Lina: Timing-Constrained High-Level Synthesis Performance Estimator for Fast DSE

André Bannwart Perina and Jürgen Becker
Karlsruhe Institute of Technology
Karlsruhe, Germany
Emails: abperina@usp.br, juergen.becker@kit.edu

Vanderlei Bonato
University of São Paulo
São Carlos, Brazil
Email: vbonato@usp.br

***Abstract*—The adoption of Field-Programmable Gate Array (FPGA) for general use in the High-Performance Computing scenario has been limited by its complex development flow required to get optimised designs coupled with a time-consuming compilation. High-Level Synthesis (HLS) tools are adopted to improve programmability, however the developer must perform several iterations of optimisation schemes in order to achieve reasonable performance results, which is tedious and not trivial. Several works employ Design Space Exploration (DSE) through different optimisation possibilities, coupled with fast performance estimators to avoid the unacceptable compilation times. This paper presents Lina, an expansion of the Lin-Analyzer fast peformance estimator for C/C++ HLS including timing-constrained scheduling and an extended analysis for nested loops. Results over the PolyBench benchmark show that the average relative error dropped from 8.85% to 3.02% when loop unrolling and pipelining directives were considered. As a result Lina becomes a better estimator for non-perfect loop nests and for different timing constraints, which can be adopted as an additional design space exploration knob.**

I. Introduction

In an effort to increase the compute capability of High-Performance Computing (HPC) systems while maintaining the energy consumption to acceptable levels, alternate architectures such as Graphics Processing Units (GPU) and Field-Programmable Gate Arrays (FPGA) are employed to increase the performance per Watt ratio.

High-Level Synthesis (HLS) tools such as Xilinx Vivado and Intel FPGA SDK for OpenCL are able to generate designs for FPGA from high-level software codes, reducing the FPGA programming effort. However, "software-to-hardware" conversion is not trivial and often leads to poor performance results, requiring iterations of optimisation to achieve speedup [1]. One possibility is to iteratively explore the software code using Design Space Exploration (DSE) of optimisation parameters. Since the design space can easily explode in complexity, a fast analyser for each design point is essential [2][3].

This paper presents Lina[1], which is in essence an expansion of the Lin-Analyzer [2] fast C/C++ performance estimator for Xilinx Vivado HLS. It expands Lin-Analyzer by providing timing-constrained scheduling analysis (enabling frequency as an additional exploration knob) and the capability to estimate more complex loop nests. Experimental results show that for a small design space with 48 points, the average relative error for 9 kernels was of 13.01% when array partitioning was explored and 3.02% without it, respectively, against Lin-Analyzer's 16.45% and 8.85%.

The paper is structured as follows: Section II presents the original Lin-Analyzer and Lina. Section III presents the experimental setup for validation, with the results at Section IV. Section V presents related works and finally Section VI concludes the paper.

II. Overview

Lin-Analyzer estimates the cycle count for a C/C++ loop nest by scheduling a Dynamic Data Dependence Graph (DDDG) generated through analysis of a dynamic trace, which is in turn generated through instrumented execution. DDDGs are dynamic variants of Data Flow Graphs (DFG) and can be scheduled to hardware operations using similar approaches as the HLS tools with As-Soon-As-Possible (ASAP), As-Late-As-Possible (ALAP) and Resource-Constrained List Scheduling (RCLS) algorithms. The instrumented execution is performed only once per application and different combinations of optimisations parameters can be explored simply by modifying the DDDG scheduling, which is fast. Thus, Lin-Analyzer is able to explore large design spaces.

Lina expands Lin-Analyzer by implementing a Timing-Constrained Scheduler (TCS) and Non-Perfect Loop Analysis (NPLA). Lina also brings some adjustments in inner values of Lin-Analyzer (e.g. the number of cycles to test the enter/exit conditions of a loop) to improve accuracy. Fig. 1 provides an overview of the Lina framework, encompassing the original instrumentation part and the improved DDDG building/scheduling.

A. Timing-Constrained Scheduler

With HLS, software instructions are allocated to Functional Units (FU) in FPGA. FUs are capable of solving basic operations such as floating-point arithmetic, load, store, etc. In terms of timing, an FU is defined by its latency l (amount of cycles it must iterate to provide a result) and critical path delay t_{cp} (the delay it takes to finish a cycle processing). To provide valid results, the operational frequency f_{op} of the system must be adjusted so that all FUs are able to compute their cycles without timing violations (i.e. $t_{cp} < \frac{1}{f_{op}}$).

[1]Lina is available at https://github.com/comododragon/lina

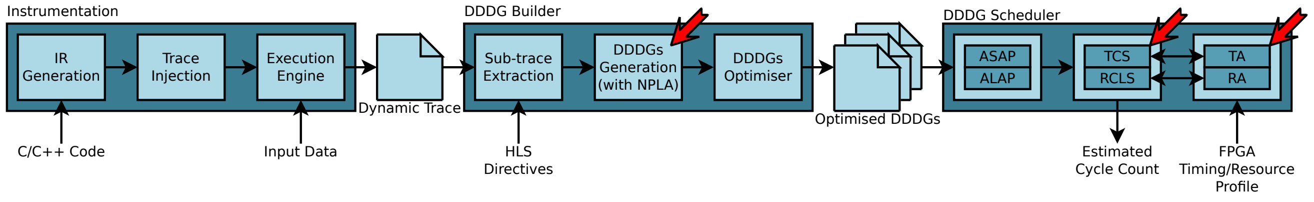

Fig. 1. Lina framework, with Lin-Analyzer modules and the features implemented by Lina marked with red diagonal arrows. Adapted from [2].

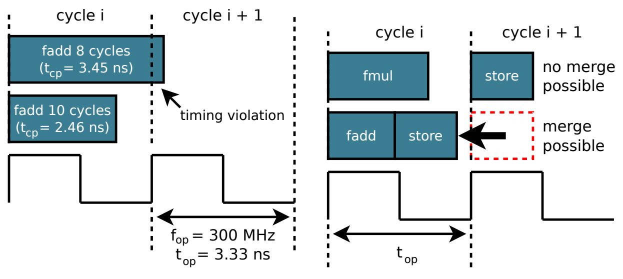

Fig. 2. Timing analysis performed by Lina exemplified at 300MHz. At left (a) an example of FU adjust to avoid timing violation (the `fadd` must be adjusted to 10 cycles, as with 8 cycles the critical path violates timing). At right (b) two examples of cycle merge attempts.

To avoid violations, an HLS compiler may supply a set of configurations $\{(l^0, t^0_{cp}), (l^1, t^1_{cp}), ...\}$ for each FU, where lower latencies reflect in more work per cycle, i.e. larger t_{cp}. If a high f_{op} is requested, the HLS compiler will allocate FUs with lower t_{cp} but consequently having a large latency. Thus f_{op} is not only related to the total execution time, but also to the total cycle count c and the peak performance is not always at the highest f_{op} or lowest c.

In Lin-Analyzer, the latencies of all FUs are constrained to a fixed frequency of 100MHz. In Lina, a hardware profile library was created containing the set of configurations for each FU. We collected such information from the detailed scheduling reports from Vivado HLS by varying the target f_{op} and analysing the reaction on l and t_{cp} for relevant FUs.

During operation scheduling, the RCLS uses an Resource Allocator (RA) to check if there are free resources to allocate an operation. If positive, Lina proceeds to TCS with the Timing Allocator (TA) which is performed in two parts: first, Lina selects the configuration for each FU that has the smallest l but without violating timing. Then, it performs timing-constrained scheduling where cycle merges are possible: if two consecutive dependent operations `a` and `b` (e.g. an arithmetic operation and its store) have critical paths t^a_{cp} and t^b_{cp} so that $t^a_{cp} + t^b_{cp} \leq \frac{1}{f_{op}}$, it is possible to schedule both at the same clock cycle. Fig. 2 presents both FUs adjustments and cycle merge.

B. Non-Perfect Loop Analysis

Lin-Analyzer estimates performance by scheduling the DDDG based on the sub-trace of the innermost loop body. The estimation will have deviations when the loop nest is non-perfect, worsening if the outer loop bounds are significantly larger than the innermost bound.

Lina implements Non-Perfect Loop Analysis (NPLA) by applying the same DDDG scheduling algorithm to DDDGs located between loop nests to refine the estimation. Considering a loop depth i, the cycle count for this loop body c_i is given as:

$$c_i = \frac{(c^{bf}_i + c^{af}_i + (c^{bw}_i \cdot (u_i - 1)) + (c_{i+1} \cdot u_i)) \cdot b_i}{u_i} + \sigma \quad (1)$$

where c^{bf}_i, c^{af}_i, c^{bw}_i are the cycle counts for the scheduled DDDGs before, after and between nested loops (when unroll is active), u_i is the unroll factor, b_i the loop bound and σ is the loop's enter/exit test latency.

Whenever possible, Vivado will merge the enter/exit tests to neighbouring operations. This means that σ can vary depending on the loop structure. In the non-perfect case, the values c^{bf}_i, c^{af}_i and c^{bw}_i are adjusted accordingly. For perfect loop nests, this can only occur when loop unrolling is enabled and thus the total loop cycle count c_i is calculated as:

$$c_i = (c_{i+1} \cdot b_i) + \sigma - \frac{(u_i - 1) \cdot b_i}{u_i} \quad (2)$$

III. Experimental Setup

To validate Lina, we used kernels that are provided together with Lin-Analyzer to test its accuracy, composed of 9 C kernels from the PolyBench benchmark [4]: `atax`, `bicg`, `conv2d`, `conv3d`, `gemm`, `gesummv`, `mvt`, `syr2k` and `syrk`.

We performed a small parameters exploration with loop unroll, pipelining, array partitioning and effective clock frequency (detailed information about the parameters can be found in the project repository). For each configuration, we measured the relative percentage error from Lina and Lin-Analyzer estimations against the values reported by Vivado HLS 2018.2 for the Xilinx UltraScale+ ZCU102 (the original Lin-Analyzer was adapted to support this board). All synthesis were performed on a system with Arch Linux, Intel i7-5500U CPU and 16GB of RAM.

IV. Experimental Results

Fig. 3 presents the relative error from Lina and Lin-Analyzer against the cycle counts reported by Vivado HLS for the `atax` and `conv2d` kernels. Due to page limitations and to avoid polluted plots, we show only the results for some kernels and

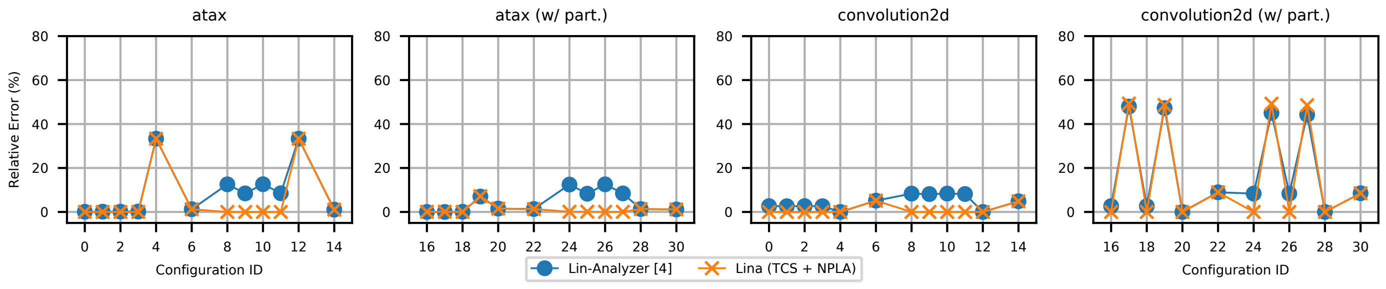

Fig. 3. Relative error between Lina and Lin-Analyzer against the cycle counts reported by Vivado HLS for different configurations of `atax` and `conv2d`.

for a partitioning factor of 2 (IDs 16-30). Other results can be found in the project repository.

Both estimators presented little or no error without partitioning and pipelining (ID < 4) and improved accuracy on `conv2d` and `conv3d`. With pipelining (IDs 4 and 6) both had comparable results, with some configurations with error of $\sim 20\%$. Until this point both estimators were configured to the same target frequency (100MHz with 27% uncertainty). For the IDs 8-14, a different frequency was used and Lina gave in many cases the exact cycle count, while Lin-Analyzer deviated from 10% to 30%. One source for the mentioned errors is due to internal optimisations performed by Vivado that both estimators cannot reproduce.

With array partitioning enabled (ID ≥ 16), Lina presented improved overall accuracy, but for some configurations both had notable deviations, with a peak value of $\sim 65\%$. One of the reasons is that Vivado conservatively assumes loop-carried dependencies that Lina, Lin-Analyzer and other estimators [3] do not. In some cases, Lin-Analyzer performed better than Lina. This happens when: both Lina and Lin-Analyzer are not able to accurately estimate the inner loop latency and; innacuracies in Lin-Analyzer involving σ (see end of Section II) and latency for `load` operation reduce the error (e.g. an estimation that is higher than the expected value is deducted by these innacuracies, inadvertedly reducing the error). We found these differences by comparing σ and `load` latency from Lin-Analyzer against the actual values from the Vivado scheduling reports.

In average for all kernels and tested configurations, the estimations from Lin-Analyzer had a relative error of 16.45%, while Lina 13.01%. Excluding array partitioning, the error for Lin-Analyzer dropped to 8.85% and Lina to 3.02%. Thus, in average, Lina presented better accuracy than Lin-Analyzer.

Fig. 4 presents the relative errors for `gemm` with a larger frequency for IDs 8-14 and 24-30. As expected, the increased frequency disparity from the fixed 100MHz of Lin-Analyzer yielded larger error for these configurations, since it does not perform any timing analysis.

Fig. 5 presents the cycle count for `bicg` and `conv2d` under different frequencies with optimisations disabled. It can be noted that Lina accurately reacted to the different timing constraints. Furthermore, the peak performance was not at the highest frequency nor the lowest cycle count: the optimal frequency was estimated at 285.71MHz and 153.85MHz for `bicg` and `conv2d` respectively.

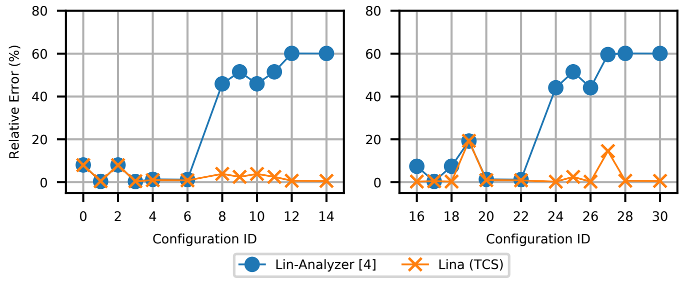

Fig. 4. Results for `gemm` with higher frequency for IDs 8-14 and 24-30.

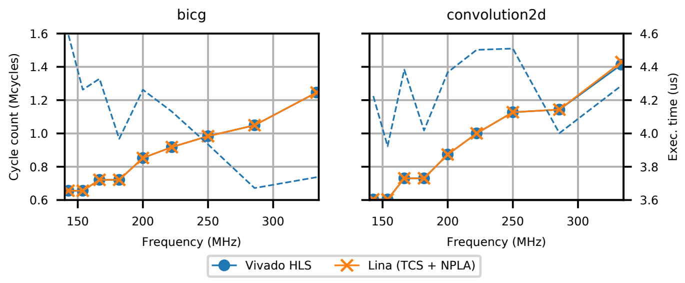

Fig. 5. Cycle count reported by Lina and Vivado HLS for `bicg` and `conv2d` in different frequencies (all optimisations disabled). The dashed line represents the total execution times considering frequency and cycle count. Both plots are scaled to the same intervals.

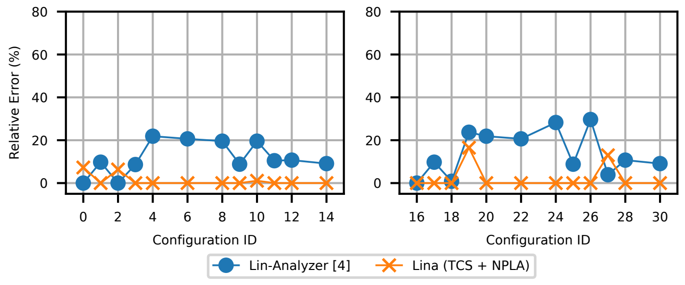

Fig. 6. Relative errors for a variant of `gemm` with different loop bounds.

Fig. 6 presents a variation of `gemm` where different loop bounds were used in a way that the proportion of executed operations between the outer loops and the innermost loop increased. As expected, the error difference between Lina and Lin-Analyzer was more noticeable, since Lin-Analyzer ignores the instructions between loop nests. For three configurations

Lina had worse accuracy, which was caused by a similar effect as previously explained: the innermost loop latency had a positive error for both estimators and NPLA only worsened the cycle count, which in turn increased the error for Lina.

Regarding the execution time of the estimations, the trace part was the most significant portion of the total time, accounting for 95% and 93% of the time of Lin-Analyzer and Lina respectively. Since it has to be performed only once, its impact decreases with the design space size. For the experiments in this paper, the total exploration time was 7.50 minutes with Lin-Analyzer and 7.67 with Lina, while Vivado HLS took 397.91 minutes.

V. Related Works

According to [5], FPGA HLS estimators for DSE are basically divided in two flavours: the ones that require HLS compilation and the ones that avoid it. Both avoid the whole synthesis process, which would lead to hours for a single design point. However, even the pre-synthesis HLS compilation itself might take several minutes, still hindering large DSE.

HLScope+ [6] uses code instrumentation and analytical modelling to improve the accuracy of the Vivado HLS simulator. It is two orders of magnitude faster than the whole synthesis process but since it requires HLS simulation for each design point, it is not well suited for very large DSE.

COMBA [3] has a similar methodology than Lina and Lin-Analyzer. It enables fast exploration by using a metric-guided DSE, discarding uninteresting points according to performance metrics. It also supports more exploration knobs, such as function pipelining, dataflow, inlining. The average errors for the PolyBench kernels are less than 2%.

For comparison purposes, we executed COMBA and Lina for `bicg` and `gemm` kernels with large and small loop bounds. For `bicg`, COMBA explored 1758 and 514 points for large and small variants respectively, leading to a per-point execution time of 0.40s and 0.46s against Lina's 0.12s and 0.05s including trace. For the `gemm` kernel, 1444 and 604 points were explored by COMBA for the large and small variants, leading to per-point 0.35s and 0.38s while Lina took 1.75s and 0.07s. The increased time for `gemm` large bounds with Lina is due to the dominating trace time, which would better dissolve with larger design spaces. Furthermore, the metric-guided DSE could be applied to Lina as well, reducing the amount of analysed points.

Even though COMBA also performs timing analysis and cycle merging, it has only 5 options of frequencies and the values used for the analysis were extracted through profiling, which is potentially less accurate than acquiring the values directly from the Vivado reports as we did. Furthermore, they analysed the merging possibilities in an instruction-by-instruction basis, which potentially exclude particular cases. We created a preliminary hardware profile for the board used in their paper and compared the cycle count from `bicg` against the actual values of Vivado for different frequencies as shown in Fig. 7. It can be noted that COMBA presents deviation from 150MHz onwards, while Lina keeps accurate.

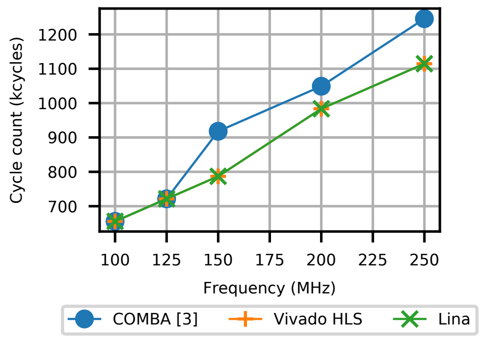

Fig. 7. Cycle count comparison between COMBA, Lina and Vivado HLS for the `bicg` kernel at different frequencies (all optimisations disabled), using a preliminary hardware profile library.

VI. Conclusion

This paper presented Lina, a performance estimator for C/C++ codes when used as input for Xilinx Vivado HLS. Lina is an improvement of Lin-Analyzer by including timing and non-perfect loop nest analyses. Results show that for different optimisation configurations for 9 kernels from PolyBench, Lina had an average relative error of 13.01% and 3.02% if array partitioning is not explored, an improvement over Lin-Analyzer's 16.45% and 8.85%. Lina is able to accurately predict different timing budgets, opening the opportunity for the user to find a good balance between operating frequency and total cycle count. Future works include attaching models for external memories and improvements in the loop scheduling.

Acknowledgment

The authors would like to thank FAPESP (São Paulo Research Foundation, grants no. 2018/22289-6 and 2016/18937-7) and Karlsruhe Institute of Technology for the financial supports given to this research project.

References

[1] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 35.

[2] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-Analyzer: A High-level Performance Analysis Tool for FPGA-based Accelerators," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 136.

[3] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, and B. He, "COMBA: A Comprehensive Model-Based Analysis Framework for High Level Synthesis of Real Applications," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 430–437.

[4] L.-N. Pouchet, "PolyBench: the polyhedral benchmark suite," 2012, available at https://www.cs.ucla.edu/pouchet/software/polybench, accessed 9th apr. 2019.

[5] K. O'Neal and P. Brisk, "Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 763–768.

[6] Y.-k. Choi, P. Zhang, P. Li, and J. Cong, "HLScope+: Fast and Accurate Performance Estimation for FPGA HLS," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 691–698.