Work-in-progress: Impact of compilation optimization levels on execution time variability

Mohamed Amine Khelassi, Yasmina Abdeddaïm

▶ **To cite this version:**

Mohamed Amine Khelassi, Yasmina Abdeddaïm. Work-in-progress: Impact of compilation optimization levels on execution time variability. International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, Sep 2024, Padova, Italy. hal-04667091

HAL Id: hal-04667091
https://hal.science/hal-04667091v1

Submitted on 23 Aug 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Impact of compilation optimization levels on execution time variability

Mohamed Amine Khelassi
Univ Gustave Eiffel, CNRS, LIGM
F-77454 Marne-la-Vallée, France
mohamedamine.khelassi@esiee.fr

Yasmina Abdeddaïm
Univ Gustave Eiffel, CNRS, LIGM
F-77454 Marne-la-Vallée, France
yasmina.abdeddaim@esiee.fr

***Abstract*—Compiler optimizations play a crucial role in enhancing software performance by improving execution speed and reducing resource consumption. However, these optimizations can also introduce variability in execution times, a significant concern for real-time systems where predictability is paramount. This paper investigates the relationship between GCC compiler optimizations and the execution time variability. By compiling a set of benchmark programs under different optimization levels (O0, O1, O2, O3), we analyze the impact on execution time variability using the WCET/BCET ratio, and the dispersion of the execution times around the WCET.**

***Index Terms*—Compilation optimisation levels, Execution time variability.**

I. Introduction

Compiler optimization options allow a user to change the final binary code of a program without changing the original high level source code. Using compiler optimizations options may result in a program performing better, in term of execution time, than the non optimized version. The major challenge in compilation is choosing the right set of compiler optimization sequence, taking in fact that these code optimizations are programming language, application and architecture dependent. Additionally, there is the problem of choosing which optimization options to apply and the order of applying these optimizations.

Real-time systems are designed to perform critical tasks within stringent timing constraints, where the predictability of execution times is more important than their minimization. In such systems, the Worst-Case Execution Time (WCET) is a key metric used to ensure that tasks always meet their deadlines.

Using compiler optimizations in real-time systems can enhance programs performances , but it can also negatively impact their execution times. Indeed, while these optimizations can lead to performance gains, they can also introduce variability in execution times, posing challenges for WCET analysis and the predictability of real-time systems. By execution time variability we refer to the variations in the time it takes for a program to execute across different runs. This variability can be influenced by several factors, including compiler optimizations, hardware characteristics, operating system behavior, and input data variations. While there is no agreement on a formal definition of time variability, we find in the literature [1] definitions like ratio between the $WCET$ and the best case execution time $BCET$.

The impact of compiler optimizations on execution time variability is not fully understood. This variability complicates the task of accurately estimating WCET, which is crucial for the design and verification of real-time systems. Understanding how different optimizations affect execution time variability can lead to more reliable WCET estimations and, consequently, more robust real-time systems.

This work in progress aims to investigate the relationship between compiler optimizations and execution time variability in the case of the GCC compiler on the Armv8-A architecture. The objectives are:

- To identify and analyze the impact of various GCC compiler optimizations (O0, O1, O2, O3) on the execution time variability of programs using WCET/BCET ratios, and the dispersion of the execution times around the WCET using the VWCET parameter [2], [3].
- To provide insights and guidelines for selecting compiler optimizations that balance performance improvements while managing the variability of execution times in real-time systems.

II. Comparative study

Our study involves an examination of a set of benchmark programs compiled under different GCC optimization levels $(O0, O1, O2, O3)$. We will analyse the impact of the optimization levels $(O0, O1, O2, O3)$ on the following metrics, the average execution time ACET, the WCET and the BCET to see what is the best optimization level for each program if our goal is to increase the execution time performance. Then we compare this analysis with the impact of optimization on time variability. To do this, we use the metrics presented in the next section.

A. Time variability metrics

To estimate the execution time variability we consider two parameters the WCET/BCET ratio and the VWCET parameter [2], [3]. The WCET/BCET is an indicator of the range or spread of execution times for a program. This ratio provides insight into the variability of execution times and the potential unpredictability of a program's performance. The smaller the

ratio, the less variation there is between the largest and smallest execution time values.

The $VWCET$ parameter measures the dispersion of a set of n execution times $\{X_1, \dots X_n\}$ of a program around the WCET.

$$VWCET = \frac{\frac{\sum_{i=1}^{n}(WCET - X_i)}{n}}{WCET} \times 100 \quad (1)$$

This parameter was proposed to compensate the fact that the ratio gives no information on how the execution times are distributed. This information is important to properly dimension a system. Indeed, execution times that are more concentrated on the BCET may indicate that using the WCET as a program execution time may lead to under-utilization of the platform, whereas if the data is more concentrated on the WCET, it may be easier to estimate the WCET. Given that the VWCET is based on the sum of deviations, $(WCET - X_i)$, between the WCET and the other execution times, the smaller it is the more likely the data are shifted towards the WCET.

B. Experiments and discussion

We perform experiments on the Zynq ZCU104 platform, featuring a quad-core Arm Cortex A53. We generate a set of execution times by executing 4 programs (bubble sort, dijkstra, powerwind, matrix1) 1000 times using RT-bench an Extensible Benchmark Framework for the Analysis and Management of Real-Time Applications [4], we note that the input of the programs are not changed.

We consider in our study that a program is more predictable if it has the lowest ratio value and the lowest $VWCET$ value suggesting that the $WCET$ is not a rare event.

We identify 4 different cases depending on the similarity of the obtained results between the execution time metrics (WCET, BCET, ACET) and the time variability metrics:

- **Case 1: Concordant results**
  In Fig. 1 we notice that increasing the optimization level from $O0$ to $O3$ decreases the $WCET, BCET$ and the $ACET$. We notice that $O3$ produces the lowest $WCET, BCET$ and $ACET$.
  Concerning time variability, in Fig. 2 $O3$ also produced the lowest ratio and the lowest VWCET. This means that $O3$ decreases the gap between the $BCET$ and $WCET$, and according to the $VWCET$ the execution times are more shifted towards the $WCET$.
- **Case 2: Adversarial results**
  In Fig. 3 the optimization level $O2$ yields the best solution considering the $WCET, BCET$ and the $ACET$. Concerning the time variability in Fig. 4 $O0$ is the best solution, however the WCET, BCET, ACET results of $O0$ are the worst ones.
- **Case 3: Trade off results**
  In Fig. 5 we notice that $O3$ yields the best $WCET, BCET$ and $ACET$. For the time variability results in Fig. 6 we notice $O2$ is the best solution. Although $O2$ is not the optimal optimization for $WCET, BCET$ and $ACET$, it is not far from the optimal solution, so we can say that $O2$ is a good trade-off solution to ensure both execution time optimisation and time variability minimisation.
- **Case 4: Objective-dependant results**
  In Fig. 7 $O0$ yields the best solution for optimizing the $WCET$, $O3$ yields the best solution for optimizing the $BCET$ and $ACET$. For the time variability in Fig. 8, $O0$ is the best solution.

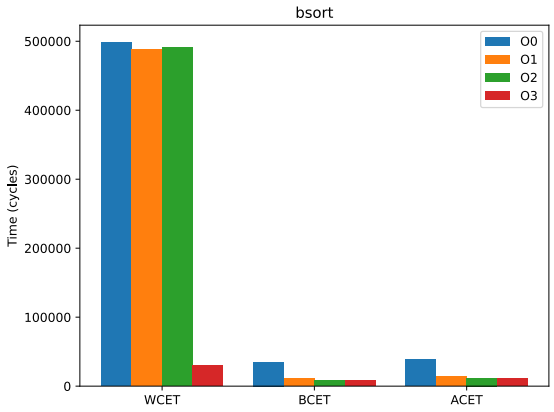

Fig. 1: bubble sort execution times

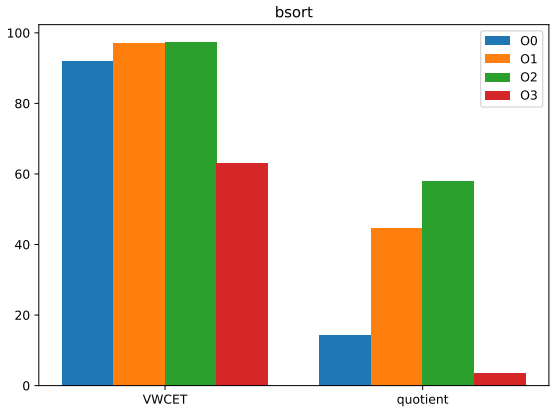

Fig. 2: bubble sort VWCET and quotient

Throughout these first experiments we can draw some conclusions :

- We confirm that applying increased compiler optimization options ($O3$ instead of $O0$) doesn't necessarily produce the desired outcome, we take as en example the program 'matrix1', if the goal is to decrease the $WCET$ applying the different compiler optimizations actually increases the $WCET$, same thing for the program 'dijkstra' increasing the optimization level from $O2$ to $O3$ actually increases the $WCET, BCET$ and $ACET$
- In case 1, the execution time metrics yields the same results as the time variability results suggesting $O3$ as an optimal optimization level, we call it concordant results as the suggestion of the execution time metrics matches that of the time variability parameters.

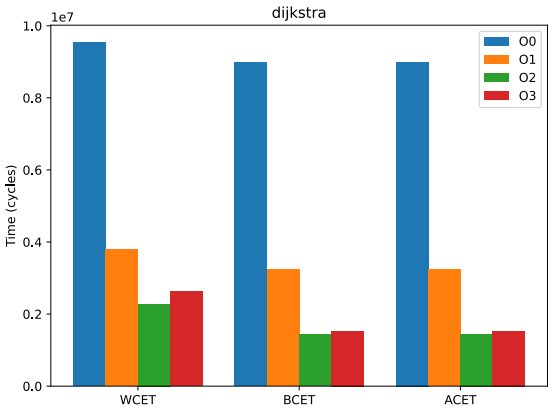

Fig. 3: dijkstra execution times

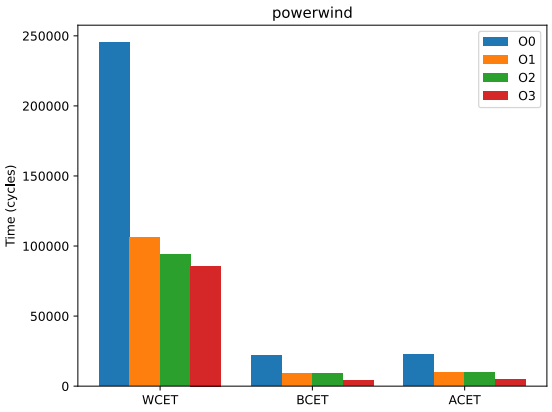

Fig. 5: powerwind execution times

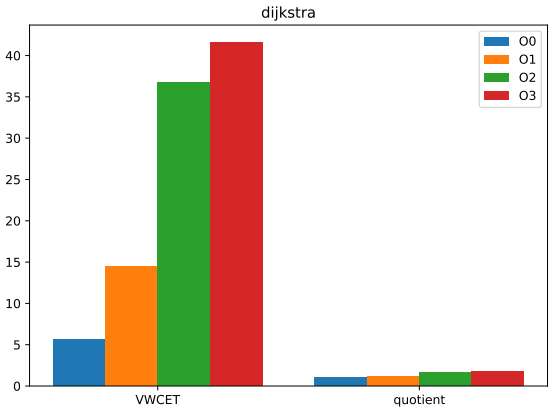

Fig. 4: dijkstra VWCET and quotient

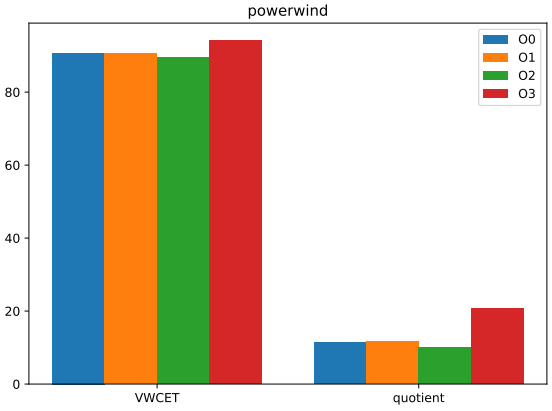

Fig. 6: powerwind VWCET and quotient

- In case 2, the results are adversarial in the sense that the best case in execution time metrics gives the worst results in time variability, if it is critical to ensure predictability of execution times we should choose optimization level $O0$ even if it is the one with the highest execution time. However, it is important to note that there is no universal solution, each scenario needs to be studied according the designers objectives.
- In case 3, the best result is trade off in the sense that the best solution suggested by the time variability ($O2$) is not far from the best solution suggested by the execution time metrics, hence the best solution suggested by the time variability parameters ($O2$) is a balanced choice, offering a compromise that minimizes execution time variability while maintaining acceptable execution performance.
- In case 4, the results in execution time metrics are not the same, $BCET$ and $ACET$ suggest $O3$ and $WCET$ suggests $O0$. If our aim is to minimize the WCET, $O0$ optimization level, is the best option as it also minimize the time variability. This is not the case if our goal is to optimize the $BCET$ or the $ACET$. The best choice is objective dependant in the sense that it depends on the metric that we want to optimize.

As a general conclusion we conclude that relying only on the $WCET$, $BCET$ and $ACET$ as a criteria when choosing the best compiler optimization level is not sufficient for the design of real-time systems.

Note that in this analysis, we have considered that the solution with the execution time closest to $WCET$ is the best. However, we can consider that for some tasks, we allow some deadline misses and that for these programs, we give priority to optimization leading to execution times closer to $BCET$ and therefore with the highest $VWCET$.

III. Related work

Compiler optimizations are crucial for enhancing the performance and efficiency of software applications. Their use depends on the objectives, in the literature [5] we find that these objectives aim to optimize the mean execution time, the code size, the power consumption, the runtime memory used, trying to balance between these objectives is a challenging task [6]. Take in mind that there is no guarantee that the transformed code will behave better than the original code [5], in fact aggressive optimizations can even degrade the performance of the code that they are trying to optimize. Our work in progress is the first work that aims to deal with the link

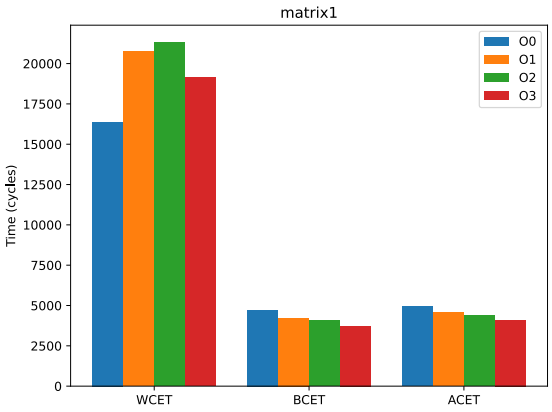

Fig. 7: matrix1 execution times

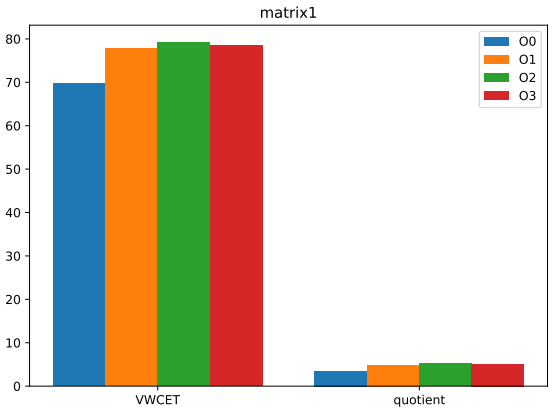

Fig. 8: matrix1 VWCET and quotient

between execution time variability and compiler optimizations as opposed to the previously mentioned objectives of using compiler optimizations. We also find in the literature research done on compilation for predictability [7] where the goal is to transform the source code into sequential pieces of predicated code of the same functional behavior in order to enforce equal timing for all the different contexts, this can be achieved by means of inserting/removing instructions into the code, contrary to our approach where we do not want change the high level source code.

IV. Conclusion and Future Work

In this study, we examined the behavior of execution times of programs compiled under different GCC optimization levels. Different compiler optimization levels may enhance the performance or average execution time of a program, however, in real-time systems execution time variability is more important than the average execution time to ensure predictability.

Using a set of experiments we noticed that there is no universal solution for choosing the best compiler optimization level, each scenario requires its appropriate optimization strategy, in this paper we identified four different case studies with different solutions.

We plan to continue investigating this problem by examining the hardware performance counters, including the number of instructions executed and cache miss rates (L1 instruction and data cache misses, L2 cache misses) and their behavior with different optimizations sequences, to gain a deeper understanding of the underlying causes of execution time variability.

Inspired by [8], [9] that uses machine learning models to choose the appropriate compiler optimization sequence to minimize the WCET and the runtime memory footprint, we also plan to develop models for choosing the best compiler optimization sequence in order to control the execution time variability.

References

[1] M. Schoeberl, "Is time predictability quantifiable?" in *2012 International Conference on Embedded Computer Systems (SAMOS)*. IEEE, 2012, pp. 333–338.

[2] M. A. Khelassi and Y. Abdeddaïm, "Leveraging mixed criticality task budgets," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2024, accepted.

[3] M. A. Khelassi and Y. Abdeddaïm, "Execution time budget assignment for mixed criticality systems," in *10th International Workshop on Mixed Criticality Systems at the Real Time Systems Symposium (RTSS 2023)*, 2023.

[4] M. Nicolella, S. Roozkhosh *et al.*, "Rt-bench: An extensible benchmark framework for the analysis and management of real-time applications," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, 2022, pp. 184–195.

[5] A. H. Ashouri, W. Killian *et al.*, "A survey on compiler autotuning using machine learning," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–42, 2018.

[6] G. Palermo, C. Silvano *et al.*, "Multi-objective design space exploration of embedded systems," *Journal of Embedded Computing*, vol. 1, no. 3, pp. 305–316, 2005.

[7] E. J. Maroun, M. Schoeberl *et al.*, "Compiling for time-predictability with dual-issue single-path code," *Journal of Systems Architecture*, vol. 118, p. 102230, 2021.

[8] V. Pasquale and I. Puaut, "Winston: Revisiting iterative compilation for wcet minimization," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, 2022, pp. 151–161.

[9] J. Chang and D. Park, "Work-in-progress: Searching optimal compiler optimization passes sequence for reducing runtime memory profile using ensemble reinforcement learning," in *Proceedings of the International Conference on Embedded Software*, 2023, pp. 3–4.