Mitigating AVX-Induced Performance Variability with Core Specialization

Mathias Gottschlag, Frank Bellosa

Operating Systems Group
Karlsruhe Institute of Technology
E-mail: os@itec.kit.edu

CPU performance is increasingly limited by the power dissipation of the chip. In this situation, one method to increase power efficiency – and thereby also performance – is to add accelerators for specific tasks. These accelerators can be switched off when inactive for prolonged periods of time. When they are used, however, they can consume significant additional power.

To prevent excessive peak currents and voltage instability, the rest of the chip therefore might have to temporarily reduce its frequency. One example for such a situation can be found in current Intel CPUs with support for AVX2 and AVX-512 vector instructions. Whenever a core executes one of these instructions, the core automatically reduces its frequency [1]. This behavior is documented in the form of different AVX2 and AVX-512 turbo frequencies for the affected CPUs.

As frequency changes are not instantaneous, they also affects scalar code following the vector instruction, which can lead to significant overall performance loss, even if only a small part of the code is vectorized: For example, in a web server scenario, enabling AVX-512 instructions for SSL with ChaCha20-Poly1305 encryption caused a 10% slowdown for the whole web server stack, even though the SSL library only made up for 2.5% of all executed instructions [3].

If, as in this example, the performance of code depends significantly on the instruction set usage of other unrelated code running on the same core, two problems arise: First, on a multi-tenant system, a user can significantly degrade the performance for other users of the same system simply by periodically executing AVX-512 instructions. The potential for such unexpected performance degradation can be especially significant for any software workload with real-time requirements.

Second, even when all code on the system is controlled by one single user, development and deployment of complex software becomes more difficult and error-prone, as seemingly insignificant changes such as a minor SSL library update can have significant performance impact as described above. As a result, software changes need to be closely monitored for their impact on overall performance and instruction set usage needs to be coordinated over the whole software stack.

The ability to limit the performance impact of problematic code would be highly beneficial to system reliability. We therefore propose a technique to migrate any code executing AVX2 and AVX-512 instructions to dedicated cores. We show that this type of core specialization can mitigate the impact of AVX-induced frequency reduction on performance as well as performance predictability. This result yields an additional interesting insight: Even though modern Intel server CPUs are commonly assumed to be symmetric CPUs, it can be advantageous to think of these systems as reconfigurable heterogeneous systems with slow cores for vectorized code and fast cores for scalar code.

1 Approach

When a core executes AVX2 or AVX-512 instructions, current Intel CPUs only reduce the frequency of that single core to compensate for the increased power consumption [1]. As a result, if a subset of the cores is limited to scalar code, the system therefore behaves like a heterogeneous system – some cores support vector code and run at lower frequency, whereas the other cores only execute scalar code, but at significantly higher frequency. We therefore propose a variant of *core specialization* to improve performance predictability on these systems by executing code containing AVX instructions on a set of dedicated cores.

Although we expect a fully automatic implementation of such a policy to be possible, we demonstrate the general viability of the concept by manually instrumenting the nginx web server to execute the OpenSSL functions for encryption and decryption on a separate core using the `sched_setaffinity` function.

2 Evaluation

We evaluated this prototype with a setup derived from tests at Cloudflare [3]: We placed the nginx web server on 10 cores of a system with a 14-core Core i9 processor, with the other 4 cores running the wrk2 benchmark client. The web server served a simple web page which was compressed at runtime with the brotli compression method. The server used HTTPS with either AVX2 or AVX-512 implementations of ChaCha20-Poly1305 for encryption and decryption.

Note that OpenSSL with support for AVX-512 is 14% faster than when using only AVX2 instructions, which makes the use of AVX-512 worthwhile in other situations. In our evaluation setup, however, the throughput of the whole web server stack dropped from 7184 requests per second when using AVX2 to 6703 requests per second when using AVX-512 (6.7% overhead). If, instead, all encryption and decryption code was executed on a separate core, the system achieved a throughput of 7366 (AVX2) and 7205 (AVX-512) requests per second. These numbers show a 2.5% (AVX2) and 7.4% (AVX-512) speedup over a system without core specialization. Performance with core specialization is generally higher than without, which shows that core specialization can completely mitigate the performance impact of AVX-induced frequency reduction.

To show that this improvement is caused by increased frequency of the non-AVX cores, we measure the CPU frequency during all these experiments. As expected, the average frequency of all cores is almost completely proportional to the throughput, ruling out other effects caused by frequent thread migration.

3 Related Work

Our proposed approach is a variant of *core specialization*. Core specialization techniques dedicate cores to a subset of the overall functionality of the system by only executing a subset of a program code on each core. For example, FlexSC [5] and SchedTask [2] are techniques which improve cache locality by scheduling threads on cores which already have the working set of the next segment of code in their local cache.

On heterogeneous systems consisting of processors with different microarchitectures, core specialization is instead used to place code on the type of core which provides the best power efficiency [4]. This type of core specialization is most similar to our approach. We show that cores of current Intel CPUs have different performance characteristics depending on whether AVX-512 instructions are executed. We therefore propose to treat these systems like heterogeneous systems and to dedicate some cores to AVX-512 code.

4 Conclusion

Accelerators, while providing an energy-efficient path to improve CPU performance, increase the peak current consumption of the chip and therefore can force the rest of the chip to reduce its frequency whenever they are active. Current Intel CPUs show this behavior and provide lower maximum frequencies for code executing AVX instructions. These frequency reductions can affect unrelated scalar code.

In order to improve performance predictability, we propose a variant of core specialization where threads are migrated to a separate set of cores whenever they execute AVX-512 instructions, to be migrated back once the AVX-512-accelerated portion of the program has finished. Our prototype demonstrates the viability of such approaches and is able to completely mitigate the performance impact of AVX-512 instructions in a web server scenario. Further research has to explore methods to migrate threads as well as techniques to efficiently determine the AVX-512 accelerated parts of the program.

References

[1] Wikichip: Frequency Behaviour – Intel. `https://en.wikichip.org/wiki/intel/frequency_behavior`.

[2] P. Kallurkar and S. R. Sarangi. Schedtask: a hardware-assisted task scheduler. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 612–624. ACM, 2017.

[3] V. Krasnov. On the dangers of intel's frequency scaling, 2017. `https://blog.cloudflare.com/on-the-dangers-of-intels-frequency-scaling/`.

[4] J. C. Saez, M. Prieto, A. Fedorova, and S. Blagodurov. A comprehensive scheduler for asymmetric multicore systems. In *Proceedings of the 5th European conference on Computer systems*, pages 139–152. ACM, 2010.

[5] L. Soares and M. Stumm. FlexSC: Flexible system call scheduling with exception-less system calls. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 33–46. USENIX Association, 2010.