Milestone Report: 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/`

Major Changes:

There have been no major changes since the project proposal.

Accomplishments:

We have configurable optimization passes that perform loop unrolling and SLP vectorization using the internal methods in LLVM. The passes are designed to work one after the other; loop unrolling then SLP vectorization. The passes working in combination can expose more parallelism than SLP vectorization alone. Results of the combined loop unroll + SLP Vectorization are included on the project web page linked above.

The first example, (test1.c) performs loop unrolling by a factor of three before SLP vectorization. After unrolling, two stores to adjacent elements of an array are contained in the the same basic block. The SLP vectorizer groups the contiguous stores and generates a vector store instruction with a SIMD factor of two. The transformations applied to the LLVM bitcode are shown in test1-m2r.ll (the intial IR generate at -o0), test1-unroll.ll (the IR after unrolling), and test1-slp.ll (the final IR with vector instrinsics).

The second example, (test2.c) performs loop unrolling by a factor of four before SLP vectorization. After unrolling, the multiplication, addition, and store operations for separate loop iterations are in different basic blocks. The SLP vectorizer generated vectorized add, multiplication, and store operations with a SIMD factor of two in each of the unrolled iterations. The transformations applied to the LLVM bitcode are shown in test2-m2r.ll (the intial IR generate at -o0), test2-unroll.ll (the IR after unrolling), tes2t-slp.ll (the final IR with vector instrinsics).

The accomplishments so far align with our expectations about how loop unrolling and SLP vectorization would interact. Namely, loop unrolling can create more opportunities for super word-level parallelism when multiple unrolled iterations are contained within the same basic block. This also confirms our presumption that SLP will not vectorize operations across basic blocks.

Meeting the Milestone:

We are on track with our milestone goal. We were able to successfully generate vector instructions using loop unrolling and SLP vectorization. This brings us to our 75% goal for the project overall.

Surprises:

We have not encountered significant surprises since the time of the proposal. After performing loop unrolling, we had to use LLVM's simplify CFG pass to enable the SLP vectorizer. This was an unexpected result since we assumed that the SLP vectorizer would be able to directly optimize the output of the unroll pass. We were surprised by how long it took to enable SLP's vectorization pass. Enabling the pass required modifying the default SLP cost threshold and minimum tree size.

Revised Schedule:

We are mostly on schedule with the outline in the project proposal. Familiarizing ourselves with LLVM's loop vectorizer is the first priority. Next, we plan to evaluate the performance of our unroll + SLP vectorizer versus the loop vectorizer. As a stretch goal, we will insert a loop fusion pass before the unroll + SLP vectorizer to extract opportunities for vectorization across adjacent loops.

- Compare the results of LLVM's loop vectorizer with our unroll + SLP vectorizer pass on hand-made programs to see where performance might differ. Specifically, we hope to see our approach find vectorization opportunities in outer loops that LLVM's loop vectorizer would disregard. LLVM's loop vectorizer does not fully support outer loop vectorization.
- Compare results of the LLVM's loop vectorizer with our unroll + SLP vectorizer on the PolyBench test suite [1].
- Depending on feedback from the professor, the remainder of our time could be devoted to evaluating our approach's performance in contrast to LLVM's loop vectorizer (in particular, as it relates to outer loop vectorization), or we could attempt to support some basic reasoning about limited control flow within a loop body.

Resources Needed:

We have all the resources we need. To evaluate our optimizations, we plan on using LLVM's machine code analyzer for simulating the instructions per cycle and resource pressure on an x86 CPU.

References

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