

# Enabling Integrated Data Analysis Pipelines on Heterogeneous Hardware through Holistic Extensibility

Extended Abstract (New Idea)

Patrick Damme<sup>1</sup> Matthias Boehm<sup>2</sup>

## 1 Introduction

Integrated data analysis (IDA) pipelines, that combine data management/query processing, high-performance computing, and machine learning training/scoring, become increasingly common in practice. Systems of these areas share many compilation and runtime techniques, and stress every hardware aspect of storage, computation, and networking. Accordingly, these systems are strongly impacted by hardware challenges such as the end of Dennard scaling and the end of Moore's law, which ultimately lead to dark silicon and increasing specialization at device level (CPUs, GPUs, FPGAs, ASICs), storage level (computational memory/storage, storage hierarchies), and workload level (data types and sparsity).

While this makes research on novel and heterogeneous hardware more exciting than ever, researchers are increasingly confronted with the question of how to integrate their prototypes to evaluate their impact on end-to-end IDA pipelines. Building yet another dedicated system offers a lot of flexibility, but requires substantial infrastructure efforts. However, enhancing an established system requires deep knowledge of the system internals and can be very hard. Thus, already in the 1980/90s, there was a wave of research on *extensible* DBMSs [CH90]. One of the most famous systems developed at that time is Postgres, which allows adding user-defined data types, functions, and access methods [SAH87]. Since then, concepts for *extensibility* and *variability* have been proposed for various system components, at different abstraction levels, and in different kinds of data systems. Recently, extensibility has also gained traction in the context of component-based systems [HD23]. However, to the best of our knowledge, there is no system infrastructure that *holistically* supports user extensions for all components relevant to the efficient execution of IDA pipelines on today's heterogeneous compute/storage hardware. To overcome this problem, we propose *holistic extensibility*.

In this talk, we present the concept of holistic extensibility for IDA pipelines, sketch how we approach this concept in DAPHNE, and provide an overview of our ongoing work.

---

<sup>1</sup> Technische Universität Berlin, Germany, patrick.damme@tu-berlin.de

<sup>2</sup> Technische Universität Berlin, Germany, matthias.boehm@tu-berlin.de

## 2 Holistic Extensibility for IDA Pipelines

*Holistic* extensibility means that every aspect of a data processing system for IDA pipelines should be easily extensible by users without a deep understanding of the system internals. This concept can be seen as an *ideal*, since it is hard to define a provably complete set of aspects requiring extensibility, and the need can evolve over time (e.g., integrating heterogeneous hardware was not a focus in the 1980s). However, we identify the following *extensibility aspects* relevant to the integration of novel computing or storage hardware:

**Operators.** Supporting different hardware accelerators typically requires dedicated operator code for each device. For instance, a CPU operator may be written in C++ employing SIMD intrinsics while a GPU operator may be written in CUDA. An extensible system should enable the integration of different physical operators, targeting different devices, for the same logical operator. Moreover, it should allow the definition of new (e.g., composite) operators in cases where this facilitates the execution on a particular device.

**Data Representation.** Operators targeted at specific hardware often *require* or *enable* specific data representations in terms of the overall storage layout (data types) and individual values (value types). E.g., in linear algebra for ML and simulations, different dense and sparse matrix data types were proposed. Furthermore, new value types for certain accelerators are emerging, e.g., `tf32` (GPUs) or `bf16` (TPUs). Moreover, specialized hardware often addresses specific applications, requiring the extension by domain-specific data representations.

**Optimization & Scheduling.** To make effective use of extensions, the system's optimizer must be able to reason about them, which requires an extensible internal representation (IR). The crucial decisions include when to use which custom operator and data representation, and how to place operators and data on the available (heterogeneous) computation and storage devices. For this purpose, it must be possible to add specific optimization passes to statically decide based on inferred data properties and system architecture, as well as to add specific runtime schedulers to make dynamic decisions which take the current execution behavior and system load into account. Both of these can benefit from custom cost models.

Typically, all of these extensibility aspects need to interact to fully integrate a novel hardware device. However, a *low barrier of entry* is crucial to achieve adoption and to facilitate *exploratory specialization*. For instance, it should be possible to add a physical operator for an accelerator without building an entire new processing engine. Moreover, existing operator implementations should be reusable for a custom data representation with acceptable out-of-the-box performance, to allow the user to focus on specializing and optimizing heavy-hitter operators for this representation. Finally, a deep integration into the optimizer should be *optional* by supporting *hints* on which physical variant and accelerator to use for an operator and which representation and storage device to use for an intermediate result.

The crucial aspects of holistic extensibility include: (1) how to balance expressiveness and additional complexity of the extensible system, and (2) how to achieve superb performance underneath the newly introduced abstractions.

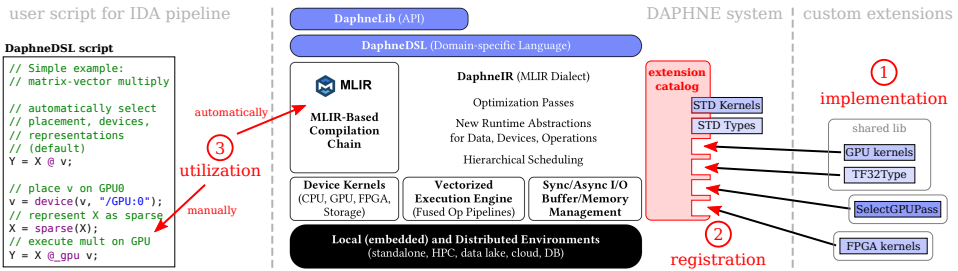


Fig. 1: DAPHNE System Architecture with Three-step Extension Approach.

### 3 Towards Holistic Extensibility in DAPHNE

DAPHNE<sup>3</sup> [Da22] is an open and extensible system infrastructure for IDA pipelines, including language abstractions, compilation and runtime techniques, multi-level scheduling, heterogeneous hardware accelerators, and computational storage for increasing productivity and eliminating unnecessary overheads. IDA pipelines are expressed in DaphneDSL, a domain-specific language for linear algebra and extended relational algebra over matrices and frames. DaphneDSL is parsed into DaphneIR. In an MLIR-based [La21] compilation chain, DaphneIR is optimized by domain-specific and traditional programming language optimizations, lowered to LLVM with calls to *pre-compiled* operator kernels, JIT compiled, and executed in a local or distributed runtime. DAPHNE’s *vectorized engine* fuses pipelines of operators, serves as the central means for parallelism, and is the central component for simultaneously utilizing heterogeneous hardware such as GPUs, FPGAs, and computational storage. Next, we give an overview of our extensibility design and mention some interesting research questions. To extend DAPHNE, users follow a three-step approach (Figure 1).

**1. Implementation.** The user implements the custom extensions for kernels, data/value types, optimizer passes, or scheduling techniques outside the DAPHNE code base in C++ adhering to well-defined *extension hooks*, and compiles them as a shared library. This does not require a deep understanding of the DAPHNE code base. Research questions include defining the right interfaces to balance expressiveness and complexity, achieving efficiency underneath these abstractions, and combining existing kernels and new data/value types.


**2. Registration.** The user registers the extension in DAPHNE’s *extension catalog* either through configuration files or from DaphneDSL. This requires providing the name and shared library as well as information specific to kernels (e.g., DaphneIR operation, expected input/output data/value types, required interesting data properties), data types (logical data type, preferred slicing axis for partitioning), and value types (bit width, semantics). More information can *optionally* be provided, e.g., traits and cost models to be used by the DAPHNE compiler. As the extension catalog can get large, its internal structure is decisive to efficiently serve relevant access patterns like look-up by operation and hardware device.

<sup>3</sup> <https://github.com/daphne-eu/daphne>

**3. Utilization.** By default, DAPHNE makes all decisions like the selection of kernels and physical data types as well as placement *automatically*, to increase users' productivity. To support this behavior for custom extensions, one option is to provide traits and cost models (e.g., for operator execution times or physical data size) in the extension catalog, which can be used by built-in optimization passes. Another option is to add a new optimization pass employing the extension where beneficial, which is simplified in DAPHNE due to the modular nature of the optimizer pipeline in MLIR. Even entire third-party MLIR dialects could be added, including operations, traits, and transforms. In fact, this is a promising option for generating code for hardware accelerators. This approach allows integrating new accelerators through dedicated dialects. Interesting questions include suitable abstractions for cost models and the integration of custom traits and interesting properties into the existing optimizer. To facilitate experimentation, DAPHNE also supports *manual* decisions. In DaphneDSL, users can provide *hints* on which device, kernel, or physical data representation to use for an operation or intermediate. These hints are treated as constraints by the optimizer. Interesting questions include the propagation of hints through the IR.

## 4 Conclusions and Outlook

We proposed *holistic extensibility* for IDA pipelines to handle increasing specialization from operators for heterogeneous hardware over the often co-designed data representations to the corresponding optimization and scheduling techniques. We sketched the extensibility design of DAPHNE, which offers users great benefits, while requiring low effort. We are currently implementing this design with a focus on kernels and data/value types, including some useful example extensions to showcase its simplicity. Our vision is to enable researchers to easily integrate their prototypes into a full-fledged system for IDA pipelines with minimal effort, thereby simplifying experimentation with and sharing of their work.

**Acknowledgments.**  The DAPHNE project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 957407.

## Bibliography

- [CH90] Carey, Michael J.; Haas, Laura M.: Extensible Database Management Systems. SIGMOD Rec., 19(4):54–60, 1990.
- [Da22] Damme, Patrick et al.: DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. In: CIDR. 2022.
- [HD23] Haffner, Immanuel; Dittrich, Jens: mutable: A Modern DBMS for Research and Fast Prototyping. In: CIDR. 2023.
- [La21] Lattner, Chris et al.: MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In: CGO. 2021.
- [SAH87] Stonebraker, Michael; Anton, Jeff; Hirohama, Michael: Extendability in POSTGRES. IEEE Data Eng. Bull., 10(2):16–23, 1987.