SLP Vectorization in the Presence of Control Flow

Josh Clune and Thomas Talbot

Project Title: SLP Vectorization in the Presence of Control Flow

Group Info:

Thomas Talbot (ttalbot), ttalbot@andrew.cmu.edu
Josh Clune (jclune), jclune@andrew.cmu.edu

Project Webpage:

`https://ttalbot-cmu.github.io/`

Project Description:

Super-word level parallelism (SLP) is a technique for vectorizing code within a single basic block by replacing sets of independent, isomorphic instructions with their SIMD equivalent. In this project we will explore how SLP combined with loop unrolling can be used as a more flexible alternative to traditional loop vectorization. The goal is to vectorize instructions from unrolled iterations in the presence of control flow. To evaluate our project, we will measure the speedup of our test programs over LLVM's traditional loop vectorizer. As a stretch goal, we will apply loop unrolling and SLP vectorization after executing a loop fusion optimization.

75% goal: Unroll a loop and generate vector instructions with SLP.

100% goal: Unroll a loop and generate vector instructions with the SLP in the presence of arbitrary control flow.

125% goal: Fuse together multiple loops and then apply loop unrolling and SLP vectorization.

Logistics

Plan of Attack and Schedule:

There are approximately four weeks before the project's milestone deadline, and six weeks before the project's final deadline. To try to keep on schedule, we've set approximate weekly goals, though these are subject to change, particularly once we learn more about the capabilities of LLVM's native loop unrolling and SLP passes.

- Week 1: Familiarize ourselves with LLVM's Loop Vectorizer and SLP Vectorizer. In particular, it will be important to understand the Loop Vectorizer's cost model that determines the optimal vectorization and unrolling factor. This will have a potentially significant impact on the shape the rest of the project takes.
  - If the Loop Vectorizer's cost model seems to operate under the assumption/intention that the SLP Vectorizer will be called afterwards (meaning the cost model will attempt to optimize for, among other things, SLP vectorization opportunities), then we will likely put greater attention on dealing with other forms of control flow (e.g. an if-else-statement inside of a loop). This is because if optimizing for SLP Vectorization is something that LLVM's cost model already attempts to do, it is unlikely that a six week project will produce a cost model that compares favorably.
  - If the Loop Vectorizer's cost model does not seem to operate under the assumption/intention that the SLP Vectorizer will be called afterwards (or if the cost model appears to be agnostic to whether the SLP Vectorizer will be called later), then we may choose to devote some time attempting to modify the cost model to better optimize for SLP vectorization opportunities.

- Week 2: Write some basic code to use and test the outcome of naively using LLVM's Loop Vectorizer (specifically to perform loop unrolling) and subsequently using LLVM's SLP Vectorizer. This will serve both to establish a benchmark against which our code can be compared as well as to provide insight concerning potential areas of improvement. The circumstances in which LLVM's Loop Vectorizer and SLP Vectorizer perform well (or poorly) will help inform the direction of our efforts.
- Weeks 3 and 4: Attempt to improve upon the results seen from Week 2's testing. By the **milestone** deadline, our minimum target will be to have code that performs loop unrolling followed by SLP vectorization in some way that is less naive than simply calling LLVM's built-in passes.
- Weeks 5 and 6: Further weeks are more difficult to forecast. Depending on the results of weeks 1 and 2, we may or may not have broached issues of control flow within loops by this point. If we haven't then we will aim to do so in the last few weeks to try to support SLP vectorization across at least some simple control flow. If we have, we will explore more ambitious stretch goals (e.g. first performing loop fusion).

Literature Search:

This project was inspired by the framework in [1] which extended SLP to vectorize across different basic blocks and loop nests. The earlier work in [2] can help with understanding the fundamentals of SLP vectorization and loop unrolling. The loop fusion optimization is presented in [3]. More exploration is needed to find previous studies that implemented loop unrolling with SLP Vectorization in LLVM. However, the LLVM documentation on loop auto-vectorization and unrolling is a sufficient starting point.

Resources Needed:

Our project will be implemented within the LLVM-10 framework used for the previous course assignments. We plan on using the existing LLVM passes for SLP vectorization and loop unrolling to start our analysis. We do not anticipate needing additional hardware resources. To evaluate our study, we plan on using the LLVM exegesis benchmarking tool to characterize the speedup of our transformations on a hand-made set of test programs. Additional evaluation can be done using open-source benchmarks such as PolyBench. PolyBench, a commonly used benchmark for loop-nest optimizations has data-access patterns that could benefit from unrolling plus SLP vectorization [4].

Getting Started:

We have read (to varying degrees of closeness) the works included in the references section. In particular, we have discussed the approach described in [1] and determined that although implementing much of their approach would be outside of the scope of a six week project, we may borrow some ideas about control predicates. There are not any current questions/constraints preventing us from getting started immediately (besides the exploratory questions we intend to resolve in the first week or two).

References

[1] Y. Chen, C. Mendis, and S. Amarasinghe, "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*, ser. PLDI

2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 301–315. [Online]. Available: https://doi.org/10.1145/3519939.3523701

[2] S. Larsen and S. Amarasinghe, "Exploiting superword level parallelism with multimedia instruction sets," *SIGPLAN Not.*, vol. 35, no. 5, p. 145–156, may 2000. [Online]. Available: https://doi.org/10.1145/358438.349320

[3] K. Kennedy and K. S. McKinley, "Maximizing loop parallelism and improving data locality via loop fusion and distribution," in *International Workshop on Languages and Compilers for Parallel Computing*, 1993.

[4] L.-N. Pouchet, "Polybench/c the polyhedral benchmark suite," 2021. [Online]. Available: https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/