The Isabelle Community Benchmark

Fabian Huch, Vincent Bode

Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany

Abstract

Choosing hardware for theorem proving is no simple task: automated provers are highly complex and optimized programs, often utilizing a parallel computation model, and there is little prior research on the hardware impact on prover performance. To alleviate the problem for Isabelle, we initiated a community benchmark where the build time of HOL-Analysis is measured. On 54 distinct CPUs, a total of 669 runs with different Isabelle configurations were reported by Isabelle users. Results range from 107 s to over 11 h. We found that current consumer CPUs performed best, with an optimal number of 8 to 16 threads, largely independent of heap memory. As for hardware parameters, CPU base clock affected multi-threaded execution most with a linear correlation of 0.37, whereas boost frequency was the most influential parameter for single-threaded runs (correlation coefficient 0.55); cache size played no significant role. When comparing our benchmark scores with popular high-performance computing benchmarks, we found a strong linear relationship with Dolfyn ($R^2 = 0.79$) in the single-threaded scenario. Using data from the 3DMark CPU Profile consumer benchmark, we created a linear model for optimal (multi-threaded) Isabelle performance. When validating, the model has an average R^2-score of 0.87; the mean absolute error in the final model corresponds to a wall-clock time of 46.6 s. With a dataset of true median values for the 3DMark, the error improves to 37.1 s.

Keywords

Isabelle, theorem proving, user benchmark, run-time performance, performance prediction

1. Introduction

Choosing appropriate hardware and tuning configuration parameters is a common task when one wants to run software optimally. For a complex and truly parallel interactive proof assistant such as Isabelle, many factors influence run-time performance: The prover needs a Java and a Meta Language (ML) run-time, the number of threads is variable, as is the amount of heap memory – which in turn (in combination with the CPU architecture family) dictates which ML platform and hence Poly/ML backend may be used. On a hardware level, CPU specs, the memory hierarchy, and interconnects all influence how well the software components perform and how the system as a whole behaves. The parallel efficiency of Isabelle (i.e., the ratio of actual time versus sequential time divided by the number of parallel units) decays according to a non-linear characteristic [1], as is the case in most parallel systems. As a result, there is no single hardware or software characteristic that dominates the observed performance behavior.

In Isabelle, performance is important both in the interactive mode (such that processing

PAAR'22: 8th Workshop on Practical Aspects of Automated Reasoning, August 11–12, 2022, Haifa, Israel

huch@in.tum.de (F. Huch); vincent.bode@tum.de (V. Bode)
https://home.in.tum.de/~huch (F. Huch); https://home.in.tum.de/~bodev (V. Bode)
0000-0002-9418-1580 (F. Huch); 0000-0003-2353-389X (V. Bode)

changes and running solvers is faster) and in a batch build mode, where *sessions* (i.e., collections of formalizations) can be processed. Independent sessions can even be run in parallel with multiple ML processes.

However, making informed decisions on hardware is no trivial task. Members of the Isabelle community have to rely on word of mouth to determine which processors and memory to use, and configuration parameters (such as the number of threads or heap sizes) are largely folk knowledge – backed by experience collected over the years, ad-hoc experiments, and sometimes intuition. While there is some performance data available, it is not helpful in that regard as it only covers a very small number of machines.

With new and exciting hardware being developed at a fast pace, one can often be overwhelmed by the sheer variety of hardware options available. Hence, the question of which hardware to recommend for Isabelle can often not be answered exhaustively or satisfactory. This is relevant both for individuals working with Isabelle, and for the larger-scale server infrastructure maintained for continuous integration and for Isabelle and the Archive of Formal Proofs.

To alleviate this problem, a solid data base with performance benchmark results for a wide variety of involved hardware and configurations is needed. Not only would that directly answer the question of optimal configurations for a given system and allow one to compare the hardware on the market, but such a collection of data (if large enough, and kept up to date) would also allow one to predict performance of other hardware for which no Isabelle data is available yet.

In this paper, we outline our Isabelle community benchmark, discuss the immediate results and findings, and derive a model to predict the Isabelle performance of unknown CPUs with the help of widely used benchmarks for which more data is retrievable. Our source-code and data and is made available publicly[1]. Section 2 covers related work; we explain our benchmark set-up in Section 3, and discuss the results in Section 4. In Section 5, we conclude and discuss future work.

2. Related Work

Parallel run-time performance has been first analyzed for Isabelle when parallelism was introduced by Wenzel in [1]. Benchmarks for multiple different sessions on a single test machine already showed that the speedup (in terms of run-time) peaked at three worker threads with a factor of 3.0, and slightly decreased for four cores. Matthews and Wenzel described the necessary adaptations to the Poly/ML run-time that were necessary for introducing parallelism, and analyzed the resulting bottlenecks [2]. They found that the parallelization model for Isabelle sometimes failed to fully utilize all worker threads. Moreover, the synchronization model that uses a single signal across all threads for guarded access was identified (but not analyzed) as a potential bottleneck. Finally, it was observed that the single-threaded garbage collection is responsible for up to 30 % CPU-time for 16 threads. Overall, a maximum speedup of 5.0 to 6.2 could be achieved using 8 threads.

In automatic theorem provers, run-time is an important factor, since it can dictate whether a goal can be proven within the given cut-off time. As a result, much research includes analysis of the run-time performance of provers or individual prover components. Typically, only a

[1] `2022-paper` folder in https://isabelle.systems/benchmark

single hardware configuration is used, which is reasonable for the analysis for single-threaded systems [3]. However, since performing such analysis on a wide range of different hardware is often impractical, run-time performance analysis of parallel approaches is frequently carried out on single systems or clusters [4, 5, 6]. These results don't always generalize, because the hardware used can have a significant impact on the observed results.

In contrast, results for the Isabelle `sledgehammer` proof-finder tool show that when running *multiple* automatic provers to solve a goal, run-time becomes less important: In their *judgement day* study [7], Böhme and Nipkow found that running three different Automated Theorem Provers for five seconds each solved as many goals as running the most effective one for 120 s. Subsequently, in direct follow-up work [8], run-time was not analyzed.

For automatic provers, a large range of benchmarks exist to judge their effectiveness on a given set of problems. One of these is the widely known TPTP library [9]. However, there is not much work investigating the effect of hardware in the field of automated reasoning. To the best of our knowledge, there exists no other benchmark comparing the hardware impact on run-time performance of any theorem prover, and this is the first work that analyzes this effect on a wide range of different hardware.

3. Benchmarking Methodology

The benchmark has to fulfill to multiple requirements: It needs to capture typical computations found in Isabelle — mostly symbolic computation —, have a reasonable run-time for the end user, and motivate users to want to see how their machines perform (i.e., results should be self-evident). We settled for a clean build of the HOL-Analysis session: It is a typical Isabelle formalization which runs in approximately five minutes on typical consumer machines. Many Isabelle users have likely run a similar workload for their own work in the past.

While users can easily contribute results for their favourite Isabelle configuration, we supplied a small script to run a comparable set of configurations automatically [2]. This way, the whole benchmark can be run with a single command (assuming a working installation of Isabelle 2021-1) on any platform. We vary the underlying ML platform between 64_32 (64-bit mode with 32-bit values) and true 64-bit mode, heap sizes of both the ML and JVM process (set to the same value to reduce the number of linear combinations, as early benchmark results indicated they play only a minor role here), and the number of worker threads for parallel proof checking.

Results are collected in a collaborative spreadsheet[3] with automatically updated figures for fastest CPUs and parallel efficiency. The benchmark is not intended as one-shot experiment, but rather as a continuous community effort to maintain an overview over the Isabelle computing landscape as new hardware emerges. It is being kept open for new results, and will be maintained for future Isabelle versions.

[2]Documentation and code at https://isabelle.systems/benchmark

[3]https://docs.google.com/spreadsheets/d/12GhEwSNSopowDBq5gSem3u39fliiIcoTIZHMnX4RE3A

3.1. Benchmark Score (Isascore)

For the benchmark results, we use the wall-clock build time as an intuitive metric. Together with the well-known HOL-Analysis build target, the metric immediately gives a good understanding of Isabelle performance. However, it is not well suited to compare to other metrics such as throughput, because the relationship between time to solution and throughput is inverse. To still allow using simple linear models such as the Pearson correlation, we introduce a benchmark score that we call *Isascore*. It reflects the number of HOL-Analysis runs one could complete in a day, i.e.:

$$\text{Isascore} = \frac{1\,\text{d}}{\text{wall-clock time}} \tag{1}$$

3.2. Threats to Validity

The experiments discussed in this paper could not be performed in a controlled environment, since they were run by members of the Isabelle community rather than exclusively by the authors of this paper. This means that various outside factors may have influence on the reported results, though it seems reasonable to assume that those factors should usually be constant between different configurations of the same benchmark run. The effect of machine-local anomalies can be mitigated for hardware where we received several independent measurements by using statistical techniques. Furthermore, due to reasons of practicality in orchestrating data collection, extended system specifics beyond the CPU model, OS, and memory configuration were not recorded. There is a possibility that relevant parameters may have been missed. Therefore, like all performance benchmarks, these results represent upper bounds of what might be achieved with a given system configuration. Lastly, while the benchmark was posted on the Isabelle-users mailing list, in principle the data entry was open to public and could have been misused.

4. Results

At the time of writing this paper, 669 results for a total of 594 unique configurations have been reported, utilizing 54 distinct CPUs. Those include Intel Desktop/Server CPUs from Sandy Bridge to Alder Lake, AMD Ryzen Zen2 to Zen4 processors as well as Epyc and Threadripper server systems, a Fujitsu A64FX, and Apple M1 processors.

Table 1
The five processors with the lowest time to solution, using each processor's optimal configuration (median value taken for multiple runs).

Processor Model	Total Cores	Base Clock	Wall-Clock Time
12th Gen Intel(R) Core(TM) i7-12700K	12	3.6 GHz	108 s
Apple M1 Pro (native arm64)	10	2.1 GHz	137 s
AMD Ryzen 9 5900X	12	3.7 GHz	143 s
Intel(R) Core(TM) i9-9900K	8	3.6 GHz	163 s
Apple M1 Pro	10	2.1 GHz	164 s

Table 1 shows the five CPUs with the lowest time to solution, using the median value as an aggregate for multiple runs of the same configuration. Older Intel and AMD consumer hardware is surpassed by the Apple M1 Pro chip; only the most recent Intel core line performs better. Due to the nature of the benchmark, server and high performance hardware does not rank highly, with the best performing system (2x AMD Epyc 7742) clocking in at 184 s.

In the following, we analyze how Isabelle configuration influences performance, investigate the impact of hardware parameters, and then compare our results to other computational benchmarks. Where individual CPUs were concerned, we filtered out special system configurations (e.g., overclocked hardware, dual-cpu systems, power-saving mode). We also encountered a small number of extreme outliers where Isabelle run-time was much longer than expected. For two of those, we could identify the user and investigate; in both cases, the system configuration was at fault (excessive swapping, UEFI set to "silent mode") and when corrected, results were much closer to the rest of the data. We could not investigate the third extreme outlier but excluded it from the following, since it is likely to stem from a similar cause.

4.1. Multi-Threaded Performance

The number of threads used plays a major role in the overall performance. Figure 1 illustrates how the wall clock time and CPU time compare from a single thread to up to 128 threads. The optimal wall-clock time is achieved with 8 to 16 threads depending on the hardware and greatly increases if more threads are used. This is typical behavior for a strong-scaling benchmark like ours, where the relative impact of communication increases with an increase in the number of threads used. For more than the optimal number of threads, the run-time increases substantially. The underlying limitations of the parallel computing model – the single-threaded garbage collection of the Poly/ML run-time and worker starvation after parallelization is saturated – were already discussed in [2], albeit tests were run on a machine with 32 cores. It might be a surprise that the scalability is so low when distributing across more threads. In contrast, the CPU time divided by number of threads (which is not an ideal metric, but the only feasible solution due to the nature of the benchmark) flattens out at eight threads. In small-scale experiments, we found that the JVM process takes up a constant amount of CPU time independent of the number of threads (about 26 % in single-core mode). This means that there is not too much computation overhead but the hardware can not be properly utilized by the ML process, most likely due to the single-threaded garbage collection that stops all threads when running. This is an inherently sequential task, which means that Amdahl's Law (the speedup is limited by $1/(1 - \text{parallelizable portion})$ [10]) limits the achievable speedup for this problem.

The parallel efficiency paints a similar picture in Figure 2, decreasing almost linearly (on the logarithmic x-axis) up to 32 threads at which it is at a median of 0.065. With the number of threads tending to the limit of 128, it approaches 0.002. There is an outlier where the parallel efficiency is over one – super-linear speedup is unusual but can appear in practice because of caching effects or resource contention in the measured system.

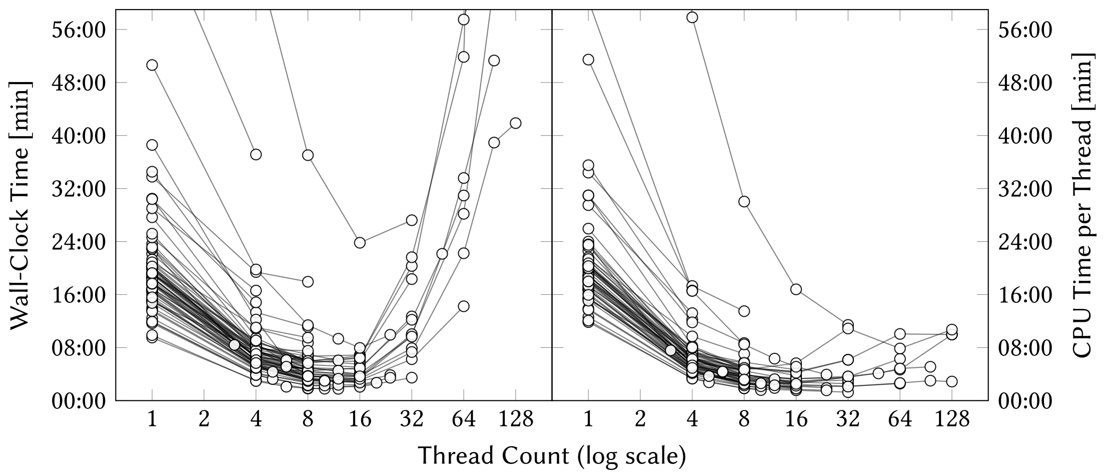

Figure 1: Run-time by number of threads (log scale): Wall-clock time on the left, CPU time per thread on the right.

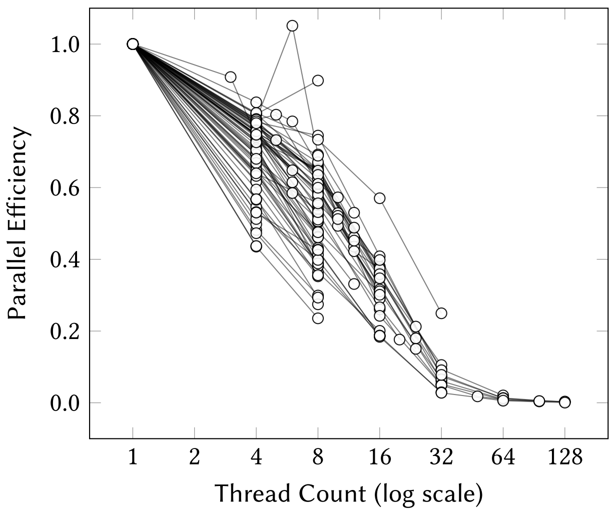

Figure 2: Parallel efficiency by number of threads. The HOL-Analysis build does not scale past a low number of threads, with parallel efficiency already down by half when using eight threads.

4.2. Performance Impact of Heap Memory

As preliminary results indicated that heap memory (as long as sufficient) only plays a minor role in performance, we keep the JVM and Poly/ML processes at the same heap size. We know from experience that a few gigabytes of memory suffice for HOL-Analysis; however, increased parallelism requires more memory in principle due to memory overhead. Hence, the range of examined heap sizes depends on the number of threads used. Figure 3 shows the change in run-time for different heap settings relative to the minimal setting. The boxes capture the 25 and 75 percentiles as height and sampling size as width; whiskers correspond to the extreme values. The results show that performance is not affected very much by heap size. Following the line of medians, wall-clock time slightly increases above 16 GB (where the 64 bit Poly/ML backend needs to be used, as the more efficient 64_32 mode does not allow more than 16 GB), as well as for very large values. We observed a single outlier for 64 threads and 128 GB heap memory at a relative factor of 0.66.

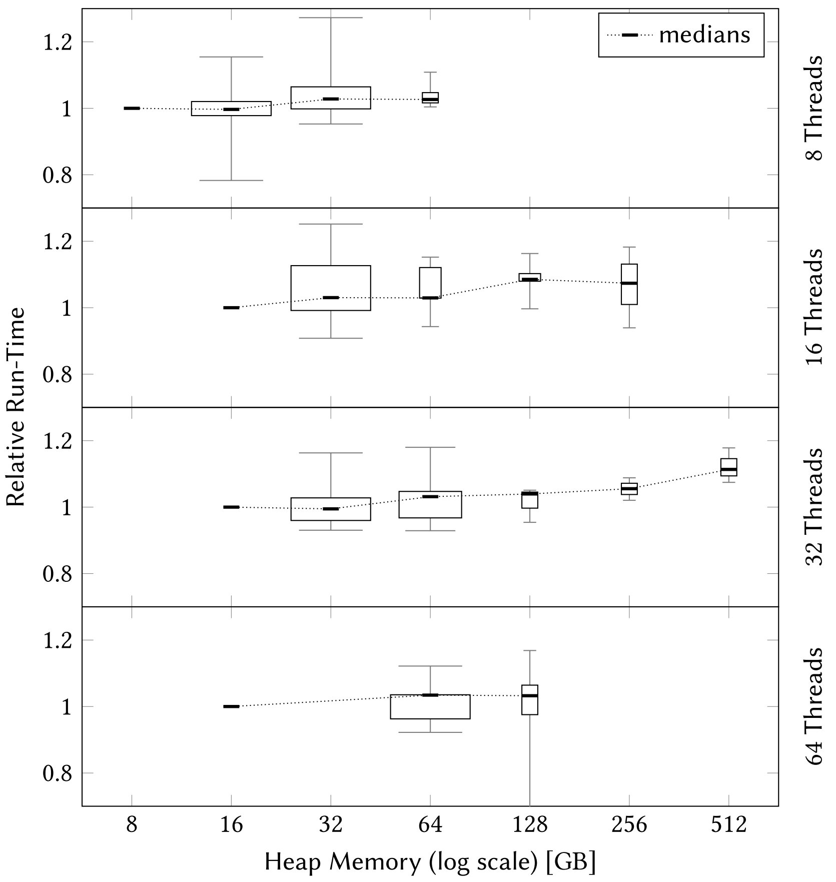

Figure 3: Relative run-time changes to minimal memory configuration, by heap memory. The available heap memory has only a marginal impact on the run-time, with large amounts of available heap memory degrading performance independently of the number of threads used.

4.3. Influence of Hardware Characteristics

Based on folk knowledge about Isabelle performance, we suspected that cache size would be a major factor; it was debated whether boost clock speed would be relevant. To test the hypotheses, we analyzed the impact of size of the L3-cache, base clock speed, and maximal (boost) clock speed (ignoring power-save cores where applicable) on Isabelle performance. Table 2 shows the correlation between Isascore and those parameters (APA notation as explained in caption). At our significance level of 0.05, we did not find cache size to impact performance significantly. Base frequency is weakly correlated with the Isascore for a single thread (though at the edge of significance) and a bit more strongly (and much more significantly) in the multi-threaded scenarios. Finally, boost frequency has a significant medium correlation for all modes, which is strongest in the single-threaded configuration with a value of 0.55.

Table 2
Pearson correlation of hardware parameters with Isascore in APA notation: degrees of freedom (number of observations minus two), r (strength) and p (significance). Cache size is left out as there is no significant correlation.

Mode	Base Frequency	Maximum Boost Frequency
1 thread	$r(51) = .27$, $p = .050$	$r(51) = .55$, $p < .001$
8 threads	$r(47) = .39$, $p = .006$	$r(47) = .43$, $p = .002$
optimal config	$r(52) = .37$, $p = .006$	$r(52) = .50$, $p < .001$

A possible explanation is that boost frequency can only be sustained for a single core in most CPUs, hence single-threaded performance profits from it a lot; in the multi-threaded scenario, the actual core frequency is much closer to the base frequency and thus its impact is larger.

4.4. Comparison to Computational Benchmarks

Performance benchmarks exist for many applications; additionally, synthetic benchmarks are often used to evaluate hardware performance. They can roughly be categorized into scientific computing versus consumer benchmarks. In the following, we compare the results of our Isabelle community benchmark with a number of publicly available datasets for such benchmarks. For the comparison, we selected results with matching processors, and matched the benchmarks' multi-thread setting (e.g., specific thread count, or all cores). To obtain sufficiently large datasets, we selected some of the most popular benchmarks.

4.4.1. Benchmarks in High-Performance Computing

The first analysis we wish to conduct is a comparison of Isabelle performance with some scientific programs. For this analysis, we chose to import data from the High Performance Computing suite on OpenBenchmarking. We selected the three benchmarks that had the most public results available (in their primary configuration) at the time of writing: *Himeno*, *NAMD*, and *Dolfyn*. Himeno[4] is an incompressible fluid analysis code written in Fortran and C [11]. While a distributed memory parallel version exists (using MPI with Fortran VPP), we concern ourselves with the sequential implementation. NAMD[5] is a shared memory parallel molecular dynamics code based on C++ and Charm++ [12]. The data we use stems from machine-wide parallel trials. Finally, Dolfyn[6] is a sequential computational fluid dynamics code based on Fortran [13].

Figure 4 shows the results when correlating each of the high-performance computing benchmarks with Isabelle performance. Himeno reports performance in terms of work done over time (where higher is better), while NAMD and Dolfyn measure time (per simulated ns, and

[4]Results from https://openbenchmarking.org/test/pts/himeno
[5]Results from https://openbenchmarking.org/test/pts/namd
[6]Results from https://openbenchmarking.org/test/pts/dolfyn

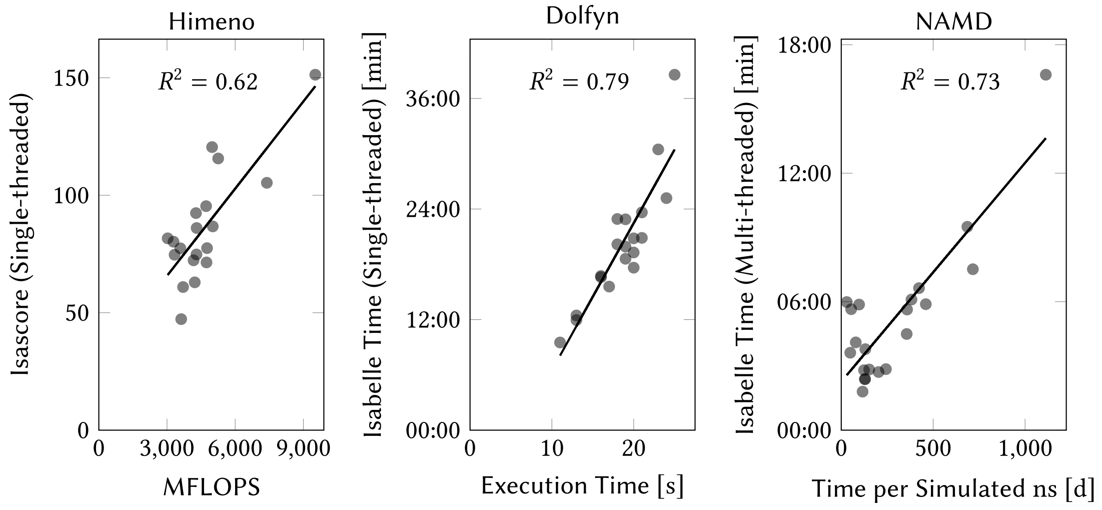

Figure 4: Comparison of single-threaded (Himeno and Dolfyn) and multi-threaded (NAMD) Isabelle results with different high-performance computing benchmarks, with linear regression line.

to solution; lower is better). For Himeno, we therefore compare against Isascore, while with NAMD and Dolfyn we compare against our observed wall clock time.

NAMD, as the only benchmark of these three that scales well with parallel resources, has no significant correlation with single-threaded Isabelle time. However, it has a strong linear relation with multi-threaded time. The two less scalable benchmarks correlate much closer with Isabelle single-thread performance, where Dolfyn has a particularly nice correlation that holds well for the most performant processors. In both cases, correlation with multi-threaded Isabelle results is much worse (R^2-values: Himeno 0.46, Dolfyn 0.52). For both the Isabelle benchmark and Dolfyn, the top processor that was tested is the same (Intel i7-12700K), and on both benchmarks it has a margin on the runner-ups. This is also visible on the Himeno benchmark, where the 12700K produces the highest floating point throughput of all tested processors. However, it is not a highly parallel processor, which is why its NAMD results are less favorable. This again shows that Isabelle performance is significantly impacted by the single-thread performance of the underlying processor.

4.4.2. Consumer CPU Benchmarks

For our second comparison, we chose some of the most common consumer benchmarks to compare to: *PassMark CPU Mark*[7], *Geekbench 5*[8], *Cinebench R15*[9], and *3DMark CPU Profile*[10]. For sequential performance, Figure 5 shows the scatter plots of Isascore to consumer benchmark scores, which are normalized to a [0; 1] range so the plots can be compared against. A strong positive relationship can be observed for all benchmarks, with R^2-values in the range 0.63 to

[7]Results from https://www.cpubenchmark.net/CPU_mega_page.html
[8]Results from https://browser.geekbench.com/processor-benchmarks.json
[9]Results from https://us.rebusfarm.net/en/tempbench
[10]Results from https://www.3dmark.com/search, median over the top-100 values

0.82. A few moderate outliers are present (possibly due to system configuration). All in all, the Isabelle benchmark seems quite similar to those consumer benchmarks for a single thread.

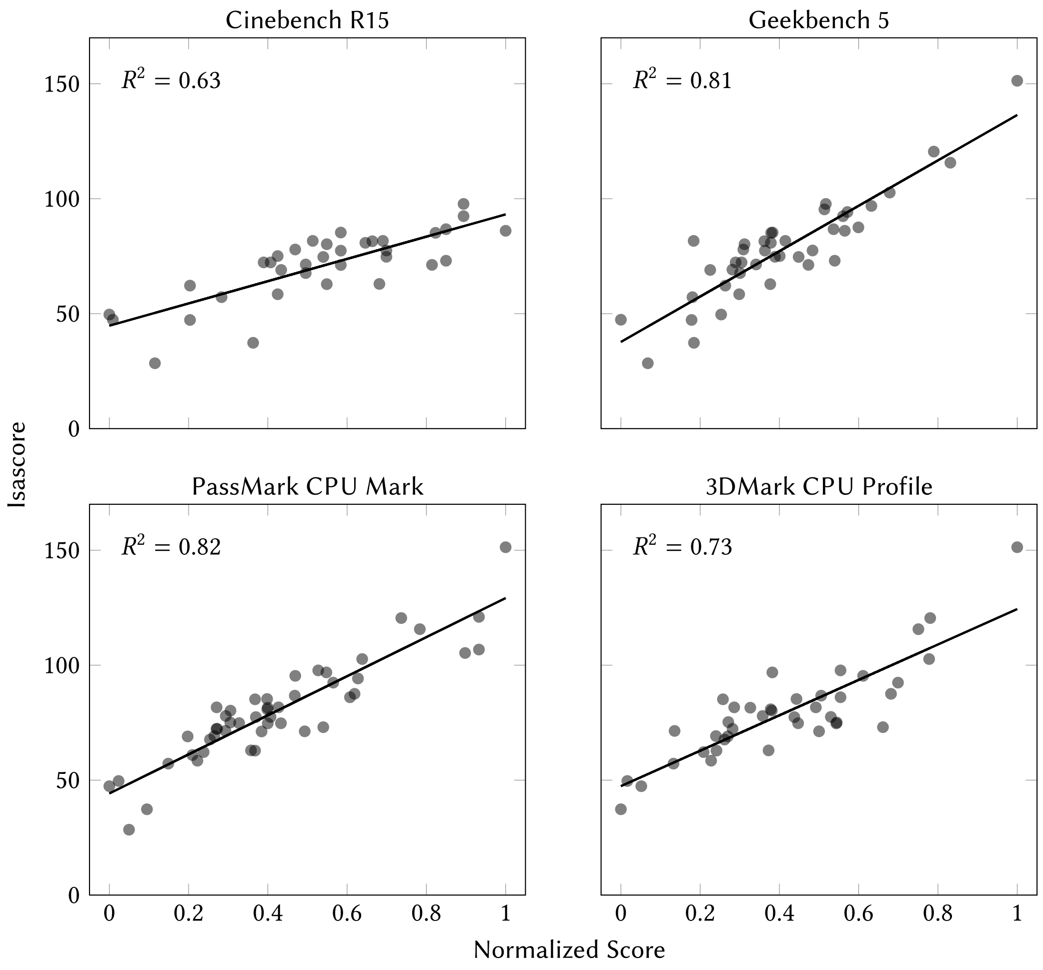

Figure 5: Comparison of single-threaded results with different consumer benchmarks, with linear regression line. Benchmark scores are normalized to $[0, 1]$ range.

This gives rise to prediction of Isabelle performance, which would allow one to judge hardware on which Isabelle was not executed before. However, single-threaded results are not meaningful for real-world performance, and scaling them according to the average parallel efficiency did not yield helpful results (R^2-values: Cinebench 0.59, Geekbench 0.70, PassMark 0.68, 3DMark 0.58). Not many datasets for consumer benchmarks report on results for different number of threads, most report only a single “multi-core” value where all threads are utilized. An exception to that is the 3DMark CPU Profile benchmark, where results are reported for 1 to 16 threads individually (in steps by power of two). This allows us to create a better correlation, because all consumer benchmarks tested had a far better parallel efficiency in the limit and were hence not suited for direct prediction.

When using 8 and 16 threads in both the 3DMark and Isabelle benchmark, score and Isascore are strongly to moderately correlated and have individual R^2-values of 0.77 and 0.61, respectively. This makes the 3DMark well suited for performance prediction. Since the optimal number of threads is in between, we use the average of its 8-thread and 16-thread results to create a linear model for performance prediction (tuning for a non-uniform split did not yield better results). Using ten times ten-fold cross-validation (i.e., averaging results over multiple iterations, splitting the data into ten parts, and using each part as a test set and the remainder as training set), the linear regression has an average R^2-value of 0.87. Figure 6 shows the final model ($R^2 = 0.84$) and the resulting predictor for wall-clock time, which has a mean absolute error (MAE) of 46.6 s. However, that error is somewhat exaggerated by the data collection method: The public 3DMark data only shows only the top-100 results, from which we use the median values. The regression improves slightly when the (non-public) true medians are used, and the MAE decreases to 37.1 s.

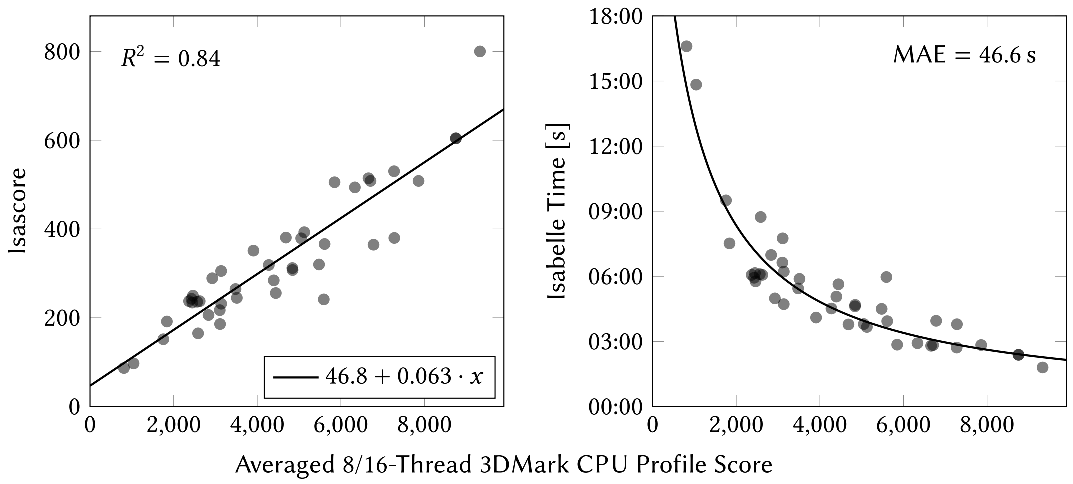

Figure 6: Scatter plots for final model of Isabelle performance. Linear model on Isascore on the left, corresponding wall clock run-times on the right.

The residual plot displayed in Figure 7 has no noticeable patterns and the residual distribution roughly follows a normal distribution. All in all, the model simplicity and good fit indicate that this linear model is quite well suited for performance prediction, as long as the other system parameters are kept within reasonable bounds and the configuration is well tuned.

5. Conclusion

This work resolves our questions on Isabelle performance for the 2021-1 version. The Isabelle community benchmark that we initiated saw lively participation and hundreds of results were reported for a total of 54 distinct CPUs. The results form a solid data base for tuning of Isabelle configuration; when not constrained, the optimal configuration is at 8 to 16 threads with 16 GB heap memory for both the Java and ML process (at least for HOL-Analysis, larger sessions might require more).

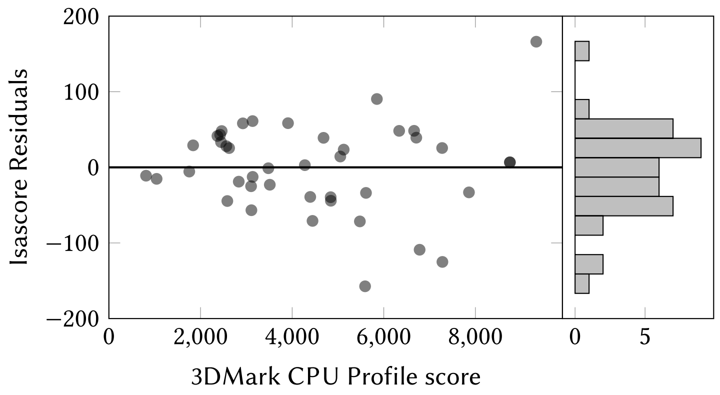

Figure 7: Residual plot for final linear model, with histogram. The residuals show no apparent patterns and roughly follow a normal distribution.

When buying new hardware, the benchmark results give a good indication of which processor is desirable for Isabelle. Individual CPU parameters are not as important as clock speeds are only correlated with medium strength (though boost clock more than base clock) and cache size not significantly at all. Instead, for hardware that has not yet been tested with Isabelle, other benchmarks can greatly help in judging performance: While the single-threaded Isabelle benchmark score is strongly correlated with many benchmarks (most strongly with the Dolfyn high-performance benchmark and the PassMark CPU Mark), the multi-threaded scenario was more difficult to model; In the end, we found a good predictor by using 3DMark CPU Profile scores from 8 and 16 threads, with a final mean absolute error of 46.56 s. The model has a good fit, and one can assume that it is fairly future-proof given that hardware from the last ten years is properly predicted.

6. Future Work

If the Isabelle computation model does not change, there is not much left to be done on the topic of performance prediction: The benchmark from which Isabelle performance is predicted is widely popular and data for new hardware is usually quickly added, and the fit is about as good as one can hope for. Still, the model should be validated after a few years, and we are curious to see whether future hardware characteristics will be able to break the trend.

One other aspect that we did not touch on is running multiple independent sessions in parallel, which is often possible on automated large-scale Isabelle builds, e.g., for the Archive of Formal Proofs. This can be done on multiple processes that run independently of each other and greatly increases the usefulness of larger server CPUs with many cores; then, other parameters such as memory bandwidth might be more important and have to be analyzed. However, given the large cost of such machines, it would be much more economical to instead distribute the build on multiple cheap but fast desktop CPUs (especially when latency is a concern, not throughput).

Acknowledgments

Thanks to all the countless individuals who participated in the benchmark. Parts of the performance results have been obtained on systems in the test environment BEAST (Bavarian Energy Architecture & Software Testbed) at the Leibniz Supercomputing Centre. NAMD was developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. Many thanks to Phoronix Media (OpenBenchmarking), PassMark Software (PassMark), RebusFarm GmbH (Cinebench), Primate Labs Inc. (Geekbench) and especially UL Benchmarks (3DMark), for providing their benchmark data.

References

[1] M. Wenzel, Parallel Proof Checking in Isabelle/Isar, Proceedings of the ACM SIGSAM 2009 Internaional Workshop on Programming Languages for Mechanized Mathematics Systems (PLLMS 2009) (2009) 13–21.

[2] D. C. Matthews, M. Wenzel, Efficient parallel programming in Poly/ML and Isabelle/ML, in: DAMP'10 - Proceedings of the 2010 ACM SIGPLAN Workshop on Declarative Aspects of Multicore Programming, ACM Press, New York, New York, USA, 2010, pp. 53–62. doi:10.1145/1708046.1708058.

[3] S. Schulz, M. Möhrmann, Performance of clause selection heuristics for saturation-based theorem proving, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 9706, Springer Verlag, 2016, pp. 330–345. doi:10.1007/978-3-319-40229-1_23.

[4] W. Ertel, Performance Analysis of Competitive OR-Parallel Theorem Proving, Technical Report, Technische Universität München, 1991.

[5] A. Jindal, R. Overbeek, W. C. Kabat, Exploitation of parallel processing for implementing high-performance deduction systems, Journal of Automated Reasoning 8 (1992) 23–38. doi:10.1007/BF00263447.

[6] C. H. Wu, S. J. Lee, Parallelization of a hyper-linking-based theorem prover, Journal of Automated Reasoning 26 (2001) 67–106. doi:10.1023/A:1006469202251.

[7] S. Böhme, T. Nipkow, Sledgehammer: Judgement day, in: J. Giesl, R. Hähnle (Eds.), Automated Reasoning (IJCAR 2010), volume 6173 of *LNCS*, Springer, 2010, pp. 107–121.

[8] J. C. Blanchette, S. Böhme, L. C. Paulson, Extending sledgehammer with SMT solvers, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 6803 LNAI, Springer, Berlin, Heidelberg, 2011, pp. 116–130. doi:10.1007/978-3-642-22438-6_11.

[9] G. Sutcliffe, The TPTP problem library and associated infrastructure: The FOF and CNF Parts, v3.5.0, Journal of Automated Reasoning 43 (2009) 337–362. doi:10.1007/s10817-009-9143-8.

[10] M. D. Hill, M. R. Marty, Amdahl's law in the multicore era, Computer 41 (2008) 33–38. doi:10.1109/MC.2008.209.

[11] riken jp, Introduction to the Himeno benchmark, 2001. URL: https://i.riken.jp/en/supercom/documents/himenobmt/.

[12] J. C. Phillips, D. J. Hardy, J. D. C. Maia, J. E. Stone, J. V. Ribeiro, R. C. Bernardi, R. Buch, G. Fiorin, J. Hénin, W. Jiang, R. McGreevy, M. C. R. Melo, B. K. Radak, R. D. Skeel, A. Singharoy, Y. Wang, B. Roux, A. Aksimentiev, Z. Luthey-Schulten, L. V. Kalé, K. Schulten, C. Chipot, E. Tajkhorshid, Scalable molecular dynamics on cpu and gpu architectures with namd, The Journal of Chemical Physics 153 (2020) 044130. doi:10.1063/5.0014475.

[13] dolfyn, Dolfyn, Open-Source CFD Solver, 2005. URL: https://www.dolfyn.net/.