DAPHNE Runtime: Harnessing Parallelism for Integrated Data Analysis Pipelines

Aristotelis Vontzalidis[1], Stratos Psomadakis[1], Constantinos Bitsakos[1], Mark Dokter[2], Kevin Innerebner[3], Patrick Damme[4], Matthias Boehm[4], Florina Ciorba[5], Ahmed Eleliemy[5], Vasileios Karakostas[6], Aleš Zamuda[7], and Dimitrios Tsoumakos[1]

[1] ICCS–National Technical University of Athens
[2] Know-Center GmbH/TU Graz
[3] Graz University of Technology
[4] Technische Universität Berlin
[5] University of Basel
[6] University of Athens
[7] University of Maribor

Abstract. Integrated data analysis pipelines combine rigorous data management and processing, high-performance computing and machine learning tasks. While these systems and operations share many compilation and runtime techniques, data analysts and scientists are currently dealing with multiple systems for each stage of their pipeline. DAPHNE is an open and extensible system infrastructure for such pipelines, including language abstractions, compilation and runtime techniques, multi-level scheduling, hardware accelerators and computational storage. In this demonstration, we focus on the DAPHNE runtime that provides the implementation of kernels for local, distributed and accelerator-enhanced operations, vectorized execution, integration with existing frameworks and libraries for productivity and interoperability, as well as efficient I/O and communication primitives.

Keywords: Machine Learning Systems · High Performance Computing · Vectorized Execution · Distributed Systems

1 Introduction

Complex end-to-end analysis requirements of modern analytics create a definite trend towards integrated pipelines where data management, high-performance computing and ML tasks are arbitrarily "mixed-and-matched". Distinctive such use-cases or domains include ML-assisted simulations, exploratory query processing and data cleaning, etc. The DAPHNE project[8] is building an open and extensible system infrastructure for such integrated data analysis pipelines. DAPHNE [1] is built on top of MLIR [2], allowing seamless integration with existing applications and runtime libraries while also enabling extensibility for specialized data types, hardware-specific compilation chains and custom scheduling algorithms. Its technical contributions are available as open source[9] under the Apache-2.0

[8] https://daphne-eu.eu/
[9] https://github.com/daphne-eu/daphne

license. In this demonstration, we present an overview of the current design and implementation of the DAPHNE execution engine and describe the demonstration scenarios and level of interaction with the participants.

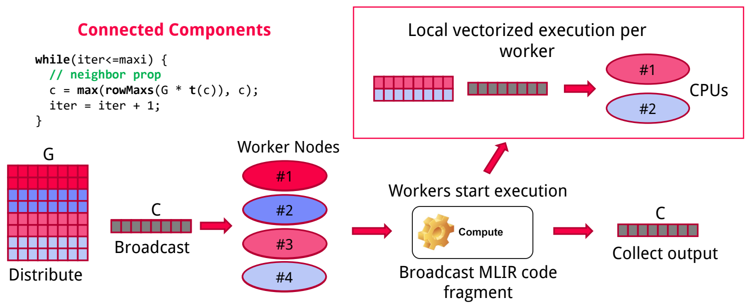
2 DAPHNE Runtime Overview

The DAPHNE Runtime system [3] (DR henceforth) is a crucial component of DAPHNE. It supports the execution of user-defined workflows and operations specified in DaphneDSL (a high-level scripting language) or DaphneLib (high-level Python API). The system utilizes a multi-level compilation chain based on the MLIR infrastructure to convert DaphneDSL scripts into DaphneIR, an intermediate representation. Multiple optimization passes allow for cost-based, pipelined and efficient execution of kernels, i.e., code that implements the logical operations on specific hardware in standalone or distributed mode. The runtime system is designed hierarchically. The coordinator receives DaphneDSL user code and generates an execution plan. The compiler determines whether each workload should be executed locally or across multiple worker nodes. The local runtime system handles execution on a single compute node, while the distributed runtime system coordinates the distribution of work among worker nodes and collects the results.

DR includes local and distributed kernels for executing computational, I/O and combined operations, supporting heterogeneous hardware devices (CPUs, GPUs and FPGAs). DR supports data structures such as matrices (both dense and sparse formats) and frames that have a schema and rely on column-oriented storage. DR's vectorized execution works by fusing multiple operations together and exploiting data parallelism. Data is split across multiple processing units (e.g., CPUs) and each processing unit works on a chunk of data (local runtime case). In distributed execution mode, DR uses distribution primitives (such as broadcast, all-reduce, ring-reduce, scatter/gather, etc.) to distribute data and code to worker nodes, similarly to the local runtime. Instead of CPUs, there are multiple distributed nodes that receive chunks of data and perform computations on them. Each worker locally compiles the received code fragment in order to optimize the code generation targeting its available resources (CPUs, accelerators, etc.), and executes the generated code through the local runtime system via the vectorized execution engine. This is pictorially described in Figure 1 for the Connected Components algorithm.

Communication between cluster nodes is facilitated by utilizing common frameworks. DR's design allows for easy integration with different frameworks. Currently, DR can successfully utilize gRPC and the MPI library. I/O support is significant, currently allowing CSV, Arrow and Matrix Market formats. It is also noteworthy to mention that DR implements a DAPHNE-specific file format along with custom (de)serialization support to enable more efficient I/O and network communication.

Overall, the DAPHNE Runtime plays a central role in executing integrated data analysis pipelines and optimizing performance through parallelism in heterogeneous hardware settings. DR uniquely allows researchers to experiment



```
while(iter<=maxi) {
  // neighbor prop
  c = max(rowMaxs(G * t(c)), c);
  iter = iter + 1;
}
```

Fig. 1. Example of distributing work hierarchically with the DAPHNE Runtime.

with different modules and subsystems (integration with different storage, communication and serialization protocols), as its extensible design allows for deployment combinations otherwise impossible by a single system.

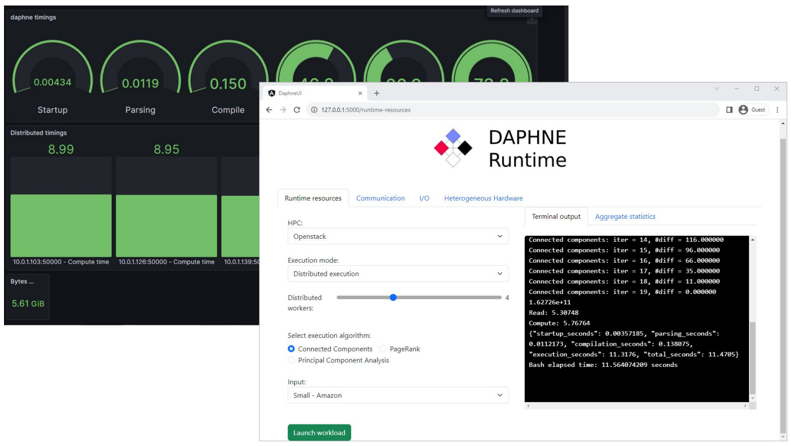
3 Demonstration Description

In this demonstration we showcase the capabilities of DR, providing participants with a comprehensive understanding of the runtime's functionality and versatility via different scenarios and configurations. While DR's contributions and parameters that affect its performance are manifold, in this demonstration we focus on the following important features-dimensions: a) Execution mode (standalone vs. distributed modes and scalability to available resources), b) communication frameworks (gRPC vs. MPI), c) I/O features (distributed filesystem integration and serialization support) and d) hardware-specific execution (utilization of accelerator resources).

Participants will be able to interact with DR via a comprehensive web-based GUI. The GUI controls two deployments, namely an in-house 16-node cluster and a Vega-bound deployment. Vega[10] is a powerful supercomputer infrastructure that boasts impressive processing power and a high-performance network, making it an ideal platform for showcasing the capabilities of DR. Three algorithms (Connected Components, PageRank and Principal Component Analysis) that utilize diverse inputs (real and synthetic data of various sizes and types) and kernels will be available for execution. The UI will integrate both textual and graphical execution feedback in order to visually inspect the quality and quantity of specific features under inspection. As such, our demonstration will showcase the following DR functionalities:

Execution mode: We will showcase the ability of executing a workload on a single machine versus distributing it across a cluster of variable size and configuration. Participants can choose between different pipelines, input sizes and cluster resources in order to study the performance trade-offs induced in each case. Moreover, they can compare the execution speed and efficiency of vectorized and non-vectorized computations, demonstrating DAPHNE's ability to leverage vectorization for performance. This functionality is shown in Figure 2, including a screenshot of our Grafana-based DR metrics visualizer.

[10] https://www.izum.si/en/vega-en/



HPC: Openstack

Execution mode: Distributed execution

Distributed workers: 4

Select execution algorithm: [✓ Connected Components] PageRank Principal Component Analysis

Input: Small - Amazon

```
Connected components: iter = 14, #diff = 116.000000
Connected components: iter = 15, #diff = 96.000000
Connected components: iter = 16, #diff = 66.000000
Connected components: iter = 17, #diff = 35.000000
Connected components: iter = 18, #diff = 11.000000
Connected components: iter = 19, #diff = 0.000000
1.62726e+11
Read: 5.30748
Compute: 5.76764
{"startup_seconds": 0.00357185, "parsing_seconds":
0.0112173, "compilation_seconds": 0.138075,
"execution_seconds": 11.3176, "total_seconds": 11.4705}
Bash elapsed time: 11.564074209 seconds
```

Fig. 2. Use of the DAPHNE Runtime UI.

Communication frameworks: We will allow workflow execution with both gRPC and MPI in different scenarios, such as latency-sensitive tasks and large-scale data transfers, so as to provide insights into their suitability for different algorithms currently implemented within DR.

I/O features: We will demonstrate how DR currently integrates with HDFS, showcasing its ability to efficiently read data of variable sizes by leveraging data locality in large-scale I/O compared to using the local file system. The participants can also observe the enhanced performance that the DAPHNE serialization protocol brings forth compared to a default (protobuf) serialization protocol.

Hardware-specific execution: We will showcase how DR efficiently manages the workload distribution and data movement using different hardware components. Specifically, for the Connected Components algorithm (with available accelerator-aware kernels), the DR will offload computational tasks to accelerators assuming their availability. By comparing the execution times and throughput of tasks on a cluster with accelerators versus a cluster without, the audience can witness the speedup achieved through heterogeneous hardware execution.

ACKNOWLEDGEMENTS

The DAPHNE project is funded by the European Union's Horizon 2020 research and innovation program under grant agreement number 957407 from 12/2020 through 11/2024.

References

1. Damme, P., et.al.: DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. In: CIDR (2022)
2. Lattner, C., et.al.: MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In: CGO 2021 (2021)
3. D4.2: DSL Runtime Prototype. Public EU Project Deliverable. https://daphne-eu.eu/wp-content/uploads/2022/12/D4.2-DSL-Runtime-Prototype.pdf (2022)