Proposal: 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 Project Description

NoisePage is an in-memory database by the CMU DB group. It executes optimized SQL query plans by compiling them to a domain specific language called TPL. These compiled TPL scripts are compiled to LLVM at which layer optimization can be done. Right now, there is only infrastructure to compile and execute SQL queries. This project aims to add UDF execution capabilities in PL/PGSQL by converting these scripts the same way to TPL and then LLVM. Along with that, we aim to add auto-vectorization passes along the lines of SLP/SN-SLP and other variants of superword level parallelism passes to the compiled procedures and observe the performance gains from doing so.
Our goals for this project are as follows.

100%
Implement a PL/PGSQL compiler with full capabilities of processing embedded static SQL queries. Auto-vectorization SLP and SN-SLP passes that we wrote should be integrated with this compiler as well.

75%
Implement a PL/PGSQL compiler with full capabilities of processing embedded static SQL queries. Auto-vectorization SLP and SN-SLP passes that we wrote should not necessarily have to be integrated with this compiler as well.

125%
Implement a PL/PGSQL compiler with full capabilities of processing embedded static SQL queries and dynamic SQL queries. Auto-vectorization SLP and SN-SLP passes that we wrote should be integrated with this compiler as well. We also make additional adjustments and improvement to the existing SN-SLP pass in order to acheive better performance on our benchmarks.

2 Plan of Attack and Schedule

1. 3/16
   (a) Tanuj: Finish UDF compilation or some prototype.
   (b) Prithvi: Begin implementing an SN-SLP LLVM pass on some test C-code in the meantime.
2. 3/23
   (a) Tanuj: Debug any issues with the UDF compilation and aim to have the benchmarks that can be compiled.
   (b) Tanuj and Prithvi: Work on SN-SLP LLVM pass implementation.
3. 3/30:
   (a) Tanuj and Prithvi: Continue working on the SN-SLP LLVM pass implementation.

4. 4/6:
    (a) Tanuj and Prithvi: Get the performance of SN-SLP on the benchmarks and look closely at the benchmarks to see what additional optimization could further improve performance.
5. 4/13:
    (a) Tanuj and Prithvi: Finish implementing the new optimizations
6. 4/20: Finish impl the new optimizations.
    (a) Tanuj and Prithvi: Test and iterate on the new optimizations if more performance gains can be realized.
    (b) Tanuj and Prithvi: Finish the project report.
7. 4/27:
    (a) Tanuj and Prithvi: Work on the poster.

3 Milestone

By April 16th, we should have implemented our SN-SLP auto-vectorization pass, run it on our benchmarks, and have a detailed performance analysis. In addition, we should have an idea of what additional improvements and optimizations we can add to our auto-vectorization pass.

4 Literature Search

Porpodas et al. [1]'s work on super-node superword level parallelism will guide our implementation of a straight line auto-vectorization pass. This work involved using the commutativity of certain operations and their relationship with their inverse operation to identify additional subgraph isomorphisms that would be useful for auto-vectorization. Another literature review may be conducted after implementing SN-SLP in order to identify existing techniques that can potentially improve the performance of the auto-vectorization pass on the benchmarks.

5 Resources Needed

We would need a good machine with vectorization hardware capabilities to develop and benchmark our passes on. We will get access to one such machine from CMU DB group.

6 Getting Started

We have started working on the UDF compiler and are close to finishing an untested prototype. We don't have any questions or constraints as of now.

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.