AMLS Programming Projects (SS 2021)

– DAPHNE projects –

#D1 Parser for SystemDS DSL → DaphneIR

Descriptive Machine Learning (DML), the R-inspired input language to SystemDS, is already well-established for writing ML algorithms. Supporting it as a front-end to the DAPHNE system will allow data scientists to ramp up quickly and re-use their existing DML scripts with a novel back-end.

The task is to implement (C++) a parser for DML as a part of the DAPHNE system. Basing the parser on the ANTLR parser generator allows to re-use the existing grammar definitions from the SystemDS code base. Thus, the crucial part is to re-wire DML syntax elements to the operations in DaphneIR, the MLIR/LLVM-based intermediate representation used in the DAPHNE system.

#D2 Parser for subset of SQL → DaphneIR

To avoid expensive border-crossing between different systems, the DAPHNE system will support linear algebra and relational algebra alike. SQL is the state-of-the-art language for expressing database queries as relational algebra programs. Thus, the DAPHNE systems needs to support SQL as well.

The project is to develop (C++) a parser for a manageable subset of SQL. The input will be an SQL query consisting of well-known clauses such as SELECT, JOIN, WHERE, GROUP BY, HAVING, and ORDER BY. Furthermore, rich expressions on attributes of relations, such as calculations, comparisons, aggregations, as well as built-in and user-defined functions shall be supported. The output is a representation of the query in DaphneIR, the MLIR/LLVM-based intermediate representation used in the DAPHNE system. Basing the implementation on the ANTLR parser generator would be very welcome, since this is already in use in the DAPHNE system.

#D3 Explain: readable IR via custom IR-level parser/printers

While the DAPHNE compiler operates on an in-memory representation of the MLIR/LLVM-based DaphneIR, a human-readable textual representation is of crucial importance for debugging purposes and for understanding the optimizer’s decisions.

The task is twofold. On the one hand, a concise notation for the DaphneIR operations shall be designed. To this end, MLIR offers an expressive yet simple pattern-based language to declaratively specify custom printers and parsers for IR operations. On the other hand, these should be complemented by a C++ implementation of facilities to better structure the textual intermediate representation of complex linear and relational algebra programs, which might consist of thousands of operators. One idea would be to introduce different verbosity levels to (not) show the contents of blocks, or additional information on interesting properties of matrices and frames, to name just a few examples.

#D4 Sparsity-aware MM chain optimization w/ rewrites

Matrix multiplications are at the core of many ML algorithms and often appear in chains. The order in which the individual multiplications are executed has a significant impact on the intermediates' physical sizes and, thus, the performance. At the same time, (ultra)sparse matrices are commonplace in real-world scenarios. As the sparsity determines the physical size of a sparse matrix it should be taken into account during matrix multiplication chain optimization.

This project is about implementing (C++) one or two sparsity-aware matrix-multiplication chain optimization techniques from the literature, e.g. MNC Sketch and/or others presented the lecture. This includes the estimation of the sparsity of the intermediate results and the propagation of the required sketches as well as the application of rewrites to optimize the chain. The implementation will extend the MLIR/LLVM-based compiler of the DAPHNE system.

#D5 Various LA and RA simplification rewrites

The initial program plans after parsing are usually inefficient and need to undergo a sequence of optimization passes. In this context simplification rewrites at the algebraic level play a crucial role.

The task is to augment the DAPHNE compiler with C++ implementations of an appropriate number of simplification rewrites working on the MLIR/LLVM-based DaphneIR. Rewrites can address linear algebra operations, relational algebra operations, or the interplay of both. Some rewrites are beneficial in all cases, others might depend on cost estimates. Some rewrites are applicable irrespective of the concrete data, others depend on certain interesting properties (e.g. symmetry).

#D6 IO readers/writers for common data formats (arrow, parquet)

Over the past years, a number of formats for the persistent storage of large data sets have been established. Two famous and widely used examples are Arrow and Parquet.

This project is about implementing (C++) efficient readers and writers for Arrow and/or Parquet that are able to obtain the in-memory representation of a dense/sparse matrix or a frame from such a file, or to save it to a file.

#D7 Matrix and frame data generators (dense and sparse, properties)

Interesting properties of matrices and frames can be exploited to improve performance by means of special rewrites in the compiler and special algorithms at run-time. When micro-benchmarking such techniques, fine-grained control over the properties of a matrix/frame is invaluable.

The task is to implement (C++) a number of expressive and efficient matrix and/or frame data generators. These shall take as input the size and certain interesting properties and produce a random matrix/frame matching these specifications. Possible properties include sparsity, symmetry, #distinct values, value distribution, sort order, and column correlations. There could be individual data generators for dense matrices, sparse matrices, and frames. Furthermore, different value types could be supported, such as floats, integers, strings/categorical, boolean, and complex numbers.

#D8 Kernels for LA and RA operations (dense and sparse)

After numerous optimization passes on the intermediate representation of the program, the low-level operators selected by the optimizer are executed by invoking pre-compiled kernels. Thus, both the feature set and performance of the DAPHNE system depends on these kernels.

The task is to implement (C++) kernels for an appropriate number of DaphneIR operations. Examples include, but are not limited to, matrix multiplication, elementwise operations, multiple forms of aggregation, reorganization, matrix decompositions, DNN operations, set operations, selection, joins, grouping, sorting, etc. The interfaces of the kernels in terms of inputs and outputs will be pre-defined. The kernel implementations could target, e.g., different data type implementations (e.g. dense or sparse matrices), different value types, and optimizations for interesting properties (e.g. symmetry). While inspiration can be taken from the SystemDS implementations, there is enough room for fresh idea. For instance, SIMD instruction set extensions may (optionally) be employed to speed up the processing.

#D9 Distributed runtime operations on Spark

When processing large-scale data sets, the inputs and outputs of an operation often do not fit into the memory of a single machine. In these cases, systems for ML usually resolve to a distributed processing on several nodes, whereby established data parallel frameworks such as Spark solve many of the involved challenges.

This project is about integrating Spark and the DAPHNE system in order to enable a cost effective parallelization of the workload. The integration could be approached from two perspectives: On the one hand, multiple instances of the DAPHNE system could be spawned and orchestrated by a Spark instance. On the other hand, an instance of the DAPHNE system could invoke Spark to distribute individual operations in a hybrid runtime plan.

#D10 Analyze: Extraction of data characteristics (interesting properties)

Interesting properties of matrices and frames can be exploited to improve performance by means of special rewrites in the compiler and special algorithms at run-time. Since the information on these properties is usually limited at compile-time, an analysis of the actual data at run-time can be very helpful.

The goal of this project is to implement (C++) the means to analyze interesting properties (e.g. sparsity, symmetry, #distinct values, value distribution, sort order, and column correlations) of matrices and/or frames. Depending on the data type implementation (e.g. dense or sparse), individual optimizations of the analysis algorithm are possible. Furthermore, the performance/accuracy trade-off could be opened up by investigating only a subset of the data by means of sampling techniques.