EPJ Web of Conferences **295**, 04040 (2024) https://doi.org/10.1051/epjconf/202429504040
CHEP 2023

Microarchitecture: A useful tool to organize machines in heterogeneous shared computing environments

Gregory Thain[1*], and *Igor* Sfiligoi, [2]

[1] University of Wisconsin – Madison, Center for High Throughput Computing, Madison, WI, USA
[2] University of California at San Diego, La Jolla, CA, USA

Abstract. The x86_64 instruction set architecture is not a single, consistent, compatible interface to execute computer programs. Since the initial release in 1999, every new generation has added new instructions, some of which were later removed. Most of these new instructions are intended to improve the performance of those programs which explicitly take advantage of them. However, running such a program on older CPUs without appropriate support, results in Linux SIGILL exception signal, which is difficult for end users to diagnose. On the other hand, compiling scientific code for the least common denominator ISA can leave significant performance on the table. High Throughput systems, containing very large number of machines, cannot require a single CPU version across hundreds of thousands of machines operating in dozens of sites. The OSG Open Science Pool alone consists of more than 20 different, subtly incompatible X86_64 implementations. In 2020, Intel, AMD and RedHat proposed new terminology and partitioned these dozens of microarchitectures into a strict hierarchy of four groups. The HTCondor Software Suite and the OSG now have first class support for these microarchitectures. This paper discusses the advantages for users and future work around microarchitecture support.

1 Introduction and motivation

The Open Science Grid (OSG)[1] is a collaboration consisting of more than one hundred independent sites which choose to share computing resources with each other. In order to maximize the capacity of the OSG, we intentionally choose to minimize our selectivity on the types of computing resources that may join. This is unlike most High-Performance Computing (HPC) or supercomputing sites, which support very homogenous hardware, often with clusters of completely identical worker nodes, or at worst, only a small handful of distinct types or generations of worker nodes. We have identified over twenty different variants of the x86-64 architecture [2] in the OSG, from both Intel and AMD. Although there is a least common denominator subset instruction set architecture (ISA) common to all these implementations, each implements many different superset instructions sets. Executables

* Corresponding author: gthain@cs.wisc.edu

which take advantage of these instructions will fault with an illegal instruction error on those machines which don't support them, which is a bewildering result to naïve users.

There are several ways to select the set of machine instructions in a program. The most direct is to write some small performance-critical section in assembly or perhaps using direct low-level compiler intrinsics that map one-to-one with machine instructions. This is a method for experts and will not be otherwise covered in this paper. A just-in-time (JIT) compiler, like found in many Java runtimes is an ideal solution, as the compiler, which only runs on the target machine, knows the exact ISA that it will be running. This is infeasible, as very few OSG users or scientific computing users in general run code based on a Java VM language. The most common solution, and the one this work alludes to is to use a traditional ahead-of-time compiler, like gcc, and pass it a compile-time flag to request a very specific ISA (.e.g. *gcc -march=haswell main.c*).

1.1 The status quo ante and least common denominatorism

The easiest approach to the problem is to insist that all binaries run in the Open Science Pool (OSPool) [3] be compiled for the oldest, universally-portable subset of the x86_64 ISA. This has been the best practice since the founding of the OSG. However, there are three problems with the least common denominator compiling tactic.

1.1.1 Lack of access to source or difficulty in recompiling from source

Most users in the OSG are domain researchers, not computing experts. Most researchers run codes they have not personally compiled. Indeed, this is one of the benefits of the HTC approach – the exact same codes that run on a researchers' laptop can often run without any modification on the HTC cluster, but the cluster runs a lot more of those codes concurrently and independently. Forcing a user to learn how to obtain source code, configure, recompile with the appropriate flags and deploy into their workflow would be a significant tax on their research time, not to mention the need to test the correctness of the new build. Many codes used by our researchers are not intended to be recompiled by the end user, as their authors believe the best way to get their users running quickly is to provide them with precompiled binaries.

1.1.2 Reduced performance when running on more capable platforms

Should a research build or acquire binary executables that can run universally on an x86_64 machine, there may be a significant performance penalty in doing so. As the peak clock speeds of CPUs have plateaued, Intel and AMD increasingly rely on improving the performance of their products by raising the instructions per clock (ipc). While some ipc improvements can be made without changes to the ISA, many require the use of new, very complex instructions. All of the benchmark results published by CPU vendors are run taking advantage of any new instructions on CPUs. This is particularly true for every new generation of vector instructions, from MMX to SSE to AVX and beyond. So, any researcher's program compiled for a least common denominator ISA will run at some performance penalty. Depending on the workload, this penalty may be significant. It is not uncommon for a program that uses the vector instructions heavily to run two to three times faster than one that does not.

1.1.3 Increased energy consumption for suboptimal binaries

Directly related to performance is energy consumption. Even if the end user is not interested in the performance of their code, a program using the vector instructions and thus running significantly faster may consume proportionately less energy, an issue which is gaining more visibility around the world, as data centers are seen to be a growing fraction of total global energy consumption.

1.2 The problems with naming

Any given CPU has several names associated with it, and it is surprising difficult to select which name family to promote to users of the OSG. The most well-known naming scheme is the so-called marketing names, e.g., i3, i5, i7, i9 for most desktop/laptop Intel CPUs, Xeon Gold, Xeon Scalable Processors for server Intel CPUs, and Ryzen and EPYC for AMD CPUs. These names are not useful for our project, as they differ from vendor to vendor, and they don't unique describe the capabilities of a CPU. One generation of i5, for instance, will not necessarily support the same instructions as the previous generation of i5. The microarchitecture for each processor also has a code name, e.g., Haswell and Sandy Bridge for Intel CPUs and Bulldozer for AMD CPUs. These are not as well-known as the marketing name but have the opposite problem – there are many different microarchitecture code names which can implement the same ISA, meaning that the code name is too specific in this use case. Consider Figure 1, which shows the distribution of CPUs in the OSPool by code name.

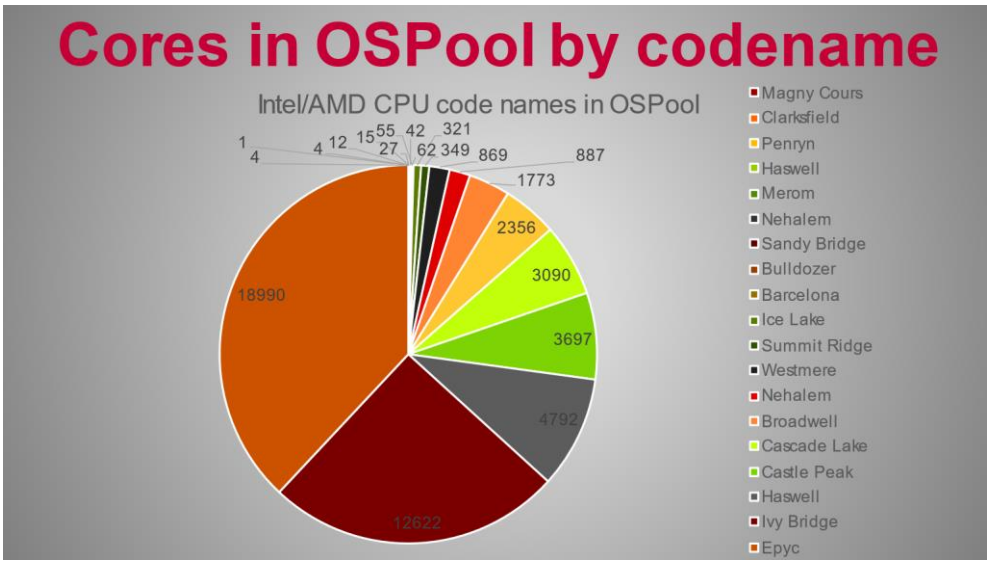

Figure 1: OSPool composition by code name.

2 Industry consensus: Microarchitecture

This naming and clustering of machine types is not a problem unique to HTC. Vendors of Linux distributions, such as Red Hat, have very similar problems. They would like their packages to run as efficiently as possible on the available hardware, but it is infeasible for them to maintain, test, and distribute dozens of different builds for various CPU subtypes. In July of 2020, Red Hat proposed [4] a clustering of all extant x86_64 cpu subtypes into three "microarchitectures", named x86-64-v2, x86-64-v3 and x86-64-v4, where progressively

higher "v" numbers denote more capability, and each set can execute any program targeted a lower set. The level of "x86-64-v1" was not defined but is implied as "not even x86-64-v2". This proposal was quickly adopted by the cpu vendors [5], the compiler implementers (who added -march=*microarch* flags to their compilers, and the Linux distributions.

Name	Required Instructions
x86-64-v2	cx16, lahf_lm, popcnt, sse4_1 sse4_2, ssse3
x86-64-v3	abm avx, avx2, bmi1, bmi2, f16c, fma, movbe, xsave
x86-64-v4	avx512bw, avx512cd, avx152dq, avx512f, avx512vl

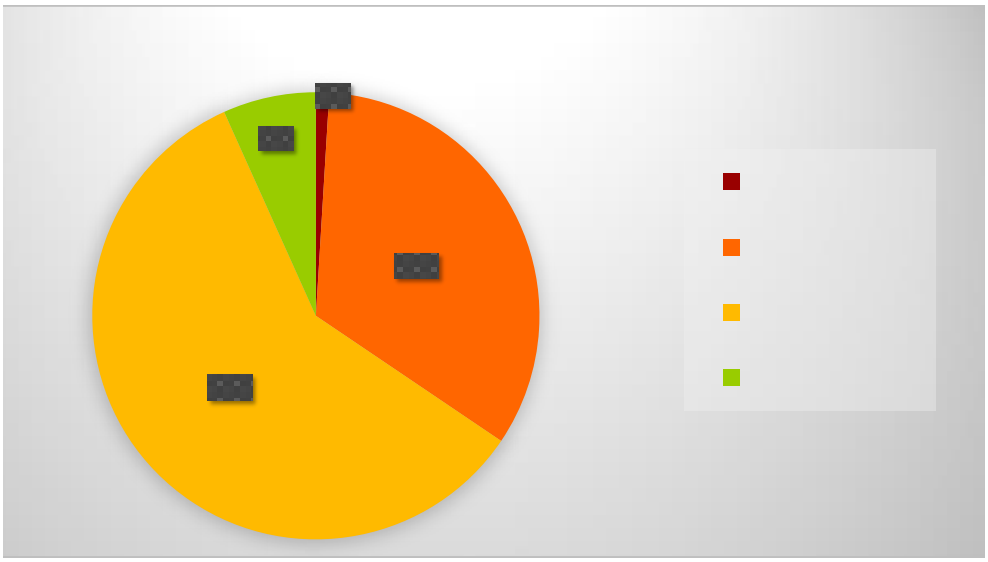

Figure 2: OSPool composition by microarchitecture.

The HTCondor Software Suite[6], which powers the OSG, added support in version 9.12.0, released in October of 2022 for this microarchitecture detection. At boot time, the HTCondor *condor_startd*, which is responsible for the worker node, detects the microarchitecture of the system it is running on, and advertises that as part of the slot *classad*. This attribute, named *Microarch*, can then be used by administrators for measuring what part of their pool has which microarchitecture, while the end users can use this attribute for matching jobs to machines. For example, they can request that their job only land on machines that support at least microarchitecture *x86-64-v3*, or perhaps exactly one microarchitecture. With this clustering, the composition of the OS Pool is much clearer, as shown in Figure 2.

With this new nomenclature, it is much easier to talk about the problem of microarchitecture, to describe our pools, and to optimize execution speeds and energy usage of our cluster. A couple of examples are given below.

2.1.1 A composite HTCondor resource selection example

To better illustrate the benefits of reseource selection based on microarchitecture, consider an application that can make good use of both f16c and AVX2 instructions. Neither are part of the original x86_64 ISA, so users running such an application will have to restrict to only CPUs supporting those two extensions.

At the time of writing, there were about 90 different CPU models that supported those extensions and about 40 CPU models that did not, so some kind of resource selection is required. Note that about 10 models supported only one of the two extensions, but not both.

HTCondor has long had support for detection and reporting of some of the x86_64 ISA extensions, including cx16, sse3/4, f16c, avx, avx2 and various avx512 flavors. The two extensions of interest are reported as *has_fp16c* and *has_avx2* attributes in the slot *classad*. The user could thus have selected the appropriate slots using these two attributes, i.e. '(has_fp16c && has_avx2)'. Using the new microarchitecture attribute, the selection becomes '((Microarch=="x86_64-v3") || (Microarch=="x86_64-v4"))'.

2.1.2 A previously unsupported HTCondor resource selection example

HTCondor does not support all of the x86_64 extensions, there are just too many of them. Example missing ISA extensions are popcnt and bmi1/2. If a user had an application that relied on either of those, there was previously no obvious way to select slots that supported them.

That said, most ISA extensions were introduced together by the CPU vendors, so selecting on one of the HTCondor-supported API extension flags would most likely produce the desired results, e.g. by using the *has_avx2* attribute. This would however require the user to do significant CPU model research, and is also not guaranteed to always work.

Using the *microarch* attribute, on the other hand, provides both easier documentation and guaranteed presence of the desired ISA extension.

2.1.3 A software provider example

As previously noted, science users rarely compile from source the applications they use. Instead, they tend to use pre-compiled binaries that have been produced by the software providers.

Software providers typically have a vested interest in providing binaries that are as fast as possible, but they have to balance that against both the need for compatibility with older hardware platforms and the need for keeping the number of binary releases reasonably small. The microarchitecture levels provide a great way to achieve that.

Having software binaries that are advertised as supporting a specific microarchitecture brings the need for users to match the binary with the CPU in use. Either by filtering out unsupported CPUs or by picking at runtime the right binary to execute. Having the microarchitecture detected by the workload management system, i.e., HTCondor, this task becomes very simple.

For example, the user (or the support personnel) can deploy the four binary versions of the desired application in four separate directories, and then pick the right binary at runtime, using the *Microarch* attribute provided by HTCondor [7].

3 Conclusions and future work

We have found that clustering our machines into the consensus industry terminology of "microarchitecture" makes it clearer for users and administrators what class of machine exists in the Open Science Pool, and the HTCondor ecosystem at-large. This also allows software providers to better bin their binary executable releases, improving the performance of their software on the most popular compute systems without the need for too many sub-versions.

An outstanding problem is that there is no easy method currently to detect which microarchitecture a given binary was compiled for. Even the brute force method of

disassembly and searching for certain instructions fails in the face of dynamic loading and run-time switching on detected CPU type. We would like to see binaries that have been explicitly compiled for a given microarchitecture noted as such in their ELF headers.

Also, we would like to see the use of the microarchitecture terminology normalized across the HTC and HPC landscape, in order to help all users understand the technologies and generations at work.

This work was supported by the National Science Foundation under Gant number 2030508.

References

1. R. Pordes et al. *The open science grid*, J. Phys. Conf. Ser., 78, 012057. (2007) https://doi.org/10.1088/1742-6596/78/1/012057
2. Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 2A: Instruction Set Reference* (2016) https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-2a-manual.pdf
3. OSG. *OSPool*. (2006) OSG. https://doi.org/10.21231/906P-4D78
4. Florian Weimer, Accessed September 2023. https://lists.llvm.org/pipermail/llvm-dev/2020-July/143289.html
5. H.J. Lu et al. System V *Application Binary Interface AMD64 Architecture Processor Supplement*, September 5, 2023. https://gitlab.com/x86-psABIs/x86-64-ABI
6. M. J. Litzkow, M. Livny and M. W. Mutka, *Condor-a hunter of idle workstations*, Proceedings. The 8th International Conference on Distributed, San Jose, CA, USA, 1988, pp. 104-111. (1988) doi: 10.1109/DCS.1988.
7. HTCondor Users' Manual. *Services for Running Job*, Accessed September 2023. https://htcondor.readthedocs.io/en/latest/users-manual/services-for-jobs.html