.pdf.

[43] S. Jacobs, Engineering Information Security: The Application of Systems Engineering Concepts to Achieve Information Assurance, First Edition, John Wiley& Sons, Inc., 2011, 700 p.

[44] Kingston Technology Corporation, Predator DDR4. Available (accessed on 07.11.17): https://www.kingston.com/us/hyperx/memory/predator-ddr4.

[45] Kubuntu devs, Kubuntu 14.10. Available (accessed on 07.11.17): www.kubuntu.org/news/kubuntu-14.10.

[46] X. Li, Survey of Wireless Network-on-Chip Systems. Available (accessed on 07.11.17): http://www.eng.auburn.edu/~agrawvd/THESIS/LI/report.pdf.

[47] J. Marguin, Histoire des instruments et machines Ã calculer, p. 125, 1994

[48] D. Matolak, A. Kodi, S. Kaya, D. Di Tomaso, S. Laha,W. Rayess, Wireless networks-on-chips: architecture, wireless channel and devices, IEEE Wireless Communications, vol. 19, no. 5, pp. 58–65, 2012.

[49] The National Museum of American History, Altair 8800 Microcomputer. Available (accessed on 07.11.17): `http://americanhistory.si.edu/collections/search/object/nmah_334396`.

[50] The New York Times, A Computer Pioneer Rediscovered, 50 Years On, 20th April, 1994.

[51] The New York Times, Bailing out of the mainframe industry, 5th February, 1984.

[52] NVIDIA Corporation, GEFORCE GTX 750. Available (accessed on 07.11.17): `http://www.nvidia.com/gtx-700-graphics-cards/gtx-750`.

[53] Oldcomputers.net, Apple II. Available (accessed on 07.11.17): `http://oldcomputers.net/appleii.html`

[54] Oldcomputers.net, Apple IIGS.Available (accessed on 07.11.17): `http://oldcomputers.net/appleiigs.html`

[55] Oldcomputers.net, Commodore 64. Available (accessed on 07.11.17): `http://oldcomputers.net/c64.html`

[56] Oldcomputers.net, Radio Shack TRS-80. Available (accessed on 07.11.17): `http://oldcomputers.net/trs80i.html`.

[57] Oldcomputers.net, Sinclair ZX Spectrum. Available (accessed on 07.11.17): `http://oldcomputers.net/zx-spectrum.html`

[58] OpenSSL: The Open Source toolkit for SSL/TLS. Available (accessed on 07.11.17): `https://www.openssl.org`

[59] V. Petrov, D. Moltchanov, M. Komar, A. Antonov, P. Kustarev, S. Rakheja, Y. Koucheryavy, Terahertz Band Intra-Chip Communications: Can Wireless Links Scale Modern x86 CPUs?, IEEE Access, Vol. 5, pp. 6095–6109, 2017.

[60] N.Y. Phing, M.N. Mohd Warip, P. Ehkan, R. Badlishah Ahmad, F. Faiz Zakaria, F. Wahida Zulkefli, Towards high performance network-on-chip: A survey on enabling technologies, open issues and challenges, In Proceedings of 3rd International Conference on Electronic Design (ICED), pp. 259-263, 2016.

[61] Perf Wiki. Available (accessed on 07.11.17): `https://perf.wiki.kernel.org/index.php/Main_Page`.

[62] E. Pugh, L. Johnson, J. Palmer, IBM's 360 and early 370 systems, MIT Press, 1991, 844 p.

[63] G. Prasad Acharya, M. Asha Rani, Survey of test strategies for System-on Chip and it's embedded memories, in Proceedings of 2013 IEEE Recent Advances in Intelligent Computational Systems (RAICS), pp. 199–204, 2013.

[64] A. Ramirez, An Overview of Intel's MMX Technology. Available (accessed on 07.11.17): `https://www.linuxjournal.com/article/3244`.

[65] B. Randell (Ed.), The Origins of Digital Computers Selected Papers (3rd edition), Springer-Verlag Berlin Heidelberg, 1982, 582 p.

[66] E. Salminen, A. Kulmala, T. Hamalainen, On network-on-chip comparison, in Proceedings of 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007), 2007

[67] SAMSUNG, SSD 850 PRO 2.5" SATA III 256GB. Available (accessed on 07.11.17): `https://www.samsung.com/us/computing/memory-storage/solid-state-drives/ssd-850-pro-2-5-sata-iii-256gb-mz-7ke256bw/#specs`.

[68] A. Shah, Intel's Itanium, once destined to replace x86 processors in PCs, hits end of line. Available (accessed on 07.11.17): `https://www.pcworld.com/article/3196080/data-center/intels-itanium-once-destined-to-replace-x86-in-pcs-hits-end-of-line.html`.

[69] C. Sterling, Military Communications: From Ancient Times to the 21st Century, ABC-CLIO, 2007, 565 p.

[70] O. Strimpel, The Early Model Personal Computer Contest, the Computer Museum Report, Vol. 17, 1986.

[71] Tachyon Software, System z Instructions Operation Code Tables. Available (accessed on 07.11.17): `http://www.tachyonsoft.com/inst390t.htm`.

[72] Total Annihilation Universe, Total Annihilation Universe. Available (accessed on 07.11.17): `www.tauniverse.com`.

[73] M. Weik, The ENIAC Story. Available (accessed on 07.11.17): `http://www.uwgb.edu/breznayp/cs353/eniac.html`.

[74] M.R. Williams, From Napier to Lucas: The Use of Napier's Bones in Calculating Instruments, Annals of the History of Computing, Vol. 5, Issue 3, pp. 279 – 296, 1983.

[75] B. Wilson, The man who made 'the world's first personal computer'. Available (accessed on 07.11.17): `http://www.bbc.com/news/business-34639183`.

[76] L. Wood, Datapoint: The Lost Story of the Texans Who Invented the Personal Computer Revolution, Hugo House Publishers, 2012, 330 p.

[77] Charles Xavier Thomas de Colmar, California State University, long beach. Available (accessed on 07.11.17): `http://web.csulb.edu/~cwallis/labs/computability/charles_xavier.html`

[78] C. K. Yuen, Computation Procedures of the Chinese Abacus, Published in: IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-4, Issue 1, pp.138 - 142, 1974