Utilizing Parallel Workers: LLVM's Vectorization Plan

Jonas Fritsch
Technical University of Munich
jonas.fritsch@tum.de

Abstract

Lorem ipsum.

1 Introduction

1. Modern CPUs (SIMD-Registers, SIMD ISAs)
2. SIMD performance gains
3. Auto-Vectorization to help programmers utilize SIMD functionality out of the box (explicit vectorization as time consuming especially when supporting multiple architectures)
4. LLVM as popular, platform-agnostic compiler framework [8]
5. auto-vectorization complexity $\rightarrow$ wish for well designed, interoperable system (VPlan) $\rightarrow$ transition from Loop Vectorizer + SLP Vectorizer to Vectorization Plan $\rightarrow$ ongoing 8 year long effort started by Intel
6. rough paper outline

2 Background

1. Vectorization explanation (with Code-Examples)
   - Loop-Level Vectorization (Inner / Outer-Loops [6, 10, 18, 22, 23])
   - Function Vectorization [22]
   - Superword-Level Parallelism Vectorization [15] (maybe already [7])
2. Vectorization Constraints [2, 11] (reuse code-examples from above)
   - not always legal / applicable, e.g.: backwards data-dependencies, pointer aliasing, platform backwards-compatability / support, control flow complexity (function calls, register pressure), FP inaccuracies (`-ffast-math`)
   - possible performance loss, e.g.: costly conversions (horizontal aggregations, integer division) resulting in slower overall code on average
   - larger code size, e.g.: need for scalar loop-tail / epilogue
   - (potential security vulnerablities [13])

3 LLVM's Vectorization Plan

Overview [3, 4, 6, 21, 22, 25, 28, 29]

1. Phase 1: Legality
2. Phase 2: Planning (Vectorization Plans)
   a. Building initial VPlan based on Phase 1 Constraints (Generate one or more up-to-date Code Examples for VPlan similar to [4, 6, 21]) $\rightarrow$ Explain different components (e.g.: `VPBasicBlock`, `VPRegionBlock`, etc.) and correlation between generated VPlan structure, original scalar code. Show some general optimizations.
   b. Modelling Cost. Not yet part of VPlan. (related work? already adopted? [20])
   c. Optimizing/Transforming VPlans [3]
3. Phase 3: Execution - Materialize best VPlan

4 Related Work

1. Other auto-vectorization algorithms / possible improvements [2, 14, 17, 19, 24]
   Some of those should already be (partially) implemented in LLVM
2. Auto-Vectorization in GCC [12, 27]
3. Comparison with GCC [9, 16]
4. (Polyhedral compilation techniques [30])
5. (LLM-based Vectorization [26])

5 Summary and Future Work

1. VPlan Summary [3, 29]
2. VPlan Future Roadmap [3, 29] (e.g.: combining outer-/inner-loop paths, full function vectorization, etc.)
3. (Comparison/Outlook auto-vectorization and explicit vectorization [1, 5, 9])

References

[1] Neil Adit and Adrian Sampson. 2022. Performance Left on the Table: An Evaluation of Compiler Autovectorization for RISC-V. *IEEE Micro* 42, 5 (2022), 41–48. https://doi.org/10.1109/MM.2022.3184867

[2] Anupama Rasale Ashutosh Nema. 2023. Improving Vectorization for Loops with Control Flow. https://www.youtube.com/watch?v=mKG0NmGtpbE Accessed: 2024-11-04.

[3] Florian Hahn Ayal Zaks. 2023. VPlan: Status Update and Future Roadmap. https://www.youtube.com/watch?v=SzGP4PgMuLE Accessed: 2024-11-02.

[4] Gil Rapaport Ayal Zaks. 2017. Vectorizing Loops with VPlan – Current State and Next Steps. https://www.youtube.com/watch?v=BjBSJFzYDVk Accessed: 2024-11-04.

[5] Lawrence Benson, Richard Ebeling, and Tilmann Rabl. 2023. Evaluating SIMD Compiler-Intrinsics for Database Systems. In *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023 (CEUR Workshop Proceedings)*, Vol. 3462. CEUR-WS.org. https://ceur-ws.org/Vol-3462/ADMS5.pdf

[6] Diego Caballero and Vectorizer Team Intel Corporation. 2018. Extending LoopVectorize to Support Outer Loop Vectorization Using VPlan. https://www.youtube.com/watch?v=z6NeHLRNVok Accessed: 2024-11-04.

[7] Yishen Chen, Charith Mendis, and Saman Amarasinghe. 2022. All you need is superword-level parallelism: systematic control-flow vectorization with SLP. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 301–315. https://doi.org/10.1145/3519939.3523701

[8] Vikram Adve Chris Lattner. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization (CGO '04)*. IEEE Computer Society, USA, 75.

[9] Jing Ge Feng, Ye Ping He, Qiu Ming Tao, and Fazli Wahid. 2021. Evaluation of Compilers' Capability of Automatic Vectorization Based on Source Code Analysis. *Sci. Program.* 2021 (Nov. 2021), 15. https://doi.org/10.1155/2021/3264624

[10] Intel. 2019. Outer Loop Vectorization. https://www.intel.com/content/www/us/en/developer/articles/technical/outer-loop-vectorization.html Accessed: 2024-11-06.

[11] Intel. 2019. Vectorization Essentials. https://www.intel.com/content/www/us/en/developer/articles/technical/vectorization-essential.html Accessed: 2024-11-11.

[12] Jakub Jelínek. 2023. Vectorization optimization in GCC. https://developers.redhat.com/articles/2023/12/08/vectorization-optimization-gcc Accessed: 2024-11-04.

[13] Sayinath Karuppanan and Samira Mirbagher Ajorpaz. 2023. An Attack on The Speculative Vectorization: Leakage from Higher Dimensional Speculation. *arXiv e-prints*, Article arXiv:2302.01131 (Feb. 2023), 15 pages. https://doi.org/10.48550/arXiv.2302.01131 arXiv:cs.CR/2302.01131

[14] Universität-Saarland Compiler Design Lab. 2024. RV: A Unified Region Vectorizer for LLVM. https://github.com/cdl-saarland/rv Accessed: 2024-11-11.

[15] Samuel Larsen and Saman Amarasinghe. 2000. Exploiting superword level parallelism with multimedia instruction sets. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI '00)*. Association for Computing Machinery, New York, NY, USA, 145–156. https://doi.org/10.1145/349299.349320

[16] Klara Modin. 2024. A comparison of auto-vectorization performance between GCC and LLVM for the RISC-V vector extension. (10 2024). https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-354873

[17] Simon Moll, Shrey Sharma, Matthias Kurtenacker, and Sebastian Hack. 2019. Multi-dimensional Vectorization in LLVM. In *Proceedings of the 5th Workshop on Programming Models for SIMD/Vector Processing (WPMVP'19)*. Association for Computing Machinery, New York, NY, USA, Article 3, 8 pages. https://doi.org/10.1145/3303117.3306172

[18] Dorit Nuzman and Ayal Zaks. 2008. Outer-loop vectorization: revisited for short SIMD architectures. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08)*. Association for Computing Machinery, New York, NY, USA, 2–11. https://doi.org/10.1145/1454115.1454119

[19] Rouzbeh Paktinatkeleshteri, João P. L de Carvalho, Ehsan Amiri, and J. Nelson Amaral. 2023. Efficient Auto-Vectorization for Control-flow Dependent Loops through Data Permutation. In *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering (CASCON '23)*. IBM Corp., USA, 74–83.

[20] Angela Pohl, Biagio Cosenza, and Ben Juurlink. 2020. Vectorization cost modeling for NEON, AVX and SVE. *Performance Evaluation* 140-141 (2020), 102106. https://doi.org/10.1016/j.peva.2020.102106

[21] Gil Rapaport. 2017. Introducing VPlan to the Loop Vectorizer. https://www.youtube.com/watch?v=IqzJRs6tb7Y Accessed: 2024-11-06.

[22] Hideki Saito. 2016. Extending LoopVectorizer towards supporting OpenMP4.5 SIMD and outer loop auto-vectorization. https://www.youtube.com/watch?v=XXAvdUwO7kQ Accessed: 2024-11-06.

[23] Nadathur Satish, Changkyu Kim, Jatin Chhugani, Hideki Saito, Rakesh Krishnaiyer, Mikhail Smelyanskiy, Milind Girkar, and Pradeep Dubey. 2012. Can traditional programming bridge the Ninja performance gap for parallel computing applications?. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*. IEEE Computer Society, USA,

440–451.

[24] Jaewook Shin. 2007. Introducing Control Flow into Vectorized Code. In *16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007)*. 280–291. https://doi.org/10.1109/PACT.2007.4336219

[25] Sebastian Hack Simon Moll. 2017. VPlan + Rv: A Proposal. https://www.youtube.com/watch?v=svMEphbFukw Accessed: 2024-11-11.

[26] Jubi Taneja, Avery Laird, Cong Yan, Madan Musuvathi, and Shuvendu K. Lahiri. 2024. LLM-Vectorizer: LLM-based Verified Loop Vectorizer. arXiv:cs.SE/2406.04693 https://arxiv.org/abs/2406.04693

[27] GCC Team. 2023. Auto-vectorization in GCC. https://gcc.gnu.org/projects/tree-ssa/vectorization.html Accessed: 2024-11-04.

[28] LLVM Team. 2021. Auto-Vectorization in LLVM. https://llvm.org/docs/Vectorizers.html Accessed: 2024-11-04.

[29] LLVM Team. 2024. Vectorization Plan. https://llvm.org/docs/VectorizationPlan.html Accessed: 2024-11-04.

[30] Polly Team. 2017. Polly - LLVM Framework for High-Level Loop and Data-Locality Optimizations. https://polly.llvm.org/index.html Accessed: 2024-11-11.