Relative performance projection on Arm architectures

Clément Gavoille, Hugo Taboada, Patrick Carribault, Fabrice Dupros, Brice Goglin, Emmanuel Jeannot

Relative performance projection on Arm architectures

Clément Gavoille[1,3], Hugo Taboada[1,2], Patrick Carribault[1,2], Fabrice Dupros[4],
Brice Goglin[3], and Emmanuel Jeannot[3]

[1] CEA, DAM, DIF, F-91297 Arpajon, France
{clement.gavoille,hugo.taboada,patrick.carribault}@cea.fr
[2] Université Paris-Saclay, CEA, Laboratoire en Informatique Haute Performance
pour le Calcul et la simulation, 91680, Bruyères le Chatel, France
[3] Inria, LaBRI, Univ. Bordeaux, CNRS, Bordeaux-INP, France
{brice.goglin,emmanuel.jeannot}@inria.fr
[4] ARM, France
{fabrice.dupros}@arm.fr

Abstract. With the advent of multi- many-core processors and hardware accelerators, choosing a specific architecture to renew a supercomputer can become very tedious. This decision process should consider the current and future parallel application needs and the design of the target software stack. It should also consider the single-core behavior of the application as it is one of the performance limitations in today's machines. In such a scheme, performance hints on the impact of some hardware and software stack modifications are mandatory to drive this choice. This paper proposes a workflow for performance projection based on execution on an actual processor and the application's behavior. This projection evaluates the performance variation from an existing core of a processor to a hypothetical one to drive the design choice. For this purpose, we characterize the maximum sustainable performance of the target machine and analyze the application using the software stack of the target machine. To validate this approach, we apply it to three applications of the CORAL benchmark suite: LULESH, MiniFE, and Quicksilver, using a single-core of two Arm-based architectures: Marvell ThunderX2 and Arm Neoverse N1. Finally, we follow this validation work with an example of design-space exploration around the SVE vector size, the choice of DDR4 and HBM2, and the software stack choice on A64FX on our applications with a pool of three source architectures: Arm Neoverse N1, Marvell ThunderX2, and Fujitsu A64FX.

Keywords: Performance Projection · Design space exploration · Arm architecture · Roofline model

1 Introduction

In the pursuit of reaching the exaflops target, the CPUs are becoming more complex both from hardware and software perspectives. Even when working on a multicore processor, it is essential to consider the single-core performance when exploring all the possibilities in its design. Indeed, there are multiple choices to make on the memory hierarchy side and the computational part with, for example, vector units. Therefore, it is meaningful to study the impact of those choices on the software stack and the applications. Considering we have access to a source machine and the software stack of a hypothetical target machine, how can we evaluate the impact of the differences between the machines and software stacks on the application performance?

This paper proposes a methodology to evaluate this impact of single-core performance from a source machine to a hypothetical target machine with a dedicated software stack. By analyzing the differences between two architectures and two binaries, this approach evaluates the performance from one machine to another with a roofline-based model, leading to an interval of performance. The obtained intervals analysis led to a study of the relevance of some hardware modifications and their impact on software. We present such an exploration around hardware vector sizes, various memory types, and different compilers on 3 Arm architectures (Marvell ThunderX2, Neoverse N1, and Fujitsu A64FX) and 3 CORAL mini-apps (Lulesh, MiniFE, and Quicksilver).

Section 2 presents the related work while Section 3 describes the methodology and its implementation. Then Section 4 presents the experimental environment used for approach validation in Section 5 and parameter exploration on 3 Arm architectures in Section 6.

2 Related Work

There are various approaches for evaluating the performance impact regarding design-space exploration. The first one relies on cycle-accurate simulators [11] leading to precise prediction but significant overhead (10000×). This drawback is too limiting for exploring the performance impact on a whole mini-app.

Hence, analytical models can be used, leading to less precise but much faster estimation. The main difficulty lies in defining the relevant metrics and obtaining them. The choices and approximations made to obtain these metrics are different in each model and result in differences in precision and speed. Some analytical approaches choose to reduce the problem by being application-dependent [4]. However, our model can explore different applications as we want to characterize diverse behavior in our applicative workload. Some of the application-independent approaches choose to use simulation on a small scale [14] to have a good prediction and limit the analysis time compared to a complete simulation. Our approach does not rely on a simulator to get metrics but only on the emulation of non-native ISA, which is much faster than fully simulating the application. Furthermore, it allows exploring parameters on applications with a larger input size. While it is possible to consider a hardware-independent representation of the application [9, 8], it is essential to look at the impact of software stack targeting an architecture in an environment as recent and diverse as the Arm HPC environment. The choices in the software stack have a non-negligible impact on performance, as shown on A64FX [5]. Therefore, our approach considers that having access to the target machine software stack is necessary for our model.

The idea of projecting the performance from a source machine to a target machine is behind some machine-learning-oriented approaches [7]. One of the current limitations is the low number of machines in the Arm environment, making it hard to have a sufficient dataset necessary for training. However, we could consider coupling our approach with machine learning as more and more machines appear in the Arm HPC environment.

This article presents an analytical performance projection approach used for design space exploration. It allows to take into account the differences in hardware and in software stack when targeting a particular architecture. The exploration around different hardware parameters in this article leads to a discussion on the effectiveness and the limitations of the approach.

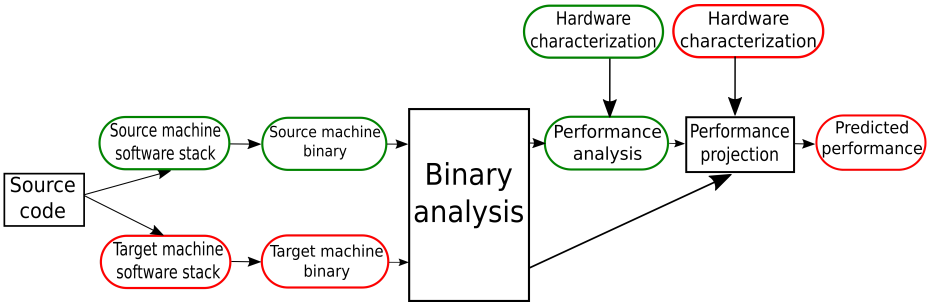

Fig. 1. Our Performance-Estimation Workflow

3 Workflow Presentation and Implementation

This section presents our approach and the methodology and its implementation for validation and design-space exploration. Figure 1 presents the main workflow in which the target and source machine characterization are represented in green and red while the model analysis running on the source machine is black. The first step is to get two binaries with the same source code: one with the software stack of the source machine and the other for the target architecture. Both binaries are then analyzed to gather the metrics directly on the source machine.

In the field of performance analysis, the Roofline model [18] is a well-known representation to characterize the behavior of an application according to the hardware limitations. So, we have chosen to use this representation to analyze performance on our source machine and evaluate a target one. The model output is a performance interval on the roofline representation of the target machine. Moreover, it helps to understand the impacts of the software stack on the target architecture.

3.1 Hardware and Software Characterization

Our approach relies on two binaries (source and target) obtained through a dedicated software stack. We consider the hardware differences thanks to the maximum available bandwidth and peak sustainable performance of the machine. In contrast, the differences brought by the software stack are visible in the metrics we obtain by analyzing the binaries. We have chosen to consider their Operational Intensities (OI) and their floating-point instruction mix. Once we have considered these hardware and software differences, we project the roofline analysis from the source machine to a hypothetical target architecture.

Hence, the first study is to obtain these hardware limitations imposed by the peak memory bandwidth of all memory levels and the peak sustainable performance of our core. These limitations are represented by the roofline (1) of the Stream Triad bandwidth of each memory level $\text{BW}_{\text{STREAM}}$ [13] and the peak performance of High Performance Linpack (Perf_{HPL}) [15]. This leads to two regions: (i) memory-bound limited by the memory bandwidth and (ii) compute-bound where the HPL peak performance represents the limit (see Figure 2).

$$\text{roofline}(\text{OI}) = min\{\text{BW}_{\text{STREAM}} \times \text{OI}, \text{Perf}_{\text{HPL}}\} \tag{1}$$

However, using HPL performance as a limitation is unrealistic because our applications do not have the floating-point instruction mix to reach that performance peak. Hence we have chosen to weigh this peak sustainable performance of a single-core following the equation (2) in which we compare the application floating-point operations per instruction to the maximum attainable on the machine which is only FMA-type of instructions on full vectors. With such a ponderation, the compute part of the roofline represents the maximum sustainable performance for our application instruction mix.

$$\text{Perf}_{\text{HPL}_{\text{ponderated}}} = \frac{\text{Perf}_{\text{HPL}}}{2 \times \frac{\text{vector size}}{\text{datasize}}} \times \frac{N_{\text{floating point operations}}}{N_{\text{floating point instructions}}} \tag{2}$$

The next main component in our model is the Operational Intensity (OI). Because we want to consider the bandwidth of the different memory levels, we need to assess the bytes accessed in these memory levels in the OI as presented in the Cache-Aware Roofline Model [6]. Hence, in a two cache-level machine, we obtain the OI from L1 using the equation (3) with B_i the total of bytes accessed in the cache level i.

$$\text{OI}_{L1} = \frac{N_{\text{floating point operations}}}{B_{L1} + B_{L2} + B_{\text{Main Memory}}} \tag{3}$$

The OI from the L1 memory level is the same OI defined in the CARM approach, and the OI of the main memory is the one used in the Original Roofline Model.

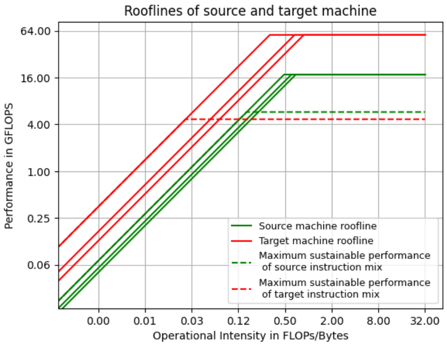

Fig. 2. Roofline representation: plain lines represent, from left to right, the rooflines of L1, L2, and Main Memory for each machine (green is source machine, red is target machine) with the peak sustainable performance obtained with HPL. The maximum attainable performance weighted by instruction mix is in dotted for both machines.

3.2 Performance Projection

The projection uses the same idea as Kwack et al. for roofline projection [12]: it considers the ratio between the performance ($\mathrm{Perf}_{\mathrm{source}}$) and one roofline on the source machine at the $\mathrm{OI}_{\mathrm{source}}$ ($\mathrm{roofline}_{\mathrm{source}}(\mathrm{OI}_{\mathrm{source}})$), and projects this ratio on the target machine using the new OI and the new roofline ($\mathrm{roofline}_{\mathrm{target}}(\mathrm{OI}_{\mathrm{target}})$). This is presented in equation (4). Thus depending on the OI value, the application is limited by the memory-level bandwidth or the core peak performance.

$$\mathrm{Perf}_{\mathrm{target}} = \frac{\mathrm{Perf}_{\mathrm{source}}}{\mathrm{roofline}_{\mathrm{source}}(\mathrm{OI}_{\mathrm{source}})} \times \mathrm{roofline}_{\mathrm{target}}(\mathrm{OI}_{\mathrm{target}}) \tag{4}$$

This analysis results in multiple values because of all the OIs and rooflines, forming a projection interval.

3.3 Methodology for Design-Space Exploration

We want to use this model to explore the parameters best reflected by this projection approach. We can make such an exploration around different software stacks and instruction mixes of the application. But we also consider hardware parameters such as the memory type and bandwidth and the hardware vector length. In the Arm environment, the exploration of the different vector sizes is allowed by the vector-length-agnostic approach of the SVE (Scalable Vector Extension) ISA of Arm architectures [17].

The model translates the hardware differences into rooflines used for projection, whereas the software stack and instruction mix changes are shown in the OI and its peak compute performance.

However, hardware and software changes are often not dissociated because one modification can impact the other. When we change the hardware vector size, we often observe a decrease in vectorization rate, affecting the instruction mix of the application.

Technically, when we consider a different vector size, we multiply the maximum performance obtained with HPL by $\frac{new\ vector\ size}{old\ vector\ size}$ and analyze the target binary again to see the impact this change has on the instruction mix and OI. When we consider different memory types, we do not need to run a new binary analysis as we only change the value of the main memory bandwidth and project performance with this new roofline, with this new value having the most significant impact on memory-bound applications. For example, when we introduce a HBM2 of A64FX on a DDR4 machine, we change the value of the main memory bandwidth to the one we measured on A64FX.

Hence, when running the model with each of the new parameters, we will obtain a different, or not, prediction interval. By comparing these intervals, we can analyze the impact of the evolution of diverse parameters and their impact on the performance of the application we study.

3.4 Implementation

As explained before, the machine characterization is obtained by running Stream and HPL on our source machine. We assume we have access to these benchmarks results or extrapolate this information on the source machine. For the analysis of the binaries we obtain with the different software stacks, we need to gather two kinds of metrics:

1. Instruction mix: number of floating-point instructions, total number of accessed bytes, number of flops. We rely on the dynamic code instrumentation with DynamoRIO [3] and ArmIE for SVE emulation [1] when changing the vector length. ArmIE instrumentation client allows for an easy floating-point instructions, FLOPs and bytes accessed count instruction per instruction, even for emulated SVE instructions on a non-SVE architecture.
2. Memory usage: percentage of hits in every memory level. We rely on hardware counters on the source machine but it is also possible to modelize a cache thanks to an ArmIE memory instruction trace client.

Our implementation is explained in Figure 3 adding precision to Figure 1 with the tool and benchmark used in our implementation.

4 Experimental Environment

This section describes the architectures and the benchmark applications used to validate and experiment our model.

4.1 Architectures

We chose to use three different Arm CPUs to experiment with our approach: a single-core of Marvell ThunderX2 (TX2), Arm Neoverse N1 (N1), and Fujitsu

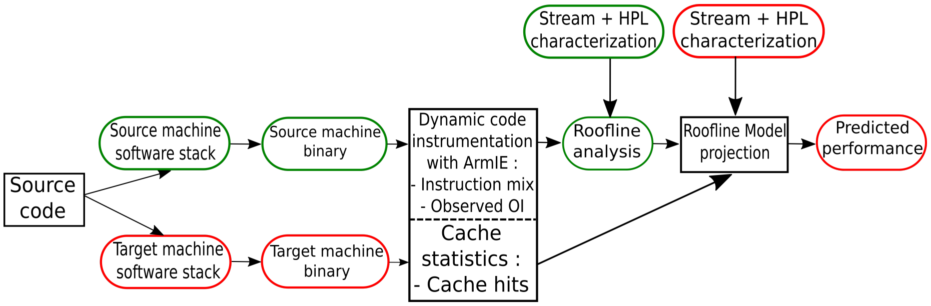
Stream + HPL characterization
Stream + HPL characterization
Source machine software stack
Source machine binary
Dynamic code instrumentation with ArmIE :
- Instruction mix
- Observed OI
Roofline analysis
Roofline Model projection
Predicted performance
Source code
Target machine software stack
Target machine binary
Cache statistics :
- Cache hits

Fig. 3. Description of the implementation flow.

Machine	TX2	N1	A64FX
Performance (GFLOPS)	17.53	18.22	56.71
MM bandwidth (GB/s)	25.43	21.14	65.52
Vector Size and ISA	NEON 128 bits	NEON 128 bits	SVE 512 bits
Memory Type	DDR4	DDR4	HBM2
Compiler	g++ 10.3.0	g++ 10.3.0	g++ 10.3.0 FCC 4.6.3 (clang mode)
Flags	-O3 -ffast-math -mcpu=thunderx2t99	-O3 -ffast-math -mcpu=neoverse-n1	-O3 -ffast-math mcpu=a64fx

Table 1. Single-Core Characteristics the 3 Test Machines.

A64FX (A64FX). Table 1 summarizes their characteristics and the results of HPL and Stream benchmarks running alone on a full node we obtained. These three architectures cover different parts of the Arm HPC environment, from the server market (N1 and TX2 processors) to the HPC focus (Fujitsu A64FX). The latter is currently the only Arm processor in production to use SVE vectors of 512 bits. With HBM2 and longer vectors than N1 and TX2, the single-core performance of a A64FX node is much higher when running STREAM and HPL benchmarks.

4.2 Applications

We use three benchmarks of the CORAL and CORAL-2 Benchmark suites: LULESH [10] , MiniFE [2] and Quicksilver [16].

LULESH (Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics) approximates hydrodynamic equations by using a regular cartesian mesh to partition the spatial problem. Our test's input is a mesh of size 100^3.

MiniFE is a mini-application based on finite element methods that implement an iterative conjugate gradient solver. Our test's input is a mesh of size 256^3.

Quicksilver is a CORAL-2 Benchmark suite mini-application that solves a simplified dynamic Monte-Carlo particle transport problem. Our input is the Coral_2_P1_1 input.

5 Model Validation

This Section validates the model using two close architectures (N1 and TX2) by ensuring that the target performance is in the predicted interval obtained by our workflow when using the same software stack (GCC). Figures 4 to 9 display the prediction interval obtained with the different projections. The blue crosses represent these projections creating the interval depicted by the blue dotted box. The source machine rooflines are green, and the target machine ones are red. We analyze each application after initialization and before finalization.

5.1 LULESH

Figures 4 and 5 present the projection of LULESH from one machine to the other. The maximum sustainable performance weighted by the floating-point instruction mix (corresponding to the dotted rooflines) is a bit higher on TX2 than N1 despite having a lower maximum performance on HPL. The OIs of the L1 memory level are similar on both machines. Situated in the TX2 memory-bound region, the differences between the bandwidth and the projections in this region create an interval that is not modified by the projections from the OIs of L2 and main memory. This interval is higher when projecting from TX2 because of the difference in L1 and L2 cache bandwidth. Because performances on both machines are nearly equal, we are closer to the TX2 roofline hence we obtain a better ratio which is then translated into a higher prediction interval. In both figures, the interval we predict includes the actual performance measured on the target machine, validating our approach in this application.

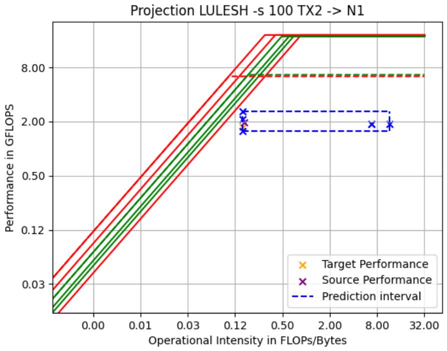

Fig. 4. Results on LULESH TX2 → N1.

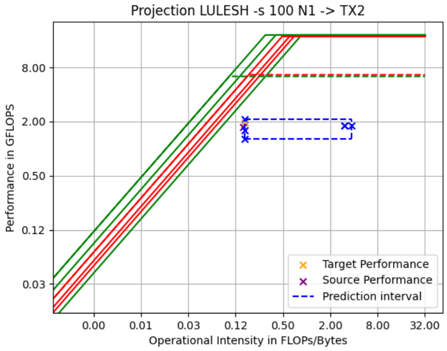

Fig. 5. Results on LULESH N1 → TX2.

5.2 MiniFE

MiniFE exploits vectorization on the two architectures. This better vectorization rate is translated into a good performance of its instruction mix (see Figures 6 and 7). Compared to LULESH, the OI of L1 is in the memory-bound region of all rooflines on both machines. Once again, the interval we predict, only being affected by the OI of L1, does not change whether we project from N1 or TX2. However, the N1 performance is higher (1.87 GFLOPS) than the TX2 performance (1.04 GFLOPS). Despite this difference in performance, our interval includes the measured performance. We can suppose that, because we are in the memory-bound region of the L1 and L2 cache levels, the better performance of MiniFE on N1 may result from the higher bandwidth of these levels.

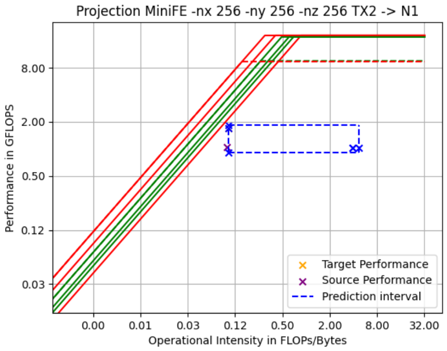

Fig. 6. Results miniFE TX2 → N1.

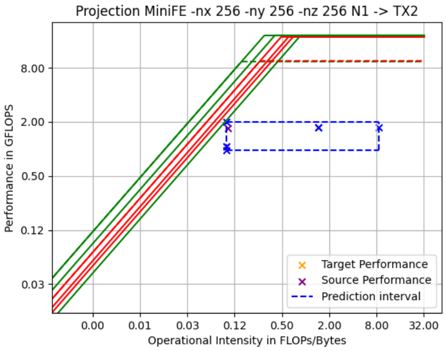

Fig. 7. Results miniFE N1 → TX2.

5.3 Quicksilver

Quicksilver is our application with the lowest OI and measured performances on both machines (Figures 8 and 9). The low performance may result from the poor vectorization rate of this application, shown in the maximum performance attainable by the instruction mix of both binaries. All the OI deducted from the L1 are in the memory-bound region of all rooflines, while the OIs derived from other memory levels are in the compute-bound regions. The prediction interval is obtained because of the OI from L1, which includes the measured performances. We observe higher performance on N1 (0.5 GFLOPS) than TX2 (0.4 GFLOPS). This difference in performance may be due to the difference in cache bandwidth, giving an advantage to the N1 core.

To conclude this validation, when we apply our model on the most similar machine in our machine pool, the prediction interval we obtain always includes the measured performance of our application. For the most memory-bound application (MiniFE and Quicksilver), we also observe higher performance when

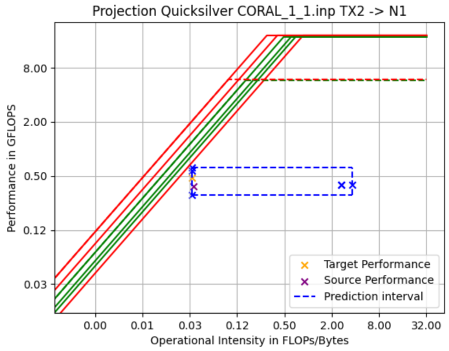

Fig. 8. Results Quicksilver TX2 → N1.

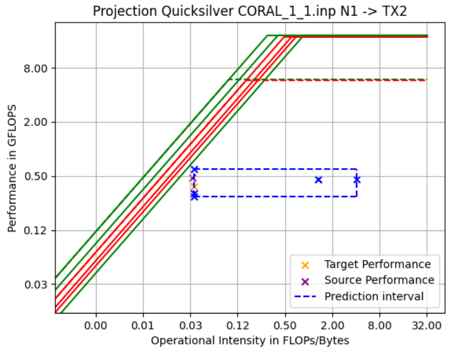

Fig. 9. Results Quicksilver N1 → TX2.

running on an N1 core that may be enabled by the higher bandwidth of the cache levels.

6 Exploration on different parameters

This section will use our approach to explore different hardware and software parameters. We have chosen to explore the different vector sizes allowed by SVE on all three machines. Hence, we compare the performance projection from a NEON machine (N1, TX2) to a hypothetical one with SVE with a vector size of 128, 256, 512, 1024, and 2048 bits. Another parameter we explore is the introduction of HBM2 for both DDR4 machines. Then, we combine these parameters to compare hypothetical SVE512 + HBM2 machines with A64FX. Finally, we observe the differences a change of software stack creates in exploring different vector sizes on A64FX. The SVE512 value is not a projection for A64FX in the following figures as it is native on this core.

6.1 Exploration on SVE vector sizes

One of the challenges in the design of future Arm core is the size imposed by the hardware on SVE vectors and the impact this choice has on the performance of the applications. We can obtain such a characterization with our model by looking at how such a change impacts the maximum performance imposed by hardware and the software stack.

Figure 10 shows that the impact of the vector size on LULESH depends on the source machine. We observe that, when targeting A64FX and TX2 architecture, the binary's predicted performance benefits more from this increase in vector size than when targeting N1. GCC does not vectorize LULESH as much when targeting N1. Despite having a very similar source performance on native N1 and TX2, this difference in vectorization predicts lower performance on N1 than TX2 with longer SVE vectors.

When doing this exercise on MiniFE (Figure 11), we observe here a similar behavior on all machines. A change in vector size impacts all the predicted performances of our architectures. But this impact is not equivalent for all our architectures. When comparing TX2 and N1, the predicted interval upper bound of TX2 gains more performance at each step to reach a maximum of 10.2 GFLOPS.

The behavior of MiniFE when exploring vector size is opposed to Quicksilver (Figure 12). This application does not benefit from the change of vector size on any architectures. On all architectures, GCC cannot vectorize the application, meaning they do not benefit from this change of vector size. If we want to gain performance on Monte-Carlo applications, increasing the vector size is not the solution.

The predicted interval is smaller on A64FX than on the other two machines on all these figures. This shows that there are not many differences when projecting performances with different bandwidths of these machines. We can suppose it is because of the bandwidths of the A64FX being much higher, meaning the OIs of our application are closer to the compute-bound region for all bandwidths.

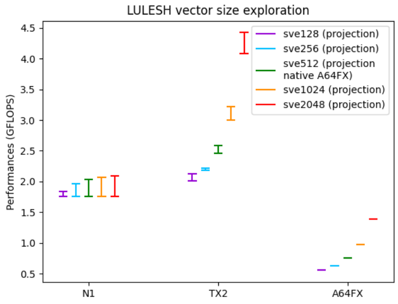

Fig. 10. Exploration of different SVE vector sizes on LULESH

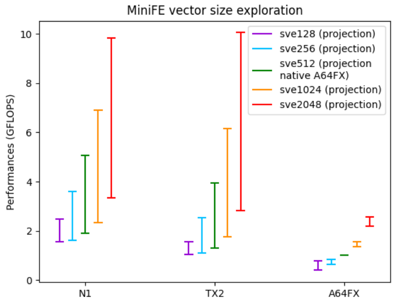

Fig. 11. Exploration of different SVE vector sizes on MiniFE

6.2 Exploration on the introduction of HBM2 on DDR4 machines

Another characterization we make with our approach is to analyze the introduction of the HBM2 memory of A64FX on N1 and TX2, creating a hypothetical machine with the same characteristics as our source machine with only the main memory bandwidth being different in our model. The first observation on Figures 13 and 14 is that the N1 core can be the one that benefits the most from this change of main memory bandwidth on LULESH and Quicksilver. This leads to a higher predicted upper bound on both applications on N1 despite LULESH having less performance with DDR4 on this machine. We suppose this is due to the N1 core having higher cache bandwidth and lower memory bandwidth than TX2. So the memory bandwidth gain is higher for the N1 core, leading

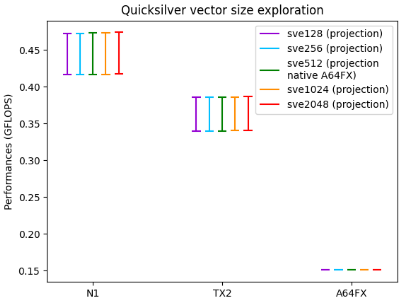

Fig. 12. Exploration of different SVE vector sizes on Quicksilver

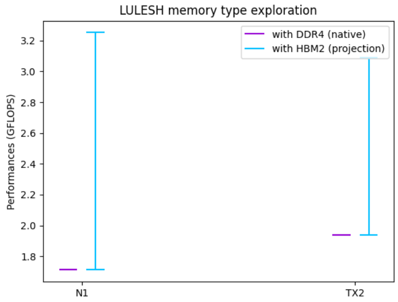

Fig. 13. Exploration of introduction of HBM2 on LULESH

to more performance gain for these applications. We also see that the lower bound of our predicted interval does not change on both applications compared to DDR4. This is due to the cache bandwidth of our hypothetical machine not being adapted to this main memory bandwidth increase. Finally, our model cannot characterize the latency aspect of our applications, which may be an issue with the introduction of HBM2 because of its latency access being higher than DDR4.

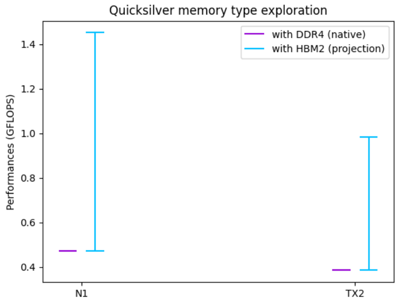

Fig. 14. Exploration of introduction of HBM2 on Quicksilver

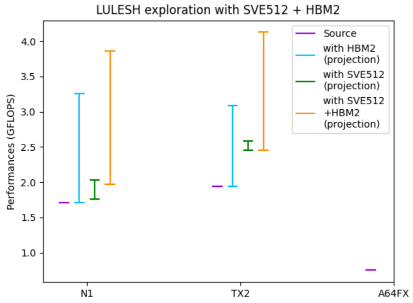

Fig. 15. Exploration of introduction of HBM2 and SVE512 on LULESH

6.3 Comparison of projections from N1 and TX2 with SVE 512 and HBM2 to A64FX

We combine both changes of parameters we made in the two previous subsections to compare the introduction of SVE 512 bits and HBM2 on N1 and TX2 and compare it with the A64FX architecture on two applications: LULESH and Quicksilver. We choose to use GCC 10.3 on all 3 machines for this comparison presented in the Figures 15 and 16. We can observe the change introduced by HBM2 to the interval predicted only with SVE512. Similarly, the use of HBM2 impacts the N1 core the most on LULESH, even with SVE512. Even if both machines can gain more performance, this leads to similar predicted performance between N1 and TX2 despite LULESH not benefitting from vectorization on N1. On MiniFE, the predicted performance is not as impacted on both applications. We only observe the predicted upper bound being higher by 0.6 GFLOPS on N1 and no change on TX2. This analysis shows it can be more impactful on performance to increase vector size than increasing the main memory bandwidth for MiniFE.

When we compare the projection on both applications to the performance on the A64FX machine, we predict performance to be higher on N1 and TX2 architecture. We can suppose the introduction of HBM2 and SVE512 on these machines changed their single-core roofline to be on par with A64FX, and GCC is more efficient when targeting N1 and TX2 architecture than A64FX, causing this higher predicted performance.

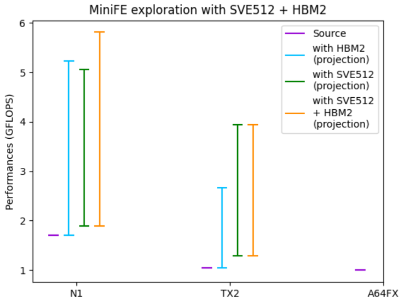

Fig. 16. Exploration of introduction of HBM2 and SVE512 on MiniFE

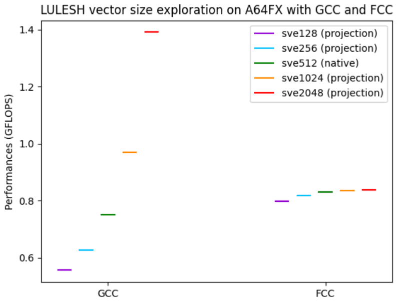

Fig. 17. Vector sizes exploration with GCC and FCC on A64FX on LULESH

6.4 Vector sizes exploration on A64FX with different software stacks

We have seen that GCC has a hard time obtaining performance on a single-core of A64FX, and we want to compare it with the use of the Fujitsu Compiler (FCC). Figure 17 presents this comparison when changing the vector sizes on LULESH with GCC and FCC compilers. We do not have an interval with both software stacks, meaning the OIs of both binaries are in the compute-bound region of A64FX. However, we observe a different evolution of the predicted value when increasing the SVE vector size. GCC binary gains more performance when increasing vector size when compared to the FCC binary because it has more vector usage. Despite this difference in vectorization, we observe higher performance on FCC with SVE vectors from 128 bits to 512 bits, with the last being the native vector size. We can suppose that FCC is more careful when vectorizing the application because of its insight of the microarchitecture impact of the A64FX, whereas GCC vectorizes a loop without considering it as much. So we have better usage of an A64FX when compiling with the Fujitsu Compiler than with GCC.

7 Conclusion and Future Work

This article presents an approach for core design space exploration of a single processor core using performance projection from a source machine. We have chosen to consider the impact of the software stack of the target machine. The workflow relies on binary analysis, hardware characterization, and the Roofline model to obtain a performance interval on the target machine. Thanks to this projection, we can characterize the impact of the differences between a single core maximum performance, the bandwidth of all memory levels, and the cache efficiency. We can also analyze the differences brought by the software stack on the application with metrics such as the observed OI and maximum performance of the instruction mix with the use of the SIMD mechanism. Thanks to an implementation using emulation for dynamic code instrumentation, we have validated our model on a core of Marvell ThunderX2 and Arm Neoverse N1 architecture for three CORAL mini-apps: LULESH, MiniFE, and Quicksilver. We followed this validation work with an exploration around different SVE vector sizes and the introduction of HBM2 memory on DDR4 machines for the three CORAL applications. We used a pool of three different Arm core architectures for this exercise: ThunderX2, Neoverse N1, and A64FX. We also analyzed the impact of using different compilers (GCC and FCC) when exploring different SVE vector lengths on A64FX. To enhance the model, we plan to characterize some microarchitectural features and parallelism at a full-node level.

RIKEN Acknowledgment : The work presented in this paper has been executed in the context of the CEA and RIKEN collaboration on High Performance Computing, Artificial Intelligence and Quantum Computing.

We thank the RIKEN team for granting us access to the Fugaku supercomputer in the context of our bilateral collaboration.

Authors' contributions :

Clément Gavoille: Conceptualization, Methodology, Software, Validation, Investigation, Writing - Original Draft, Writing - Review & Editing, Visualization

Hugo Taboada: Conceptualization, Methodology, Resources, Writing - Original Draft, Supervision

Patrick Carribault: Conceptualization, Methodology, Resources, Writing - Original Draft, Supervision

Fabrice Dupros: Resources, Writing - Original Draft, Supervision.

Brice Goglin: Conceptualization, Writing – Original Draft, Supervision.

Emmanuel Jeannot: Conceptualization, Writing – Original Draft, Supervision.

References

1. Arm: Arm instruction emulator, https://developer.arm.com/tools-and-software/server-and-hpc/compile/arm-instruction-emulator
2. Barrett, R.F., Heroux, M.A.: The mantevo projectmini-applications: Vehicles for co-design. (2013)
3. Bruening, D., Garnett, T., Amarasinghe, S.: An infrastructure for adaptive dynamic optimization. In: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization. CGO '03 (2003)
4. Davis, J., Mudalige, G., Hammond, S., Herdman, A., Miller, I., Jarvis, S.: Predictive analysis of a hydrodynamics application on large-scale cmp clusters. Computer Science - Research and Development **26** (2011)
5. Domke, J.: A64fx – your compiler you must decide! (2021)
6. Ilic, A., Pratas, F., Sousa, L.: Cache-aware roofline model: Upgrading the loft. IEEE Computer Architecture Letters (2014)
7. Ipek, E., de Supinski, B.R., Schulz, M., McKee, S.A.: An approach to performance prediction for parallel applications (2005)
8. Jongerius, R., Anghel, A., Dittmann, G., Mariani, G., Vermij, E., Corporaal, H.: Analytic multi-core processor model for fast design-space exploration. IEEE Transactions on Computers (2018)
9. Jongerius, R., Mariani, G., Anghel, A., Dittmann, G., Vermij, E., Corporaal, H.: Analytic processor model for fast design-space exploration. In: 2015 33rd IEEE International Conference on Computer Design (ICCD) (2015)
10. Karlin, I., Keasler, J., Neely, R.: Lulesh 2.0 updates and changes. Tech. Rep. LLNL-TR-641973 (2013)
11. Kodama, Y., Odajima, T., Asato, A., Sato, M.: Evaluation of the riken post-k processor simulator (04 2019)
12. Kwack, J.H., Arnold, G., Mendes, C., Bauer, G.: Roofline analysis with cray performance analysis tools (craypat) and roofline-based performance projections for a future architecture. Concurrency and Computation: Practice and Experience (2018)
13. McCalpin, J.: Memory bandwidth and machine balance in high performance computers. IEEE Technical Committee on Computer Architecture Newsletter (1995)
14. Obaida, M.A., Liu, J., Chennupati, G., Santhi, N., Eidenbenz, S.: Parallel application performance prediction using analysis based models and hpc simulations. In: Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. SIGSIM-PADS '18 (2018)
15. Petitet, A., Whaley, R., Dongarra, J., Cleary, A.: Hpl – a portable implementation of the high-performance linpack benchmark for distributed-memory computers (2008)

16. Richards, D., Brantley, P., Dawson, S., Mckenley, S., O'Brien, M.: Quicksilver, version 00
17. Stephens, N., Biles, S., Boettcher, M., Eapen, J., Eyole, M., Gabrielli, G., Horsnell, M., Magklis, G., Martinez, A., Premillieu, N., Reid, A., Rico, A., Walker, P.: The arm scalable vector extension. IEEE Micro **37** (2017). https://doi.org/10.1109/MM.2017.35
18. Williams, S., Waterman, A., Patterson, D.: Roofline: An insightful visual performance model for multicore architectures. Commun. ACM (2009)