Nearly Symmetric Multi-Core Processors

Stefan Reif, Benedict Herzog, Fabian Hügel, Timo Hönig, Wolfgang Schröder-Preikschat
Friedrich-Alexander-Universität Erlangen-Nürnberg
{reif,benedict.herzog,fabian.huegel,thoenig,wosch}@cs.fau.de

ABSTRACT

Multi-core processors are commonplace and continue to require rethinking (not only) in system software development. It is still difficult to operate several functionally identical computing cores efficiently. One misconception is to assume that functionally identical cores of a multi-core processor will behave non-functionally alike, especially at the speed at which they execute the same non-sequential program. We show that considerable deviations in the non-functional behaviour of otherwise identical cores are anything but unusual, and can be expected to vary by more than 20 %. The paper documents the applied measurement methodology, discusses measurement results obtained, and addresses consequences for the coordinated operation of logically connected concurrent threads in the context of Linux.

CCS CONCEPTS

• **Computer systems organization** → **Multicore architectures**; • **Software and its engineering** → *Software performance*; *Operating systems.*

KEYWORDS

Multi-core Processors, Symmetric Multiprocessing, Performance Variation, Operating Systems

ACM Reference Format:
Stefan Reif, Benedict Herzog, Fabian Hügel, Timo Hönig, Wolfgang Schröder-Preikschat. 2020. Nearly Symmetric Multi-Core Processors. In *11th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '20), August 24–25, 2020, Tsukuba, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3409963.3410486

ACM ISBN 978-1-4503-8069-0/20/08...$15.00
https://doi.org/10.1145/3409963.3410486

1 INTRODUCTION

Multi-core processors have conquered the domain of any-purpose computing. The technological change towards the multiplication of (identical/different) processing units integrated on a single processor chip was mainly due to diminished performance gains in face of increasing operating frequency, a phenomenon that was largely called forth by handicaps, such as memory [31], instruction-level parallelism, and power, summarised as *brick wall* [2, 19]. This, in turn, caused software to be confronted with the problem of exploiting the explicit parallelism made available by the hardware for being processed with good or even high-performance. An undertaking that evoked and still brings "dire straits" to general-purpose computing, but appears to be much more challenging for special-purpose computing in the domain of time-dependent systems.

The constraint of real parallelism not only intensified the already existing problem of non-deterministic operation, but it also amplified interference with scheduling decisions not only because of more difficult synchronisation methods required to ensure consistent operation in the data as well as time domain. The difficulties go beyond that and are increased by hardware-related shallows. For example, current operating systems for multi-core processors usually assume that a group of functionally identical cores also behave the same in a non-functional way. It is believed that equally clocked cores will also result in (nearly) identical runtimes of the same non-sequential program section for the individual concurrent threads. Among others, this assumption applies to Linux and is deceptive—or the systems are just not aware of possible deviations. Our investigations show deviations of more than 20 %. The noise that thus represents jitter can cause significant disruptions in the coordinated processing of logically connected concurrent threads, for example in the case of co-scheduling [23] or barrier synchronisation [4, 29]. As a result, strictly locally operated phase-locked schedulers [25] can get out of hand and run the risk of being unable to comply with the supposedly predictable threads based on their operation principle.

Based on measurement series (i.e., performance and power demand measurements) on systems with different "symmetric" multiprocessing (SMP) CPUs we show that apparently identical SMP cores vary greatly. The contributions of this paper are three-fold. First, we reveal significant performance differences between seemingly identical processor cores in a broad variety of COTS hardware. Second, we present the `golden` suite, a run-time system to exploit these performance differences. Third, we discuss how to extend operating systems, such as Linux, for heterogeneity awareness of *nearly* symmetric multi-core processors. The knowledge about effectively different core performance allows for (a) better load balancing by migrating urgent tasks to fastest cores, (b) improved energy efficiency by slowing down cores selectively to re-balance their speed, and (c) improved predictability of program execution times.

The paper is structured as follows. Section 2 provides background and discusses related research. Based on measurement series on several systems, we reveal in Section 3 that SMP cores are not completely equal. In Section 4 we present a run-time support system which considers the heterogeneity of SMP cores and discuss a corresponding implementation for the Linux kernel. The evaluation in Section 5 compares our run-time support system, which is aware of the heterogeneity of SMP cores, to the state of the art, which is unaware of the fact that SMP cores are unequal.

2 BACKGROUND AND RELATED WORK

Variances in processors exist in vast extents and are reasoned for and addressed in various distinct technological areas. In this section we particularly discuss related work mostly from the hardware perspective to identify the root causes of heterogeneity within SMP cores.

2.1 Semiconductor Variability

Allegedly identical chips show variable characteristics (i.e., different timing and power properties) for different reasons. Among such reasons are transistor ageing and fabrication tolerances during production. Transistor ageing [17] occurs as a result of various physical phenomena (e.g., hot-carrier injection, bias temperature instability, and oxide breakdown). Therefore, it is common to face a certain degree of wear for CMOS chips over their lifetime. In addition to wearout, the fabrication variations and the continuous shrinkage of transistor sizes increases the chip variability. Within-die fabrication variations [5, 6, 15, 30] are responsible for differing performance characteristics of individual CPU cores. For example, the channel length and non-uniformities in the creation of interconnect layers of CMOS circuits affect the maximum clock frequency.

2.2 Integrated Circuit Design

Power leakage variations and the thermal distribution within a processor die influence the performance of individual CPU cores [3, 21]. Increased power leakage leads to higher thermal dissipation and consequently prevents higher frequencies in order to avoid exceeding the thermal design power limit. Speed binning is a typical procedure of hardware manufacturers to sell processors capable of higher maximum frequencies for higher prices. Sartori et al. analyse different speed binning metrics for multi-core processors and discuss the influence of varying maximum frequencies per core on the speed binning metric [26]. However, differences in hardware do not only occur within a processor die or architecture, but also for other types of hardware, for example, in hard drive disks [18], wireless sensor nodes [16], and systems on chip [27]. This work analyses the variability of different systems on a chip of the same type. Our work, in contrast, focusses on the identification and utilisation of supposedly homogeneous processor cores in symmetric multi-core processors that actually show different characteristics (i.e., different timing and power properties). The development of new hardware architectures bears the risk that functionally equivalent processor cores show different non-functional characteristics [8] . At the same time the number of design bugs increases with the rising complexity of processors [7] eventually leading to an increased probability of core-to-core performance differences caused by design decisions in the hardware architecture.

2.3 Processor speed variability

Some papers (e.g., [1, 11, 20]) have identified run-to-run, core-to-core, and processor-to-processor performance variation in supercomputers. In addition, Marathe et al. [20] have found that the variability tends to increase with new processor generations. This paper extends these findings in various ways. First, we consider laptop, desktop, and server processors, listed in Table 1. We find performance variations in all of these three processor classes. Second, we show that the amount of performance heterogeneity is workload-dependent. While most applications have little core-to-core performance variations, some workloads benefit greatly from running on a specific core. Third, we demonstrate that single-core applications can benefit from heterogeneity awareness, and implement a prototypical run-time system.

3 HETEROGENEITY OF SYMMETRIC MULTI-CORE PROCESSORS

As a motivating example, Figure 1 visualises the execution time distributions of the sequential `cg` benchmark, which is part of the NAS benchmark suite [12], with problem size `A` on a desktop processor (`CPU1`, cf. Table 1). It shows that *some cores are faster than others*[1] and the run-to-run variation cannot explain the core-to-core differences. Considering that

[1]Text in italics indicates conclusions drawn from our measurement results.

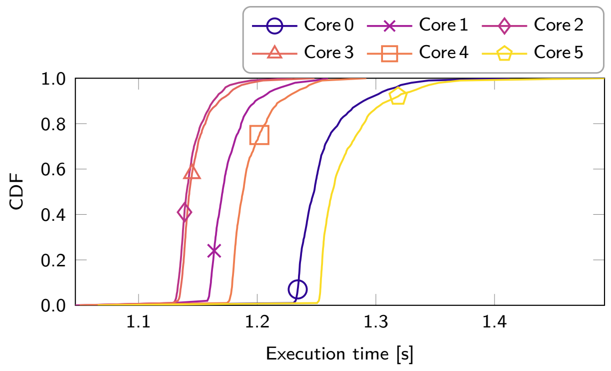


Figure 1: CDF of the execution time of the cg benchmark, per core (CPU1).

the mean execution time on Core 5 is $> 10\,\%$ higher than Core 2, this section examines in detail which workloads and what processors are affected by this type of performance heterogeneity.

3.1 Workload-Specific Behaviour

We run these measurements with the sequential versions of the NAS Parallel Benchmarks, since they constitute representative multi-core workloads, but allow for a performance assessment of individual cores. Figure 3 shows that for all NAS benchmarks, except for cg, the core differences are negligible[2]. This means that *the performance differences depend on workload characteristics* which is a strong indicator that *the measured performance differences are caused by the hardware, not the system-level software running on it (e.g., interrupts or system noise).* We argue that the performance differences of different processor cores observed for the cg benchmark are worth considering. As part of a highly-relevant benchmark suite (i.e., NAS Parallel Benchmarks) the code mimics the behaviour of real-world applications. In particular, the cg benchmark contains program code with irregular memory accesses and communication patterns [12]. We conclude that *not all workloads are sensitive to core heterogeneity*, but *some workloads can run $> 10\,\%$ faster if executed on the right core.* Thus, the processor offers free (performance) lunch [28] for some applications.

3.2 Correlation with Energy

For additional insight to the microarchitectural behaviour, we measure the energy demand via the RAPL interface. Figure 4a shows that the power demand correlates negatively

[2]We use problem sizes A and S and ignore the dt benchmark because it has no sequential reference implementation; and the bt, dc, and lu benchmarks because a single execution of the sequential version took too much time to allow for a large number of iterations. The dataset is nevertheless sufficient for the conclusions that we draw in this paper.

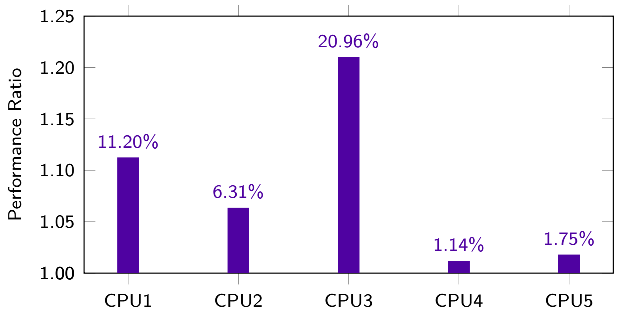


Figure 2: Mean execution time of the cg.A benchmark on the slowest core, normalised to the fastest core of each processor.

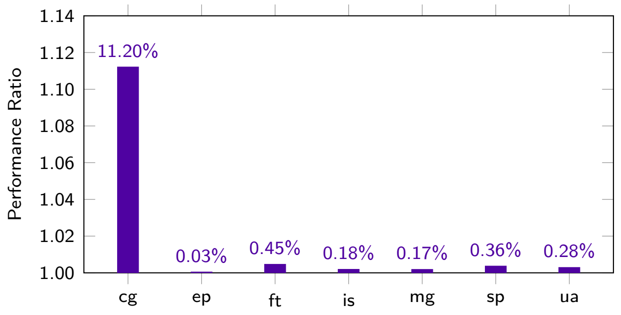
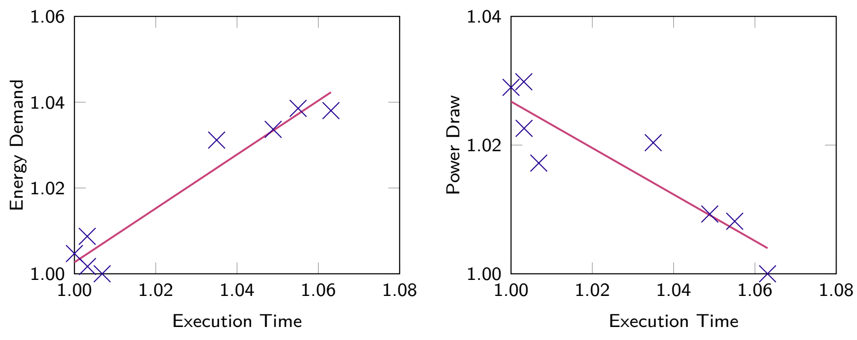


Figure 3: Mean execution time of NAS Parallel Benchmarks on the slowest core, normalised to the fastest core (CPU1).

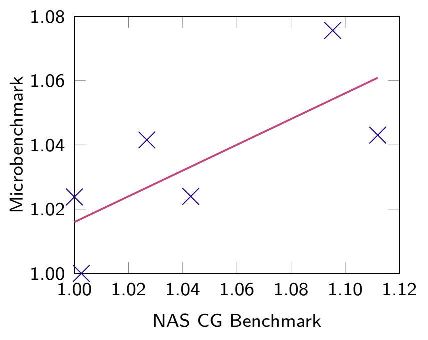
with the execution times (i.e., slower cores tend to have a lower power demand), but the energy demand shows a positive correlation (i.e., slower cores tend to consume more energy). However, RAPL applies energy models internally, and the technical reference manuals do not provide the information whether these models account for core-specific characteristics. We are therefore very cautious and only conclude that *there is no evidence that slower cores need less energy*, and that *the activity below the hardware surface varies.*

3.3 Causes of Heterogeneity

Considering that the cg benchmark is cache-intensive, we develop several microbenchmarks that stress the caches. Listing 1 shows one of these cache-stress microbenchmarks. Figure 4b visualises the result, indicating that the execution time of the cg benchmark correlates positively with the execution time of this microbenchmark. However, the proportions do not match exactly, and the order of cores is slightly off. In consequence, it is possible that the fastest core for one workload is not optimal for another workload. Further microbenchmarks that we developed and tested show similar



(a) The execution time correlates positively with the energy demand, and negatively with the average power demand (CPU2).



(b) The cg and microbenchmark execution times correlate positively (CPU1).

Figure 4: Correlation of the cg execution time with energy and power demand, and microbenchmark execution time (all values normalised, one point per core).

Listing 1: A cache-stress microbenchmark.

```
void microbenchmark() {
  char mem[N];
  for (r = 0; r < R; r++) { // repeat ...
    for (n = 0; n < N; n += sizeof(cacheline)) {
      mem[n]++;
      clflush(&mem[n]);
    }
    mfence();
  }
}
```

results, we therefore omit them in this paper for space reasons. This means that *the cache is, at least partially, responsible for the performance differences*—the cache-intensive microbenchmark triggers performance differences between processor cores. In fact, the last-level cache on recent Intel microprocessors is not fully uniform [13], which could be one potential reason for the observed performance differences. However, *another mechanism possibly causes additional performance differences* because the performance differences vary between cg and the microbenchmark.

In consequence, identification of the microarchitectural causes of core-to-core performance differences, and detection of sensitive workloads, demands for a more detailed analysis. However, we consider suitable workload classification techniques as future work. Instead, this paper examines the behaviour of sensitive workloads more closely.

3.4 Processor Model Comparison

Figure 2 displays the mean execution time of the cg benchmark on the slowest core, for multiple computers, each normalised to the mean [14] execution time on the corresponding fastest core. The processor details are listed in Table 1. The CPU4 and CPU5 machines both only show minimal performance differences. This means that *not all processors exhibit heterogeneous behaviour*. However, *all processor types can be affected*, including desktop (CPU1), laptop (CPU2), and server (CPU3) models. Besides, *the performance differences tend to be stronger on machines with more physical cores*. Remarkably, the server machine CPU3 has a performance difference $> 20\,\%$. Extrapolating these results, and considering the trend that the number of cores has been growing for years, we conclude that the *SMP core heterogeneity will be more prevalent in future (many-core) systems*.

Figure 5 visualises the core performance characteristics of CPU3 in detail. Both axes contain the logical core IDs, as enumerated by Linux (i.e., "physical" cores first, then physical cores on second socket, then their "hyperthreads"). Since CPU3 has 96 logical cores, we can examine in detail to what degree the core performances differ. For this evaluation, we compute the k-sample Anderson-Darling Test ($k = 2$) for each pair of logical cores. This test takes the execution-time distribution of the cg benchmark of each core as inputs, and categorises core pairs either as "similar" or as significantly (p-value is 0.01) "different". The black dots, which indicate that the performance is similar, are most prevalent when comparing a core to itself (i.e., the diagonal line in the centre) or to the hyper-thread that share the same physical core (i.e., the two parallel diagonal lines through the points $(0, 48)$ and $(48, 0)$). This again indicates that *caches are a cause of performance differences*, since a "physical core" and its "hyperthread" share caches. However, *almost all other core pairs exhibit significant performance differences*, even when comparing cores on the same socket. To further examine the majority of significantly different core pairs, we compute the two-sample Kolmogorov-Smirnov test statistics that quantifies the differences between cores. The resulting differences

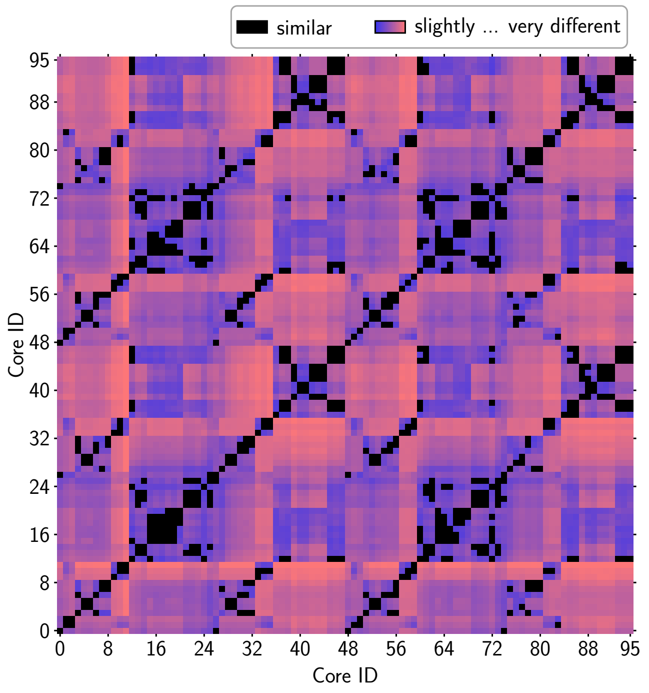


Figure 5: Core similarity matrix of the 96 logical cores of `CPU3`. A black dot represents similar performance, and non-black space indicates a significantly different performance distribution. Colours display the degree of performance difference between cores.

are visualised as a colour gradient in Figure 5, where blue colour represents small differences, and red colour represents large differences. The colours show that *there are clusters of cores with smaller performance differences*, and some clusters repeat with a period of 24 (e.g., cores 9, 33, 57, 81), indicating that *the intra-processor heterogeneity is alike on both sockets.*

4 OPERATING SYSTEM SUPPORT

Our findings in the previous section reveal that processor cores are not exactly identical. As todays operating systems treat SMP cores to be equal, we discuss our implemented prototypical run-time support system and the corresponding integration for the Linux kernel. Our implementation is publicly available under an open-source license:

https://gitlab.cs.fau.de/i4/pub/apsys2020/golden

4.1 Run-time Support System

Motivated by the core performance differences described in Section 3, we have developed a run-time support system to establish awareness for core heterogeneity. It comprises of three tools:

First, the `golden-sample` script executes the cache-stress microbenchmark described in Listing 1 on each core and analyses the results. Second, the `golden-ratio` script displays the results of `golden-sample` in a human-readable format. Thus, the `golden-ratio` script enables the system administrator to find out to which extent the system is affected by core heterogeneity. As shown in Figure 2, the performance differences vary greatly between processor models. Third, the `golden-run` program executes a given application on the fastest processor cores in the system. Thereby, the number of cores is configurable. The system administrator can thus enforce that heterogeneity-sensitive workloads run only on the fastest cores, achieving optimal performance. Linux will then automatically migrate other processes to slower cores. If these processes are not sensitive to heterogeneity, this migration has only a negligible performance penalty.

The `golden`-suite thus helps system administrators to classify their systems and to exploit core heterogeneity. Avoiding slow cores improves performance and reduces tail latencies [9] related to heterogeneity.

4.2 Linux Integration

Our run-time support system has further optimization potential by integrating it deeper into operating systems.

First, we suggest automated *workload classification*. It is helpful to automatically detect which application is sensitive to core heterogeneity. Such an automated system could utilise performance counters to detect performance bottlenecks, and react when the bottleneck causes heterogeneous behaviour (e.g., caches). However, we could not implement this automatic detection because (a) it is hard to detect performance bottlenecks in a portable way and (b) and we do not have complete knowledge about the root causes of heterogeneity. Indeed, our microbenchmark experiments suggest that further causes, besides caches, exist. Therefore, information from hardware vendors is necessary. Second, we suggest automated *workload migration*. Based on automated workload classification, the operating system can migrate heterogeneity-sensitive applications to fast cores. Similarly to our `golden-run` program, the heterogeneity-sensitive application benefits from the migration, but the *victim* that is moved to a relatively slow core has no performance penalty if it is heterogeneity-oblivious. This is the case for most NAS Parallel Benchmarks (c.f. Figure 3). A third useful feature is *noise migration*. For applications where performance consistency matters, the operating system should migrate all *noise* causes to fast cores. It can thus re-balance the relative core

Table 1: Processors used in the evaluation.

Name	CPU	# Cores	
CPU1	Intel i5-8400	6	(1×6)
CPU2	Intel i5-8250u	8	(2×4)
CPU3	Intel 2× Xeon E7-4830 v3	96	$(2 \times 2 \times 24)$
CPU4	AMD R7 1700X	16	(2×8)
CPU5	Intel Xeon E3-1275	8	(2×4)

speed and minimise its disturbance to the application performance.

The Linux kernel has recently integrated *energy-aware scheduling* [22, 24], which exploits different power and performance characteristics of heterogeneous processors. This sub-system is intended for architectures that combine "big" (i.e., fast) and "little" (i.e., slow but more energy efficient) cores. As a consequence, this system assumes that heterogeneity affects all workloads, and that cores within groups are identical. Our findings suggest extensions to this subsystem to improve heterogeneity awareness.

5 EVALUATION

First, this section describes the evaluation environment used in Section 3. Afterwards, it empirically evaluates the golden-suite, the heterogeneity-aware run-time system proposed in Section 4.

5.1 Evaluation Setup

We have conducted all of our experiments on several machines with five different CPUs, which span a large range from a laptop system to a many-core server system. Table 1 summaries the machines used in our experiments. The processors cover a wide range, including models for laptops, desktops, and servers. CPU1 does not provide hyperthreading, and CPU3 is a dual-socket machine.

On all systems, we use the performance frequency scaling governor or policy. All machines have their in-hardware boost technology enabled[3]. We start each experiment with 10 warm-up iterations, per core, which are excluded from the result data set. Afterwards, we repeat every experiment 1 000 times per core. The only exception is the many-core system, CPU3, where we execute only 100 repetitions per core because of the large number of cores (i.e., 96 hardware threads)—the total execution time on this machine of the cg experiment displayed in Figures 2 and 5 was over 6 hours, and Figure 5 shows that the performance differences are statistically significant.

[3]Linux did not recognise the boost option on the AMD machine but we verify that the hardware boosts above nominal clock speed using the cpufreq-aperf tool.

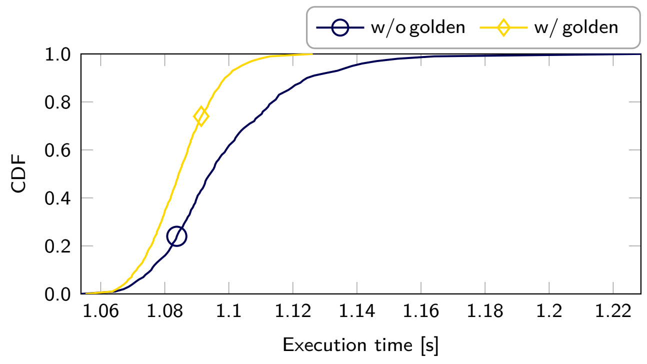


Figure 6: The golden-suite improves execution time, especially the tail latency, for heterogeneity-sensitive applications (CPU1)

5.2 Heterogeneity Awareness

Figure 6 compares the execution time of the sequential cg benchmark on the CPU1 machine with golden-run, compared to the default where Linux is free to choose a processor core for the program. Thereby, golden-run picks the second-fastest core due to discrepancies between the microbenchmark and the cg performance. Our system thus constitutes an imperfect (i.e., realistic) performance oracle.

Using golden-run, the mean execution time is reduced by 1.2 %. In the best case, the two scenarios achieve the same execution time because the OS can place the benchmark on the fastest core by chance. However, the default version has a higher tail latency when the benchmark runs on a slow core. The worst-case (99th percentile) execution time is 4.6 % slower in the default version. Notably, this performance improvement does not require modification of hardware or software components—it is achieved only by being aware of the core-to-core performance differences.

We also execute the same experiment with the is benchmark, which is equally fast on all cores. In consequence, no performance gain is expected, but we critically evaluate the overhead of the golden-run tool. Indeed, the overhead is negligible (mean $\approx$ 0.10 % slower), which matches the latency of the execvp operation within golden-run. There are, however, *indirect costs* when an administrator erroneously pins a core-oblivious program to the fastest core and thus causes a heterogeneity-sensitive application to run on a non-optimal, slower core. Such an erroneous application placement could be avoided by automated workload classification and migration.

5.3 Threats to validity

There are two significant threats to the general applicability of our results to real-world systems.

First, a selection bias is involved. Compared to the billions of processors out in the wild, we conducted our experiments only on a small number of machines. We do not know whether their results are representative for all processors. Our experiments, however, cover a broad spectrum of relevant real-world systems, ranging from laptops to many-core server processors. Regarding the AMD machine, `CPU4`, we do not know whether its result is representative for all AMD processors. For newer AMD processors, the hardware communicates core-to-core performance differences to the operating system [10].

Second, we do not know to which extent real-world applications are heterogeneity-sensitive. Most NAS Parallel Benchmarks do not show performance differences between cores. However, we argue that observed per-core performance differences of $> 10\,\%$ on the desktop processor (`CPU1`) and $> 20\,\%$ on the server machine (`CPU3`) demand for a run-time system that utilises this free (performance) lunch.

6 CONCLUSION

This paper has examined the heterogeneity of individual processor cores in supposedly symmetric multi-core and many-core machines. Our experiments show that, depending on the workload and the machine, the per-core performance can vary by more than $10\,\%$ within a single multi-core processor. On our 96-core server system, the performance difference can even be above $20\,\%$.

We have created microbenchmarks that reveal that caches are one contributor to the performance differences. However, there are additional causes for heterogeneity which we could not identify with our observation-based approach. Accurate per-core and workload-specific performance modelling, potentially based on reverse engineering, is left as future work. As discussed in Section 4, additional information from the hardware vendors would be very helpful to automate workload classification and migration.

Based on our experiment results, we have implemented a run-time support system for core heterogeneity, the `golden`-suite. It improves the mean execution time by $1.2\,\%$ compared to the default Linux process placement. The mean difference is relatively small because, in the best case, Linux unknowingly selects a fast core, resulting in equal performance. In the worst case (99th percentile), the default Linux strategy is $4.6\,\%$ slower on a typical desktop processor. Notably, this difference comes at absolutely no cost—neither hardware nor software components need to be modified. Instead, our system only exploits knowledge about heterogeneity, of which neither developers nor operating systems are aware of. Furthermore, our system's performance could greatly benefit if hardware vendors provide more heterogeneity-related information.

ACKNOWLEDGEMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants no. SCHR 603/15-2 ("e.LARN"), SCHR 603/8-2 ("LAOS"), and Project Number 146371743 – TRR 89 Invasive Computing.

REFERENCES

[1] Bilge Acun, Phil Miller, and Laxmikant V. Kale. 2016. Variation Among Processors Under Turbo Boost in HPC Systems. In *Proceedings of the 30th International Conference on Supercomputing (ICS'16)*. ACM Press, New York, NY, USA, Article 6, 12 pages.

[2] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A Yelick. 2006. *The Landscape of Parallel Computing Research: A View from Berkeley.* Technical Report UCB/EECS-2006-183. Electrical Engineering and Computer Sciences, University of California at Berkeley.

[3] Maryam Ashouei, Abhijit Chatterjee, Adit D Singh, Vivek De, and TM Mak. 2006. Statistical Estimation of Correlated Leakage Power Variation and Its Application to Leakage-Aware Design.. In *VLSI Design*. IEEE Computer Society Press, New York, NY, USA, 606–612.

[4] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, and Susan Coghlan. 2006. The Influence of Operating Systems on the Performance of Collective Operations at Extreme Scale. In *Proceedings of the 8th Annual International Conference on Cluster Computing (ICCC'06)*. IEEE Computer Society Press, New York, NY, USA, 1–12.

[5] Shekhar Borkar, Tanay Karnik, Siva Narendra, Jim Tschanz, Ali Keshavarzi, and Vivek De. 2003. Parameter Variations and Impact on Circuits and Microarchitecture. In *Proceedings of the 40th Annual Design Automation Conference (DAC'03)*. ACM Press, New York, NY, USA, 338–342.

[6] Keith A Bowman, Steven G Duvall, and James D Meindl. 2002. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of solid-state circuits* 37, 2 (2002), 183–190.

[7] Kypros Constantinides, Onur Mutlu, and Todd Austin. 2008. Online design bug detection: RTL analysis, flexible mechanisms, and evaluation. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society Press, Los Alamitos, CA, USA, 282–293.

[8] Weilong Cui and Timothy Sherwood. 2017. Estimating and understanding architectural risk. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM Press, New York, NY, USA, 651–664.

[9] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (Feb. 2013), 74–80.

[10] Advanced Micro Devices. 2020. https://www.amd.com/en/support/kb/release-notes/rn-ryzen-master-2-3-0-1591. Acc. 2020-07-24.

[11] Saurabh Dighe, Sriram Vangal, Paolo Aseron, Shasi Kumar, Tiju Jacob, Keith Bowman, Jason Howard, James Tschanz, Vasantha Erraguntla, Nitin Borkar, Vivek De, and Shekhar Borkar. 2011. Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor. *IEEE Journal of Solid-State Circuits* 46, 1 (Jan. 2011), 184–193.

[12] NASA Advanced Supercomputing Division. 2020. https://www.nas.nasa.gov/publications/npb.html. Acc. 2020-07-24.

[13] Alireza Farshin, Amir Roozbeh, Gerald Q. Maguire, and Dejan Kostić. 2019. Make the Most out of Last Level Cache in Intel Processors. In *Proceedings of the 14th EuroSys Conference (EuroSys'19)*. ACM Press,

New York, NY, USA, Article 8, 17 pages.

[14] Philip Fleming and John Wallace. 1986. How Not To Lie With Statistics: The Correct Way To Summarize Benchmark Results. *Commun. ACM* 29, 3 (March 1986), 218–221.

[15] Eric Humenay, David Tarjan, and Kevin Skadron. 2007. Impact of process variations on multicore performance symmetry. In *Proceedings of the conference on Design, automation and test in Europe*. IEEE Computer Society Press, New York, NY, USA, 1653–1658.

[16] Philipp Hurni, Benjamin Nyffenegger, Torsten Braun, and Anton Hergenroeder. 2011. On the Accuracy of Software-Based Energy Estimation Techniques. In *Proceedings of the 8th European Conference on Wireless Sensor Networks (EWSN'11)*. Springer-Verlag, Berlin, Heidelberg, 49–64.

[17] J. Keane and C. H. Kim. 2011. An odomoeter for CPUs. *IEEE Spectrum* 48, 5 (May 2011), 28–33.

[18] Elie Krevat, Joseph Tucek, and Gregory Ganger. 2011. Disks Are Like Snowflakes: No Two Are Alike. In *Proceedings of the 14th Conference on Hot Topics in Operating Systems (HotOS'13)*. USENIX Association, Berkeley, CA, USA, 1–5.

[19] John L. Manferdelli, Naga K. Govindaraju, and Chris Crall. 2008. Challenges and Opportunities in Many-Core Computing. *Proc. IEEE* 96, 5 (May 2008), 808–815.

[20] Aniruddha Marathe, Yijia Zhang, Grayson Blanks, Nirmal Kumbhare, Ghaleb Abdulla, and Barry Rountree. 2017. An Empirical Survey of Performance and Energy Efficiency Variation on Intel Processors. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing (E2SC'17)*. ACM Press, New York, NY, USA, Article 9, 8 pages.

[21] Diana Marculescu and Emil Talpes. 2005. Variability and energy awareness: a microarchitecture-level perspective. In *Design Automation Conference, 2005. Proceedings. 42nd*. IEEE Computer Society Press, New York, NY, USA, 11–16.

[22] Ingo Molnar. 2018. scheduler changes for v4.21. https://lkml.org/lkml/2018/12/24/296.

[23] John K. Ousterhout, Donald A. Scelza, and Pradeep S. Sindhu. 1980. Medusa: An Experiment in Distributed Operating System Structure. *Commun. ACM* 23, 2 (1980), 92–105.

[24] Quentin Perret. 2018. Energy Aware Scheduling. https://lkml.org/lkml/2018/7/24/420.

[25] Simon Peter, Adrian Schüpbach, Paul Barham, Andrew Baumann, Rebecca Isaacs, Tim Harris, and Timothy Roscoe. 2010. Design Principles for End-to-end Multicore Schedulers. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Parallelism (HotPar'10)*. USENIX Association, Berkeley, CA, USA, 10–10.

[26] John Sartori, Aashish Pant, Rakesh Kumar, and Puneet Gupta. 2010. Variation-aware speed binning of multi-core processors. In *Proceedings of the 11th International Symposium on Quality Electronic Design (ISQED'10)*. IEEE Computer Society Press, New York, NY, USA, 307–314.

[27] Guru Prasad Srinivasa, Rizwana Begum, Scott Haseley, Mark Hempstead, and Geoffrey Challen. 2017. Separated By Birth: Hidden Differences Between Seemingly-Identical Smartphone CPUs. In *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications (HotMobile'17)*. ACM Press, New York, NY, USA, 103–108.

[28] Herb Sutter. 2005. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. http://www.gotw.ca/publications/concurrency-ddj.htm. *Dr. Dobb's journal* 30, 3 (2005), 202–210.

[29] Dan Tsafrir, Yoav Etsion, Dror Feitelson, and Scott Kirkpatrick. 2005. System Noise, OS Clock Ticks, and Fine-grained Parallel Applications. In *Proceedings of the 19th Annual International Conference on Supercomputing (ICS'05)*. ACM Press, New York, NY, USA, 303–312.

[30] Jim Tschanz, Keith Bowman, and Vivek De. 2005. Variation-tolerant Circuits: Circuit Solutions and Techniques. In *Proceedings of the 42nd Annual Design Automation Conference (DAC'05)*. ACM Press, New York, NY, USA, 762–763.

[31] William A. Wulf and Sally A. McKee. 1995. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News* 23, 1 (March 1995), 20–24.