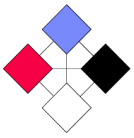
D3.2 Compiler Prototype



DAPHNE

Integrated Data Analysis Pipelines for Large-Scale Data Management, HPC, and Machine Learning

Version 1.1

PUBLIC

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 957407.

Document Description

Previous deliverables already shared the overall system architecture [D2.1] as well as the initial language and compiler designs [D3.1, D+22]. Since February 2021, a prototype of the DAPHNE infrastructure is under continuous development, which implements central components of the initial language abstractions, optimizing compiler infrastructure, local and distributed runtime backends, as well as hardware accelerator integration. We base this prototype on MLIR (multi-level intermediate representation) [LA+21] as a library of compiler infrastructure in order to facilitate a cost-effective development of our domain-specific language, reuse of compiler infrastructure, and good extensibility. This document shares a snapshot of this prototype, and describes an example scenario of running regression (and similarly classification, clustering, dimensionality reduction, and graph processing) algorithms on real datasets. The initial open source release of the DAPHNE prototype system is planned for end of March 2022.

D3.2 Compiler Prototype			
WP3 – DSL Abstractions and Compilation			
Type of document	D	Version	1.1
Dissemination level	PU		
Lead partner	ETH		
Author(s)	Matthias Boehm (KNOW), Ce Zhang (ETH), Patrick Damme (KNOW), DAPHNE Development Team		
Reviewer(s)	Andreas Laber (IFAT), Ahmed Eleliemy (UNIBAS), Patrick Damme (KNOW)		

Revision History

Version	Revisions and Comments	Author / Reviewer
V1.0	Initial structure and write-up	Matthias Boehm
V1.1	Incorporated reviewer comments, new artifact	Matthias Boehm

1 Artifact Access

The compiler prototype is publicly accessible as a password-protected snapshot of the DAPHNE development repository (created February 28, 9.30pm) under the following link

Link: https://daphne-eu.know-center.at/index.php/s/krFMFBre6swM9sa (145MB)

2 Demonstration Scenario

Step 1 Install Dependencies: Setup a Linux environment (tested with Ubuntu 20.04), and install the dependency versions specified in docs/GettingStarted.md, which includes clang, cmake, git, lld, ninja, pkg-config, python3, openjdk, gfortran, and uuid-dev. For tests related to DAPHNE's Python API, please further install numpy in your Python environment.

Step 2 Download and Extract: Download the artifact from the link in Section 1, and extract it as follows into a directory called daphne:

```
tar -xzf daphne.tar.gz;
cd daphne;
```

Step 3 Build DAPHNE: Then build the prototype, its dependencies, and the parser as follows from within the daphne directory (if it fails run "`./build.sh --clean`" for a clean start). On the first run this step might take 10 to 30min.

```
./build.sh
```

Step 4 Download the Data: Create a data directory and download the UCI wine dataset (as a regression problem for predicting wine quality) as follows:

```
mkdir data;
curl https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv -o data/wine.csv;
sed -i '1d' data/wine.csv;
sed -i 's/;/,/g' data/wine.csv;
echo '4898,12,1,f64,nnz=58776' > data/wine.csv.meta;
```

Step 5 Regression Example: Run the direct-solve linear regression script via the following command in order to train a model and summarize its prediction quality:

```
time build/bin/daphne scripts/lmDS.daph \
  XY=\"data/wine.csv\" icpt=1 reg=0.001 verbose=1
```

Step 6 Explain DaphneIR: Run the same scenario with additional explain flag in order to investigate the generated DaphneIR plans with and without vectorized (tiled) execution:

```
time build/bin/daphne --explain-kernels scripts/lmDS.daph \
  XY=\"data/wine.csv\" icpt=1 reg=0.001 verbose=1

time build/bin/daphne --vec --explain-kernels scripts/lmDS.daph \
  XY=\"data/wine.csv\" icpt=1 reg=0.001 verbose=1
```

The slightly different results for "AVG_Residuals_Yhat" (3.39898e-14 vs -5.18709e-14) originate from multi-threaded operations (with vectorization) and related round-off errors.

3 Prototype Structure

As an additional step, investigate the source code of the compiler and runtime prototype. In detail, the prototype repository is organized as follows:

- **`build`** (compiled system, and parser; generated via build.sh)
- **`doc`** (basic setup and developer documentation)
- **`scripts`** (algorithm level scripts as examples)
- **`src`** (main source code repository)
  - **`api`** (cli including daphne which orchestrates the remaining components)
  - **`compiler`** (execution, explain, inference, lowering)
  - **`ir`** (DaphneIR including the DAPHNE MLIR dialect)
  - **`parser`** (DaphneDSL, SQL)
  - **`runtime`** (distributed, local including data, I/O, kernels, and vectorization)
  - **`util`**
- **`test`** (test suite of component and integration tests, organized by components)
- **`thirdparty`** (dependencies such as llvm, including their build directories)

Finally, the DAPHNE test suite can be run via ./test.sh from within the daphne directory.

4 References

[D2.1] DAPHNE: D2.1 Initial System Architecture, EU Project Deliverable, 08/2021

[D3.1] DAPHNE: D3.1 Language Design Specification, EU Project Deliverable, 11/2021

[D+22] Patrick Damme et al.: DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines, CIDR 2022

[LA+21] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques A. Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, Oleksandr Zinenko: MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. CGO 2021