Multicore Processor Performance Analysis

Gaurav S Kolhe

George Mason University
Fairfax, Vriginia, 22030
gkolhe@gmu.edu

***Abstract* — Central Process Units (CPUs) are becoming the standard processors of current computing systems design. With the increasing performance requirements, the number of transistors on single chip unit cannot grow exponentially due to the limit area, power, and heat dissipation, etc. Therefore, multicore processor design has become a trend for current processor design. A multicore processor is an integrated computing component composed of several (two or more) CPU cores that can execute the program instructions. The individual cores can execute multiple instructions in parallel, thus significantly increasing the performance of programs which takes advantage of the unique architecture. In this survey, we introduce several aspects to demonstrate a thorough survey for multicore processor, including (1) The Need for Multicore CPU; (2) The Need for Performance Analysis; (3) The Ways of Evaluating multicore CPU Performance; (4) Factors that Affects the Performance; and (5) Multicore Benchmarking. Finally, we will discuss the existed problems, as well as future directions of multicore processor design and give the conclusions of our multicore performance analysis survey.**

Keywords—multicore processor, system design, performance analysis, power management, throughput analysis, benchmarks.

I. INTRODUCTION

According to Moore's Law [1], the number of transistors in a dense integrated circuit doubles about every 18 months. In the past 50 years, Moore's law has set the pace for the modern digital revolution, which is that computing would dramatically increase in power, and decrease in relative cost, at an exponential pace. However, due to many practical issues like power, hear dissipation, etc., such an exponential developing trend as Moore's Law is showing slowed down and predicted to be ending soon [2].

However, there are more and more data intensive as well as computationally intensive applications which demands high-performance processor design, e.g. Big Data, machine learning, and deep learning techniques and applications. As a result, current multi-core processor design has become the alternative to gain more computation resources at a lower design and implementation cost. Such multi-core design inevitably causes undesirably power consumption due to the many cores. Meanwhile, the design of multicore systems also changes the essential design of computer architectures since different designs can cause dramatic differences in the computation performance of multicore processors. For example, the design and utilization of cache and memory subsystems on the multi-core processor, the pipeline depth, etc. can all produce different performance bottleneck once they are improperly designed. Since the manufacturing multicore processors is expensive and takes long time, the careful performance evaluation and benchmarking of multicore processor become more and more essential [3-4]. Specifically, performance evaluation and analysis aim to provide criteria that define the performance and is required at every stage of the computer system lifecycle, to ensure high performance within a given cost.

Meanwhile, it's a common sense that general purpose multicore CPUs are designed for usage in a variety of applications, virtualization, programs and games, and even embedded systems [5]. Such various applications have great diversity and the distinct computation features and thus introduce different challenges to multicore design. Therefore, multicore CPU benchmarking also is a very important task to evaluate whether the designed system performs well under most or all targeted application benchmarks. What's more, given the using scenarios or application types, utilizing benchmark tools can often result in better measurements that become more relevant and accurate by profiling [6-7].

In summary, we explored and included several following aspects of multicore processor performance analysis:

- The need for multicore CPU, e.g. the techniques and computation demand of current popular applications, the trend of current processor design, etc.;
- The need for multicore CPU performance analysis, e.g. what different designs can cause performance boost or degradation;
- Method for evaluating multicore CPU performance, e.g. system profiling, pipeline and throughput analysis, power analysis, etc.;
- Factors that affects the multicore CPU performance, e.g. cache and memory subsystems, multicore communication, etc.;

- The necessity and types of multicore CPU benchmarking, e.g. popular CPU benchmarks and their included application specifications;

Then we will discuss the current existed problems of multicore CPU design, in which we hope to shed some light on future research directions. And finally, we will give the conclusion of the multicore processor performance survey. The remaining parts of this survey follows such structure: In Sec. 2, we will include backgrounds that illustrates the practical demanding reasons of multicore CPU design, as well as the need for performance analysis. In Sec. 3, we will cover techniques for evaluating the multicore CPU performance and analyze the key factors which mostly affects the CPU performance. In Sec. 4, we will introduce the background and motivation of current popular multicore CPU benchmarks and identify different characteristic of different benchmark programs for CPU performance analysis. Finally, in Sec. 5 and Sec. 6, we will discuss currently existed problems & potential future research directions and then conclude our survey.

II. BACKGROUND AND MOTIVATION FOR MUTLCORE PROCESSOR DESIGN

In this section, we will introduce the backgrounds and motivations of multicore processor design.

A. *Backgrounds*

Before the era of multicore processor, computing processors mostly have one single computing unit, also called single core. However, the clock frequency of the single processor, which determines the speed of it, cannot exceed the limit due to the power consumption and heat dissipation within the very limited chip area [2].

Nevertheless, most increasingly popular applications like big data applications, machine learning and deep learning applications appeals for strong computation performance [8-10]. Most of such applications involves large volume of data processing, matrix complication, great parallelism, etc. All of these factors greatly challenge the old-fashioned single-core processor design. As a result, the leading manufacturer of computing processors like Intel came up with a new design of processor, called multicore processor.

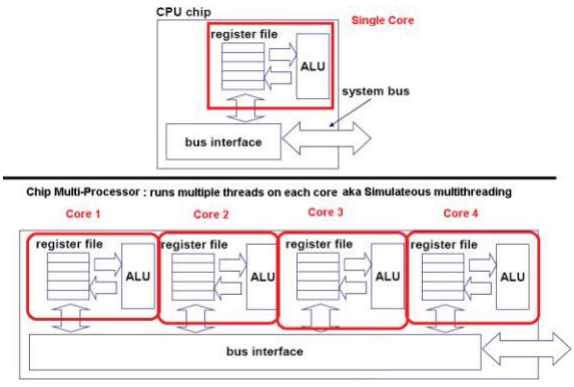

Fig.1 Single Core vs Multicore Processor

A multicore processor is a computer processor integrated circuit with several (two or more) separate processing units, also called cores, each of which can read and execute program instructions parallelly. The conceptual comparison of single core vs multicore processor is shown in Fig. 1. There are also many other popular designs of multicore processor with shared cache system, like shared L3 cache [11]. Currently, multicore processors are already widely used across many application domains, including general-purpose multicore processors, embedded processors, digital signal processing (DSP), and graphics (GPUs). Compared to single-core high-frequency processor, multicore processors have the advantages to run on lower frequency and combine the computation capacity of multicores. Therefore, they could usually achieve higher performance compared to single-core processor, and with lower power dissipation or temperature. Meanwhile, multicore processor changes the way of concurrent program running into real parallel program execution, which also greatly enhance the multi-task and parallelism capability of processors [12].

In next section, we will introduce several motivations in more details for the multicore design from the perspective of processor performance analysis.

B. *Motivation for Multicore Design*

For general-purpose processors, the performance gain from increasing the operating frequency of single processor is becoming very limited due to the power dissipation issue, as we have mentioned before. In more detail, this is due to several following factors:

1) ***The memory wall [13].*** The memory subsystems' speed grows much slower than processor speed. Therefore, there is a quickly increasing gap of speed between processor and the memory systems. The cache system design is motivated by this to hide the latency of memory access, but cannot relieve to the one hundered percentage as long as the speed gap becomes increasingly larger. Therefore, increasing the operating frequency further cannot brings the corresponding gain due to the limit of memory speed.

2) ***The ILP (Instruction Level Parallelism) wall [14]***. In a normal application, it's hard to find enough parallelable instructions in a single instruction stream to "feed" the high performance core, or in other words, to keep the high-performance single core busy. In other words, higher frequency core in such situation will remain idle or under-utilized. Therefore, increasing the frequency will further introduce more under-utilization.

3) ***The power wall [15].*** The price of increasing frequency of single cores is the exponentially increasing power usage, and thus the increasing dissipating heat. Currently, the core frequency has nearly reached the limit of maximum generated heat. And there are many new heat dissipation mechanisms that are currently explored to push the limit further. Before that, the single core frequency cannot exceed the heat dissipation limit, otherwise the chip will physically burned out. This also hinders the performance gain by increasing single-core frequency.

Besides above main factors, there are also some other facotrs that demands the new architecture design in order to deliver regular performance improvement to match the computing requirements of the increasingly computationally intensive applications. As a result, the multicore designs are selected as a favorable alternative to keep increasing the computing performance.

Compared to high-frequency single core architecture, each core in multicore processors usually runs in lower frequency. Therefore, *the memory wall* effect can be effectively handled by the multi-level cache system to hide the memory speed gap. Meanwhile, *the ILP wall* can also be relieved to certain extent since the core frequency is lowered down so the utilization rate will be higher. Lastly, *the power wall* problem, as each core of multicore processor run in lower frequency, the power consumption only gow linearly with number of cores, instead of increasing exponentially with frequency. Meanwhile, the number of cores in current multicore CPU design is still limited to 16 cores. Therefore, the power and heat dissipation can be well controlled.

In summary, the aforementioned problems motivated the manufacturers to find better alternatives to increasing core frequency. As the result, due to its great advantages in the above problems, multicore processor design has now become the new trend of manufactures like Intel and AMD.

III. EVALUTING MULTICORE PROCESSOR PERFORMANCE

A. The Need for Performance Analysis

Despite the popularity and effectiveness of multicore design, the performance gain of multicore design still needs to be carefully analyzed since one careless and impropoer design may cause catrophostic problems for the chip itself or the running applications. Meanwhile, different from software development, the manufacturing of hardware chips usually takes much longer time cycles and requires much higher investment. Once manufactured, the chips cannot be easily modified like the software programs in which bugs or problems can be fixed in an online fashion.

Table. 1 Overview of Performance Analysis Methods

Criteria	Modelling	Simulation	Measurement
Stage	Pre & Post	Pre & Post	Post
Cost	Small	Medium	Large
Accuracy	Low	Moderate	Accurate
Trade-off	Easy	Medium	Difficult
Practical	Yes	Yes	No

All these factors means that during the multicore processor deisgn, the detailed and thorough performance analysis is necessary in all stages, for all components, and with highly accurate analysis.

B. Multicore Performance Analysis Methods

According to [16], there are three main categories of methods for multicore processor performance analysis: (1) Analytical modelling; (2) Simulation tools; and (3) Post-manufacturing Measurement. The overview of three categories is shown in Table. 1.

From the perspective of analysis stage, the first two categories of analysis can be achieved before the chip manufacturing, while the last category of analysis can only be done after the real chip is manufactured. Even though such methods can get the most accurate performance analysis result, but it's usually not practical to produce the real chips without carefully performance analysis at first. Therefore, the first two categories (i.e. the modeling and simulation methods) are more widely used in both research and industry society. Specifically, there are many industrial simulation tools provided by companies for the multicore performance simulation and analysis, e.g. the CoFluent simulation tool provided by Intel for architecture design [17]. Next, we introduce several performance analysis examples in analytic modelling and measurement tools.

1) ***Performance Analytic Modelling:*** The techniques by anatic modelling usually generates predicted performance by formulating problems and solves the equations under certain assumptions. In this way, the created model can predict the processor performance as long as the assumption is satisfied. The predicted performance is then validated by the simulation or the measurements collected to evaluate the quality or accuracy of the modelling.

A well-known multicore performance analysis example is Amdahl's Law [18], which predicted the expected speedup by parallelly running an application:

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

where *S* denotes the speedup, *p* is the parallelable propotion of execution time, and *s* can be regarded as the number of parallelable cores. This parallel speedup law is based on the assumption that the parallel propotion will be evenly runned by *s* parallel cores without any scheduling or switching overhead. And it's upper bounded by *1/(1-p)*. In other words, the parallel speedup only exists in the parallel part of the application, and finally the bottleneck will be the serial part of the application. This analytic model is demonstrated to be quite close to real performance of multicore processor when running highly parallelable machine learning applications and is a popular parallel performance model.

2) ***Measurement Tools for Performance Analsis:*** There are many measurement tools to measure the performance of multicore CPUs. Utilizing these tools effectively, one can obtain many insights into the processor's performance bottleneck as well as its characteristics. Referred to [16], there are several common measurements for performance profiling to analyze the multicore CPU performance.

- ***Elapsed Time.*** The first one is the elasped time or the execution time of running a specific application. In a Linux/Unix environment, the time library <time.h> provides several commands as a straight-forward way to measure the CPU time in a multicore context. Such information can provide an overall performance speedup. However, it cannot gives us much more insights like the time spent on each individual line of code, or even each function calls. Therefore, such high level information is usually used in the thread parallelism level to validate the parallelization performance. But the advantages of elapsed time measurement is that it's simple, and doesn't require any source code or compiling information for support.
- ***GNU gprof. [19]*** Such a tool provides much detailed profiling information for performance analysis. The tool itself is also included within the open-sourced GCC package for programming languages, e.g. C, C++ and Java. Compared to the coarse-grained *elapsed time*, *gprof* could provide more detailed information, like the time spent within individual function calls, and also the function call relationships. But one thing we need to mention is that the time statistics are sampled and thus is subjected to potential statistical inaccuracy. Meanwhile, it's not suitable for multi-threading since it doesn't collect per-thread information.

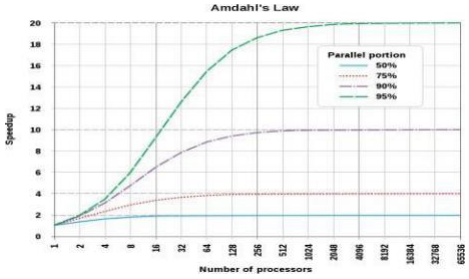

Fig.2 Amdahl's Law in Analytic Modelling [20]

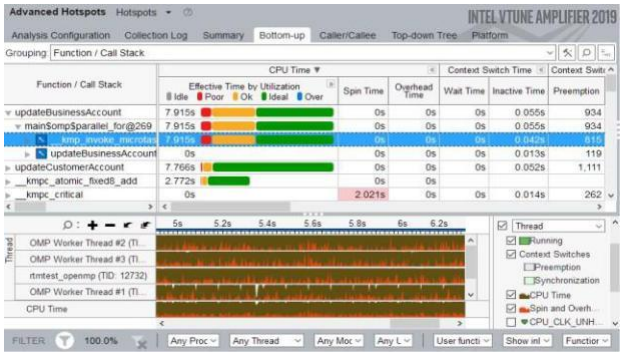

Fig.3 Intel VTune can provide system-wide information to help identify the potential performance bottlenecks.

- ***VTune Profiler. [21]*** Intel VTune is an advanced and powerful commerical performance profiler, which could help identify system configuration problems and find perforamnce bottlenecks. By VTune, we could easily get a lot of useful information about the potential performance bottlenecks including CPU and memory utilization, memory and socket interconnect bandwidth, cycles per instructions, cache miss rates, storage device access metrics, etc. As a result, these metrics provide system-wide data to help you identify if the system—or a specific platform component such as CPU, memory, storage, or network—is under- or over-utilized, and whether you need to upgrade or reconfigure any of these components to improve overall performance [21].

There are also many other tools provided by different libraries for performance analysis. For example, for the machine learning framework TensorFlow [22], Google also provide a detailed profiling mechanism called TF Profie in which users could easily observe the running status of each core, the memory access activities, etc. Due to the space limit, we will not talk into details. In next section, we will talk about the key factors that can siginificantly affect the multicore performance.

C. Factors that Affect the Multicore Performance

There are many factors in multicore CPU design that could affect the multicore performance. In this section, we will give the basic introduction to these concepts and analyze how they can become the performance bottleneck.

- ***Memory Subsystem [23].*** As we have mentioned before, the memory wall exists for high-frequency processors since the speed gap of DRAM and CPU is so huge that memory access can become a significant

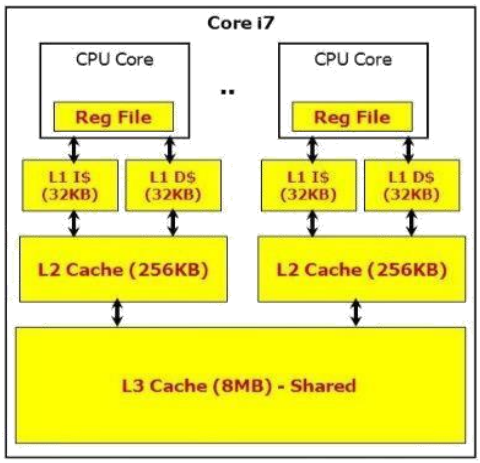

Fig.4 Cache Cohernecy problem may happen when many cores read and write simutaneously.

bottleneck for application. Things become worse when it comes to current data-intensive or data-driven machine and deep learning applications. In summary, to repetitively access the memory, the existed memory bottleneck can affect the machine's performance by slowing down the movement of data between the CPU and the RAM. Then the increased processing times lead to slow computer operations. In other cases, the memory subsystem can also become the performance bottleneck when there are insufficient RAM or not enough memory allocated for the current application, thus memory exchange can consume more time.

- ***Inter-Core Communication [24].*** Even though multicore system could to certain extent alleviate the memory wall, power wall problems, etc., it also brings new problems into the hardware design. The most significant design change is the multiple cores need to communicate with each other when running for the same application. This is not a problem in single high-frequency core architecture. But in the multicore processor, inefficient inter-core communication can cost a lot of time and thus reduce the benefits of multicore parallelism. One alternative of inter-core communication is through the shared cache and memory system, which is the current popular design. But such design also has a critical problem to fix: the cache coherency problem [25]. As shown in Fig.4, since multiple cores can run the application simultaneously, they can also generate data to write or read from the same memory location. Such read/write or write/write can cause numerous conflicts which need to be taken care of. Currently, many design utilizes the OS critical session concept to enforce no such conflicts and thus cache coherency but with the price of idle waiting cycles for other cores.
- ***Application Parallelism [26].*** The application parallelism is also one key impact factor for the multicore performance. This is easy to understand since multicore processor mainly overperforms the single high-frequency process when the application can be extracted a large proportion of parallelized execution codes. Take a common machine learning model training application as an example, when we are training a model with 4 split of data on a Quad-Core processor, it can often easily parallelize all the computations and overperform a similar-frequency single-core system. On the contrary, when the application we run is mostly sequential with tons of data dependencies inside the instructions, multicore processor can suffer from such dense data dependencies and cannot brings actual speedup compared to single-core processor.
- ***Operating System (OS).*** OS is the software-level manager for the physical multicore processor hardware. Therefore, OS's scheduling mechanism also matters a lot for the real performance of a multicore processor.
- ***Others.*** Other factors that can significantly affect the multicore processor performance include the frequency for each core, where higher frequency is usually better. I/O bandwidth can also affect the CPU performance dramatically. Other factors, like number of cores, type of cores (unified types or heterogeneous types of cores) can also influence the practical performance of multicore processor.

IV. BNECHMARKS ANALYSIS

In this section, we will focus on benchmarks analysis for multi-core processors. We first introduce the importance and motivation for using benchmarks for multi-core processor evaluation. Then we investigate a standard benchmark example. At the last, we introduce a case study for benchmark analysis.

A. Motivation for Standard Benchmark

When packing two or more CPU cores on a single processor, the processor's peak performance is supposed to theoretically follow the common Moore's prediction. However, since the same memory subsystem has to support multiple times as many instructions per second as previously, it will significantly influence the multi-core processor's practical performance. In that case, we need to use benchmarks to evaluate the processor's performance before applying a processor design into manufacturing.

Suite	Contents	Metrics
SPECspeed 2017 Integer	10 integer benchmarks	SPECspeed2017_int_base SPECspeed2017_int_peak
SPECspeed 2017 Floating Point	10 floating point benchmarks	SPECspeed2017_fp_base SPECspeed2017_fp_peak
SPECrate 2017 Integer	10 integer benchmarks	SPECrate2017_int_base SPECrate2017_int_peak
SPECrate 2017 Floating Point	13 floating point benchmarks	SPECrate2017_fp_base SPECrate2017_fp_peak

Fig.5 Four suites in SPEC benchmarks.

Benchmarks are designed to allow designers to analyze, test, and improve multicore processors. More specifically, it can measure the efficiency of the different multi-core processors with their different architectures as well as compare their performance to traditional multi-processor systems.

B. Standard Benchmark Example

We will introduce one of the most popular benchmark sets in this part, which is SPEC CPU2017. SPEC CPU2017 focuses on computing intensive performance, which means these benchmarks emphasize the performance in following three aspects:

1. Processor - The CPU chip(s).
2. Memory - The memory hierarchy, including caches and main memory.
3. Compilers - C, C++, and Fortran compilers, including optimizers.

SPEC CPU2017 intentionally depends on all three of the above - not just the processor. In SPEC CPU2017, there are four suits: SPECrate 2017 Integer, SPECspeed 2017 Integer, SPECrate 2017 Floating Point, SPECspeed 2017 Floating Point. Each of them is a set of benchmarks that are run as a group to generate one metric. The four suits are descripted in following figure.

As above figure shows, four suits can be divided into two part according to the performance metrics. The suits named as “SPECspeed” are used to measure the Time: seconds to complete a workload. The suits name as “SPECrate” are used to measure the Throughout: work completed per unit of time. Moreover, SPEC CPU2017 provides 43 benchmarks, which is shown in following figure.

SPECrate 2017 Integer	SPECspeed 2017 Integer	Language[1]	KLOC[2]	Application Area
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1,304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

SPECrate 2017 Floating Point	SPECspeed 2017 Floating Point	Language[1]	KLOC[2]	Application Area
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity
508.namd_r		C++	8	Molecular dynamics
510.parest_r		C++	427	Biomedical imaging: optical tomography with finite elements
511.povray_r		C++, C	170	Ray tracing
519.lbm_r	619.lbm_s	C	1	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	991	Weather forecasting
526.blender_r		C++, C	1,577	3D rendering and animation
527.cam4_r	627.cam4_s	Fortran, C	407	Atmosphere modeling
	628.pop2_s	Fortran, C	338	Wide-scale ocean modeling (climate level)
538.imagick_r	638.imagick_s	C	259	Image manipulation
544.nab_r	644.nab_s	C	24	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	Computational Electromagnetics
554.roms_r	654.roms_s	Fortran	210	Regional ocean modeling

[1] For multi-language benchmarks, the first one listed determines library and link options (details)

[2] KLOC = line count (including comments/whitespace) for source files used in a build / 1000

Fig.6 Detailed Applications in SPEC benchmark.

From Fig. 6 we can find that the above 43 benchmarks involve 24 application areas in total. Each benchmark is the algorithm testing process designed for a specific application area. We take *520.omnetapp_s* as an example. The benchmark performs discrete event simulation of a large 10 gigabit Ethernet network. The simulation is conducted on the OMNeT++ discrete event simulation system. OMNeT++ is a generic and open simulation framework which is basically used for the communication network simulation.

V. Case Study for Benchmark Analysis

In this part, we use a case study to explain how to leverage the benchmarks to evaluate our design. We provide synthetic kernel and natural benchmark results from an HPC system at the NASA Goddard Space Flight Center that illustrate the performance impacts of multi-core vs single core processor systems.

Three platforms are used to running the benchmark, which are Intel 5150 (Woodcrest), Intel5420 (Harpertown), and Intel 7400 (Dunnington). The cores number in each of them are dual cores, quad cores, and quad cores, respectively.

Fig.7 is the cache miss latency comparison results for three platforms. We can find that, with larger memory range, the latency will increase. Moreover, the Harpertown shows smallest latency compared to other two platforms.

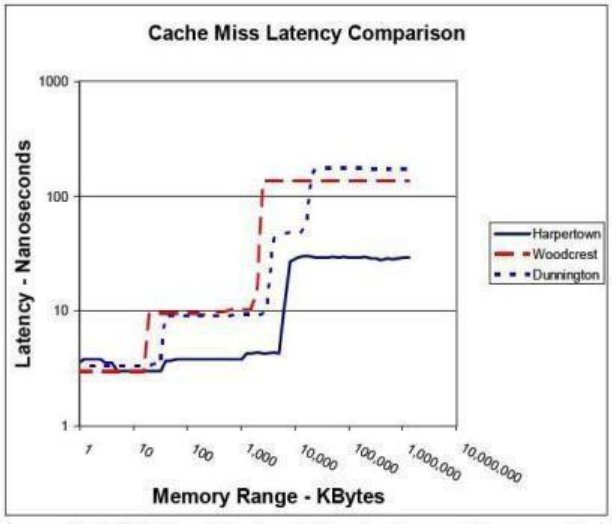

Fig.7 Cache Miss Lantency Comparison for processors with different number of cores.

VI. PORBLESM AND FUTURE WORK

A. *Existing problems*

Aforementioned the multicore processor has many advantages, which can increase the performance of a processor without increasing the clock frequency, by simply adding more cores. However, adding more cores also introduce certain problems or challenges that must be carefully addressed in order to get the benefit from the multicore technology. There are four main problems in current multicore CPU design: (1) power and temperature. (2) the memory hierarchy (3) the level of parallelism. (4) the interconnection.

1) ***Power and Temperature*** As the number of cores placed on a single chip increases, the chip will consume more power resulting in the generation of large amount of heat, which can even cause the chip to become combust. To reduce the unnecessary power consumption, the multicore design also has to make use of a separate power management unit that can manage or control unnecessary wastage of power. The power management unit has to turn off or shut down the cores which don't operate at times or the cores that are not required at times. Also, the cores run at a comparatively lower frequency than a single processor to reduce the power dissipation.

2) ***Memory Hierarchy*** Another problem that is faced by multicore processors is its memory system. There are problems with memory latency and cache coherency. All these factors have an effect on the performance of a multicore CPU design.

Memory latency does not fall as processors get faster, in fact memory latency tends to rise over time with increasing capacity. New types of memory, such as DDR3 or FBDIMMs, both increase latency compared to the previous memory types. Going multicore will increase memory latency further and significantly decrease memory bandwidth.

Another problem is cache coherency because of distributed L1 and L2 cache. Since each core has its own cache, the copy of the data in that cache may not always be the most up-to-date version. For example, in a dual-core processor, one core needs to read from a certain address in memory. Another core could have written data to that address but the data may be stored in cache having not yet made it to RAM. The first core thus needs to access the caches of the other cores if they have data that it requied. As this will happen for every memory read this process will generate an enormous amount of inter-core traffic slowing down both memory and cache access. Since the shared and private caches exist at different levels in multi-core processors, this cache coherence problem becomes worse.

3) ***Level of Parallelism.*** The level of parallelism of the process or application is another big factor that affect the performance of a multicore processor. In the mutilicore design, the parallelism can be achieve by the Instruction Level Parallelism (ILP) and Thread Level Parallelism (TLP). However, most of applications used today were written to run on only a single core processor, failing to use the capability of the multicore processor. Therefore, such parallelism requires the software developers rewrite the applicaitons to be multithreaded.

4) ***Interconnection*** The last important problem which impacts multi-core performance is the interaction between on chip components, (e.g., cores, memory controllers) and shared components, (e.g., cache and memories). Interconnection networks on the muticore chip can have an effect on performance (speed and latency), area, and power consumption.

B. *Future of Multicore Processor*

1) Heterogeneous Multi-Core Architectures To to reduce processor power dissipation, researchers propose and evaluate a heterogeneous set of cores on a single multi-core die, sharing the same ISA. This architecture employ cores of different sizes, organizations, and capabilities, allowing an application to dynamically identify and migrate to the most efficient core, thereby maximizing both performance and energy efficiency.

2) Network on a Chip (NoC) To solve the interconnection issue on the muticore processor, NoC is one of the most promising areas for the development of muticore processor.

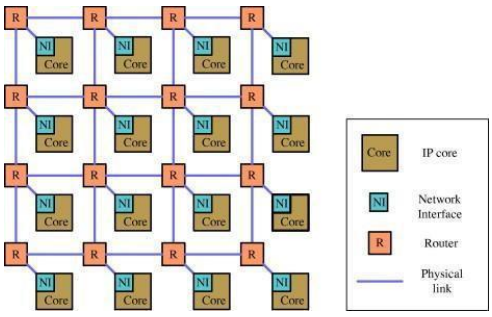

Fig.8 An Example of Basic Network on a Chip.

Prior to NoCs, buses are widely used to provide on-chip interconnects for multicore processors. However, it is hard to provide scalability for modern manycore processors. In contrast, the NoC utilize the on-chip routers to provide multiple path and parallel communications, which can significantlu incresss the throughput of on-chip

communications. As shown in the Figure 8, the processor cores are connected by the routers, the network interfaces and physical links. Routing is to transfer data from source to destination with a clearly defined strategy. In the literature, researchers have classified the routing algorithms according to different criteria.

VII. CONCLUSION

Central Process Units (CPUs) are becoming the standard processors of current computing systems design. With the increasing performance requirements, the number of transistors on single chip unit cannot grow exponentially due to the limit area, power, and heat dissipation, etc. Therefore, multicore processor design has become a trend for current processor design.

Several works studied the performance of data intensive applications on modern processors [27, 28, 29, 30, 31, 32]. All these works use performance counters to the performance and behavior of applications. In [33, 34, 35, 36, 37], authors perform experiments to study the impact of memory on the performance of big data applications. In [38, 39], author uses compress sensing and hardware accelerators to improve IO after finding the performance bottleneck using performance counters. Performance counters can be utilized to monitor applications' behavior to identify a malicious behavior [40, 41, 42, 43, 44, 49, 50, 51, 52, 53, 54]. Moreover, there are new approaches to improve the performance of modern computing systems such as hardware accelerators [45, 46, 47, 48].

In this survey, we introduce many aspects to demonstrate a thorough survey for multicore processor, including (1) The need for multicore CPU; (2) The need for performance analysis; (3) The ways of evaluating multicore CPU performance; (4) Factors that affects the performance; and (5) Multicore benchmarking. Finally, we summarized several currently existed problems in the multicore processor design and identify several future directions in the architecture design. We hope this survey could demonstrate a basic overview of multicore processor performance analysis and hopefully identify some valuable problems and potential future directions for the multicore development.

Reference

[1] Moore's Law, Moore, G. E. (1965). Cramming more components onto integrated circuits.

[2] Esmaeilzadeh, H., Blem, E., Amant, R. S., Sankaralingam, K., & Burger, D. (2011, June). Dark silicon and the end of multicore scaling. In 2011 38th Annual international symposium on computer architecture (ISCA) (pp. 365-376). IEEE.

[3] Chandramowlishwaran, Aparna, Kathleen Knobe, and Richard Vuduc. "Performance evaluation of concurrent collections on high-performance multicore computing systems." 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS). IEEE, 2010.

[4] Gal-On, Shay, and Markus Levy. "Measuring multicore performance." Computer 41.11 (2008): 99-102.

[5] Dixit, K. M. (1991). The SPEC benchmarks. Parallel computing, 17(10-11), 1195-1209.

[6] Guthaus, Matthew R., et al. "MiBench: A free, commercially representative embedded benchmark suite." Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538). IEEE, 2001.

[7] Poovey, Jason A., et al. "A benchmark characterization of the EEMBC benchmark suite." IEEE micro 29.5 (2009): 18-29.

[8] Bengio, Yoshua. "Learning deep architectures for AI." Foundations and trends® in Machine Learning 2.1 (2009): 1-127.

[9] Wu, Xindong, et al. "Data mining with big data." IEEE transactions on knowledge and data engineering 26.1 (2013): 97-107.

[10] Chu, Cheng-Tao, et al. "Map-reduce for machine learning on multicore." Advances in neural information processing systems. 2007.

[11] Chang, Jonathan, et al. "The 65-nm 16-MB shared on-die L3 cache for the dual-core Intel Xeon processor 7100 series." IEEE Journal of Solid-State Circuits 42.4 (2007): 846-852.

[12] Sodan, Angela C., et al. "Parallelism via multithreaded and multicore CPUs." Computer 43.3 (2010): 24-32.

[13] Wulf, Wm A., and Sally A. McKee. "Hitting the memory wall: implications of the obvious." ACM SIGARCH computer architecture news 23.1 (1995): 20-24.

[14] Jouppi, Norman P., and David W. Wall. Available instruction-level parallelism for superscalar and superpipelined machines. Vol. 17. No. 2. ACM, 1989.

[15] Villa, Oreste, et al. "Scaling the power wall: a path to exascale." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, 2014.

[16] Ismail, Dali. "Multicore Processor Performance Analysis-A Survey."

[17] Barreteau, Anthony. "System-Level Modeling and Simulation with Intel® CoFluent™ Studio." Complex Systems Design & Management. Springer, Cham, 2016. 305-306.

[18] Hill, Mark D., and Michael R. Marty. "Amdahl's law in the multicore era." Computer 41.7 (2008): 33-38.

[19] Fenlason, Jay, and Richard Stallman. "GNU gprof." GNU Binutils. Available online: http://www. gnu. org/software/binutils (accessed on 21 April 2018) (1988).

[20] https://en.wikipedia.org/wiki/Amdahl%27s_law

[21] Reinders, James. "VTune performance analyzer essentials." Intel Press (2005).

[22] Abadi, Martín, et al. "Tensorflow: A system for large -scale machine learning." 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016.

[23] McCalpin, John D. "Memory bandwidth and machine balance in current high performance computers." IEEE computer society technical committee on computer architecture (TCCA) newsletter 2.19–25 (1995).

[24] Baumann, Andrew, et al. "The multikernel: a new OS architecture for scalable multicore systems." Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009.

[25] Kurian, George, et al. "ATAC: a 1000-core cache-coherent processor with on-chip optical network." Proceedings of the 19th international conference on Parallel architectures and compilation techniques. ACM, 2010.

[26] Jeffers, Jim, and James Reinders. High performance parallelism pearls volume two: multicore and many-core programming approaches. Morgan Kaufmann, 2015.

[27] Makrani, Hosein Mohammadi, et al. "Evaluation of software-based fault-tolerant techniques on embedded OS's components." Proceedings of the International Conference on Dependability (DEPEND'14). 2014.

[28] Makrani, Hosein Mohammadi, et al. "Energy-aware and Machine Learning-based Resource Provisioning of In-Memory Analytics on Cloud." SoCC. 2018.

[29] Sayadi, Hossein, et al. "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures." 2017 IEEE International Conference on Computer Design (ICCD). IEEE, 2017.

[30] Malik, Maria, Dean M. Tullsen, and Houman Homayoun. "Co-Locating and concurrent fine-tuning MapReduce applications on microservers for energy efficiency." 2017 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2017.

[31] Malik, Maria, et al. "ECoST: Energy-Efficient Co-Locating and Self-Tuning MapReduce Applications." Proceedings of the 48th International Conference on Parallel Processing. ACM, 2019.

[32] Sayadi, Hossein, et al. "Power conversion efficiency-aware mapping of multithreaded applications on heterogeneous architectures: A comprehensive parameter tuning." 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2018.

[33] Makrani, Hosein Mohammadi, et al. "Understanding the role of memory subsystem on performance and energy-efficiency of Hadoop applications." 2017 Eighth International Green and Sustainable Computing Conference (IGSC). IEEE, 2017.

[34] Makrani, Hosein Mohammadi, and Houman Homayoun. "MeNa: A memory navigator for modern hardware in a scale-out environment." 2017 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2017.

[35] Makrani, Hosein Mohammadi, and Houman Homayoun. "Memory requirements of hadoop, spark, and MPI based big data applications on commodity server class architectures." 2017 IEEE International Symposium on Workload Characterization (IISWC). IEEE, 2017.

[36] Makrani, Hosein Mohammadi, et al. "A comprehensive memory analysis of data intensive workloads on server class architecture." Proceedings of the International Symposium on Memory Systems. ACM, 2018.

[37] Makrani, Hosein Mohammadi, et al. "Main-memory requirements of big data applications on commodity server platform." 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). IEEE, 2018.

[38] Namazi, Mahmoud, et al. "Mitigating the Performance and Quality of Parallelized Compressive Sensing Reconstruction Using Image Stitching." Proceedings of the 2019 on Great Lakes Symposium on VLSI. ACM, 2019.

[39] Makrani, Hosein Mohammadi, et al. "Compressive Sensing on Storage Data: An Effective Solution to Alleviate I/0 Bottleneck in Data-Intensive Workloads." 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 2018.

[40] Sayadi, Hossein, et al. "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification." 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). IEEE, 2018.

[41] Sayadi, Hossein, et al. "Customized machine learning-based hardware-assisted malware detection in embedded devices." 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE, 2018.

[42] Sayadi, Hossein, et al. "Comprehensive assessment of run-time hardware-supported malware detection using general and ensemble learning." Proceedings of the 15th ACM International Conference on Computing Frontiers. ACM, 2018.

[43] Dinakarrao, Sai Manoj Pudukotai, et al. "Lightweight Node-level Malware Detection and Network-level Malware Confinement in IoT Networks." 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019.

[44] Sayadi, Hossein, et al. "2SMaRT: A Two-Stage Machine Learning-Based Approach for Run-Time Specialized Hardware-Assisted Malware Detection." 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019.

[45] Neshatpour, Katayoun, et al. "Design Space Exploration for Hardware Acceleration of Machine Learning Applications in MapReduce." 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2018.

[46] Makrani, Hosein Mohammadi, et al. "XPPE: cross-platform performance estimation of hardware accelerators using machine learning." Proceedings of the 24th Asia and South Pacific Design Automation Conference. ACM, 2019.

[47] Neshatpour, Katayoun, et al. "Architectural considerations for FPGA acceleration of Machine Learning Applications in MapReduce." Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation. ACM, 2018.

[48] Makrani, Hosein Mohammadi, et al. "Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis Design." arXiv preprint arXiv:1907.12952 (2019).

[49] G. Kolhe, H. M. Kamali, M. Naicker, T. D. Sheaves, H. Mahmoodi, S. M. PD, H. Homayoun, S. Rafatirad, and A. Sasan. Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality. In IEEE/ACM Int'l Conference on Computer-Aided Design (ICCAD), pages 1–8, 2019.

[50] G. Kolhe, S. M. PD, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun. On custom lut-based obfuscation. In Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI), pages 477–482, 2019.

[51] V. Venugopalan, G. Kolhe, A. Schmidt, Y. Hu, P. Beerel, P. Nuzzo, J. Monson, M. French on Quantifying Security and Overheads for Obfuscation of Integrated Circuits in Government Microcircuit Applications and Critical Technology Conference, 2019.

[52] Sanket Shukla, Gaurav Kolhe, Sai Manoj P.D., Setareh Rafatirad on RNN-based Classifier to Detect Stealthy Malware using Localized Features and Complex Symbolic Sequence, International Conference on Machine Learning and Applications. ICMLA 2019.

[53] Sanket Shukla, Gaurav Kolhe, Sai Manoj P.D., Setareh Rafatirad on Stealthy Malware Detection using RNN-based Automated Localized Feature Extraction and Classifier, in International Conference on Tools and Artificial Intelligence. ICTAI 2019.

[54] Sanket Shukla, Gaurav Kolhe, Sai Manoj P.D., Setareh Rafatirad on Micro-architectural Events and Image Processing-based Hybrid Approach for Robust Malware Detection" in International Conference on Compilers, Architecture,and Synthesis for Embedded Systems. CASES 2019.