Milestone Report: Straight Line Vectorization on PL/PGSQL User-Defined Functions in a Compilation-Based In-Memory Database

Prithvi Gudapati (pgudapat), Tanuj Nayak (tnayak)

`https://prithg98.github.io/745Website/`

1 Major Changes

We initially were planning on implementing an auto-vectorization pass inspired by Porpodas et al. [1]'s work on super-node superword level parallelism. However, we are currently actually implementing an auto-vectorization pass inspired by Larsen et al. [2]'s work on superword level parallelism. We intend to especially focus on automatically vectorizing arithmetic operations in UDF code across columns of a given row.

2 What We Have Accomplished So Far

2.1 UDF Compiler

We have implemented the foundations for a compiler that converts from PL/PGSQL text programs to bytecode for our domain specific language (TPL) and, consequentially, LLVM IR. For now it compiles all non-SQL constructs such as if statements, loops, functions and parameters.

2.2 SLP LLVM Pass

We are working on the implementation of the SLP LLVM pass. As of Wednesday 10/15, we are able to successfully gather and group the seed instructions, which are stores to continuous memory addresses. In addition, we are able to trace the use-def chains for each argument being stored. Using this information, we are also able to construct the vectorization trees. We have not yet implemented the scheduling and placement of the vectorized instructions based on the vectorization trees.

3 Meeting Our Milestone

Because we have changed our implementation slightly, we are a little behind where we originally wanted to be. As a result, we have not quite met our milestone. However, we are optimistic that we should still be able to accomplish the 100% goals that we stated in the proposal.

4 Surprises

Implementing the UDF compiler to work with the domain specific language for the Terrier database has not been as easy as we thought it would be. We had to augment features of the domain specific language here and there in order to fit our needs. This difficulty is especially augmented by the fact that we are inserting code into a large preexisting codebase.

Furthermore, implementing the auto-vectorization pass has been similarly challenging. However, we now have a good understanding of what exactly we need to do and how to go about doing it. In addition, after some initial issues, we have made decent progress on this front.

5 Revised Schedule

1. Week of 4/13:
   (a) Tanuj: Get embedded static SQL calls to work inside queries.
   (b) Prithvi: Implement the scheduling and insertion of vectorized instruction.
2. Week of 4/20:
   (a) Tanuj and Prithvi: Finish the SLP pass.
   (b) Tanuj and Prithvi: Run the SLP pass on the benchmarks.
   (c) Tanuj and Prithvi: Begin working on the report.
3. Week of 4/27:
   (a) Tanuj and Prithvi: Finish working on the report.
   (b) Tanuj and Prithvi: Work on the poster.

6 Resources Needed

As mentioned in the proposal, we need a good machine with vectorization hardware capabilities to develop and benchmark our passes on. At the moment, we have access to one such machine from the CMU DB group.

References

[1] Vasileios Porpodas, Rodrigo C. O. Rocha, Evgueni Brevnov, Luís F. W. Góes, and Timothy Mattson. Super-node slp: Optimized vectorization for code sequences containing operators and their inverse elements. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, CGO 2019, page 206–216. IEEE Press, 2019.

[2] Samuel Larsen and Saman Amarasinghe. Exploiting superword level parallelism with multimedia instruction sets. *SIGPLAN Not.*, 35(5):145–156, May 2000.