Seamlessly Scaling Applications with DAPHNE

COMPAS 2024, Nantes, France

Quentin GUILLOTEAU, Jonas H. Müller KORNDÖRFER, Florina M. CIORBA

2024-07-04

University of Basel, Switzerland
{Quentin.Guilloteau, Jonas.Korndorfer, Florina.Ciorba}@unibas.ch

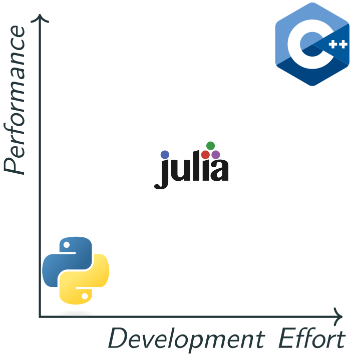
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 957407

The “Two-Language Problem”

- **Fast** development (e.g. Python) ...
- But reimplementation for **performance** (e.g. C++)

→ Julia as a “solution”

- **Trade-off**: Performance vs. Ease of Development



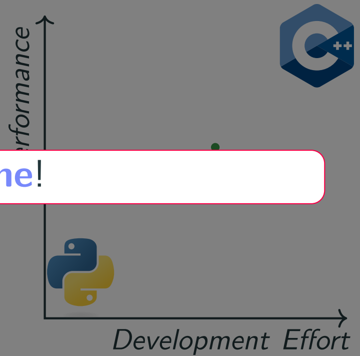
The “Two-Language Problem”

- **Fast** development (e.g. Python) ...
- Rewrite implementation for **performance** (e.g. C++) ...

→ Julia as a “solution”

↪ Only for a **single machine**!

- **Trade-off**: Performance vs. Ease of Development



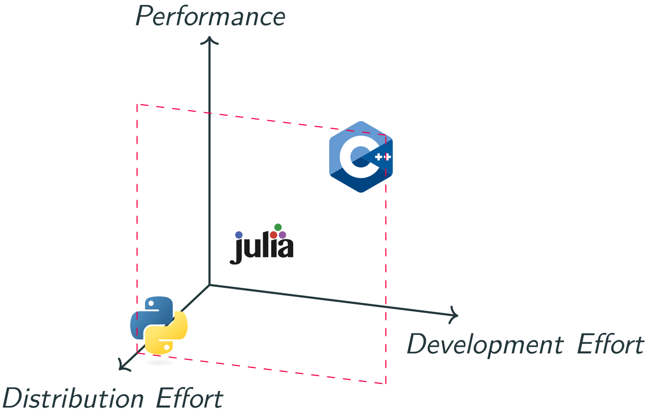
The “Two-Language Problem” – Beyond a single machine

Rewrite the application to:

- Distribute the computations
- Integrate communication library (e.g. **MPI**)

Requires:

- **More substantial effort**
- Error handling
- Additional expertise



The “Two-Language Problem” – Beyond a single machine

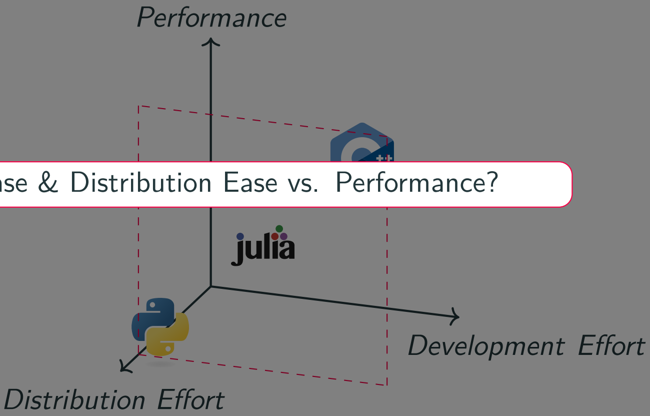
Rewrite the application to:

- Distribute the computations
- Integrate communication library (e.g. **MPI**)

↪ **Trade-off**: Development Ease & Distribution Ease vs. Performance?

Requ

- **More substantial effort**
- Error handling
- Additional expertise



DAPHNE

An Open and Extensible System Infrastructure for IDA Pipelines

DAPHNE – Motivation

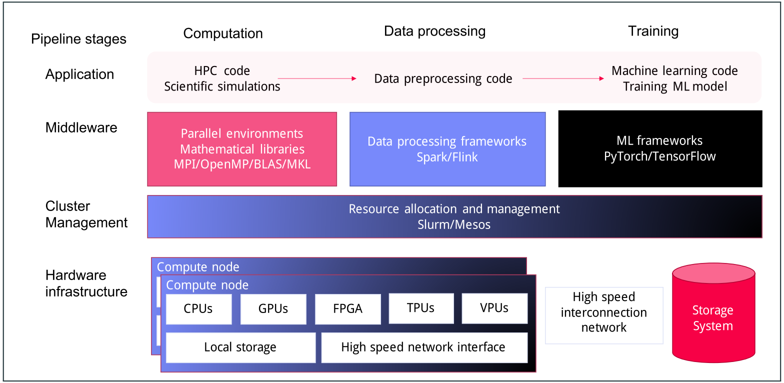
- Integrated Data Analysis pipelines: HPC → DM → ML
- **Different** libraries, programming models, etc. at **every stage**!

OpenMP

MPI

NVIDIA CUDA

hadoop



Pipeline stages

Computation

Data processing

Training

Application

HPC code
Scientific simulations

Data preprocessing code

Machine learning code
Training ML model

Middleware

Parallel environments
Mathematical libraries
MPI/OpenMP/BLAS/MKL

Data processing frameworks
Spark/Flink

ML frameworks
PyTorch/TensorFlow

Cluster Management

Resource allocation and management
Slurm/Mesos

Hardware infrastructure

Compute node

Compute node

CPUs

GPUs

FPGA

TPUs

VPUs

Local storage

High speed network interface

High speed interconnection network

Storage System

APACHE Spark

Flink

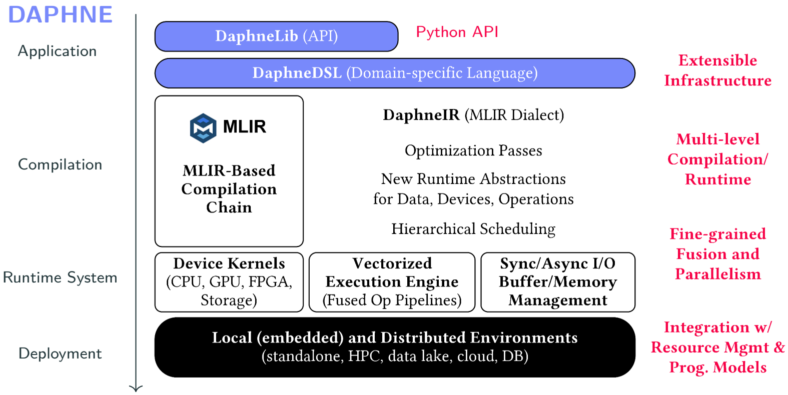
PyTorch

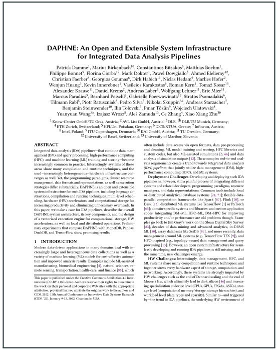
mxnet

DAPHNE – Overview

- EU H2020 Project (2021-2024)
- Open-source: https://github.com/daphne-eu/daphne

DAPHNE


DAPHNE
Application
Compilation
Runtime System
Deployment
DaphneLib (API)
Python API
DaphneDSL (Domain-specific Language)
MLIR
MLIR-Based Compilation Chain
DaphneIR (MLIR Dialect)
Optimization Passes
New Runtime Abstractions for Data, Devices, Operations
Hierarchical Scheduling
Device Kernels (CPU, GPU, FPGA, Storage)
Vectorized Execution Engine (Fused Op Pipelines)
Sync/Async I/O Buffer/Memory Management
Local (embedded) and Distributed Environments (standalone, HPC, data lake, cloud, DB)
Extensible Infrastructure
Multi-level Compilation/ Runtime
Fine-grained Fusion and Parallelism
Integration w/ Resource Mgmt & Prog. Models



DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines

Patrick Damme[1], Marius Birkenbach[10], Constantinos Bitsakos[6], Matthias Boehm[1], Philippe Bonnet[9], Florina Ciorba[12], Mark Dokter[1], Pawel Dowgiallo[8], Ahmed Eleliemy[12], Christian Faerber[8], Georgios Goumas[6], Dirk Habich[11], Niclas Hedam[9], Marlies Hofer[2], Wenjun Huang[3], Kevin Innerebner[1], Vasileios Karakostas[6], Roman Kern[1], Tomaž Kosar[13], Alexander Krause[11], Daniel Krems[2], Andreas Laber[7], Wolfgang Lehner[11], Eric Mier[11], Marcus Paradies[3], Bernhard Peischl[2], Gabrielle Poerwawinata[12], Stratos Psomadakis[6], Tilmann Rabl[5], Piotr Ratuszniak[8], Pedro Silva[5], Nikolai Skuppin[3b], Andreas Starzacher[7], Benjamin Steinwender[10], Ilin Tolovski[5], Pınar Tözün[9], Wojciech Ulatowski[8], Yuanyuan Wang[3b], Izajasz Wrosz[8], Aleš Zamuda[13], Ce Zhang[4], Xiao Xiang Zhu[3b]

[1] Know-Center GmbH/TU Graz, Austria; [2] AVL List GmbH, Austria; [3] DLR, [3b] DLR/TU Munich, Germany; [4] ETH Zurich, Switzerland; [5] HPI/Uni Potsdam, Germany; [6] ICCS/NTUA, Greece; [7] Infineon, Austria; [8] Intel, Poland; [9] ITU Copenhagen, Denmark; [10] KAI GmbH, Austria; [11] TU Dresden, Germany; [12] University of Basel, Switzerland; [13] University of Maribor, Slovenia

ABSTRACT

Integrated data analysis (IDA) pipelines—that combine data management (DM) and query processing, high-performance computing (HPC), and machine learning (ML) training and scoring—become increasingly common in practice. Interestingly, systems of these areas share many compilation and runtime techniques, and the used—increasingly heterogeneous—hardware infrastructure converges as well. Yet, the programming paradigms, cluster resource management, data formats and representations, as well as execution strategies differ substantially. DAPHNE is an open and extensible system infrastructure for such IDA pipelines, including language abstractions, compilation and runtime techniques, multi-level scheduling, hardware (HW) accelerators, and computational storage for increasing productivity and eliminating unnecessary overheads. In this paper, we make a case for IDA pipelines, describe the overall DAPHNE system architecture, its key components, and the design of a vectorized execution engine for computational storage, HW accelerators, as well as local and distributed operations. Preliminary experiments that compare DAPHNE with MonetDB, Pandas, DuckDB, and TensorFlow show promising results.

1 INTRODUCTION

Modern data-driven applications in many domains deal with increasingly large and heterogeneous data collections as well as a variety of machine learning (ML) models for cost-effective automation and improved analysis results. Examples include ML-assisted manufacturing, biomedical engineering [4], natural sciences, remote sensing, transportation, health-care, and finance [88], which often include data access via open formats, data pre-processing and cleaning, ML model training and scoring, HPC libraries and custom codes, but also ML-assisted simulations [3, 66] and data analysis of simulation outputs [12]. These complex end-to-end analysis requirements create a trend towards *integrated data analysis (IDA) pipelines* that jointly utilize data management (DM), high-performance computing (HPC), and ML systems.

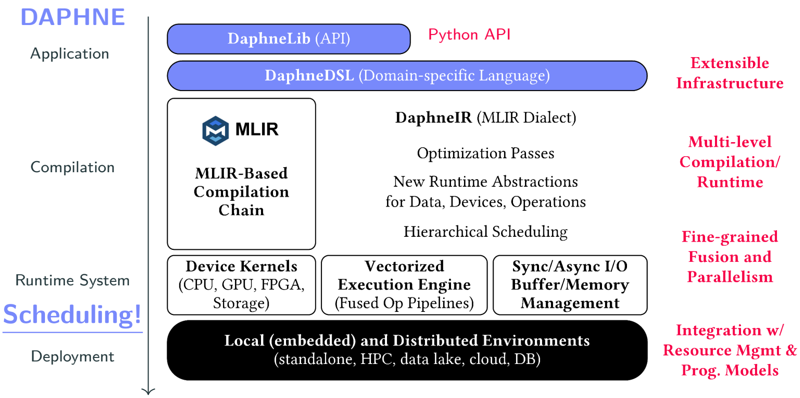
Deployment Challenges: Developing and deploying such IDA pipelines is, however, still a painful process of integrating different systems and related developers, programming paradigms, resource managers, and data representations. Common tools include local or distributed analytical database systems [24, 71]; flexible data-parallel computation frameworks like Spark [87], Flink [20], or Dask [73]; distributed ML systems like TensorFlow [1] or PyTorch [65]; domain-specific systems and libraries; and custom application codes. Integrating DM+ML, HPC+ML, DM+HPC for improving productivity and/or performance are old problems though. Examples go back to Jim Gray's work on the Sloan Digital Sky Survey [83], decades of data mining and advanced analytics, in-DBMS ML [50], array databases like SciDB [82], and more recently, data management around ML systems (e.g., TensorFlow TFX [9]), and HPC-inspired (e.g., topology-aware) data management and query processing [13]. However, an open system infrastructure for seamlessly developing and running IDA pipelines is still missing, and at the same time, new challenges emerge.

HW Challenges: Interestingly, data management, HPC, and ML systems share many compilation and runtime techniques; and together stress every hardware aspect of storage, computation, and networking. Accordingly, these systems are strongly impacted by HW challenges such as the end of Dennard scaling and the end of Moore's law, which ultimately lead to dark silicon [46] and increasing specialization at device level (CPUs, GPUs, FPGAs, ASICs), storage level (computational memory/storage, storage hierarchies), and workload level (data types and sparsity). Similar to—and triggered by—the trend to IDA pipelines, the underlying HW environment of

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

DAPHNE – Overview

- EU H2020 Project (2021-2024)
- Open-source: `https://github.com/daphne-eu/daphne`



DAPHNE

Application

Compilation

Runtime System

Scheduling!

Deployment

DaphneLib (API)

Python API

DaphneDSL (Domain-specific Language)

MLIR

MLIR-Based Compilation Chain

DaphneIR (MLIR Dialect)

Optimization Passes

New Runtime Abstractions for Data, Devices, Operations

Hierarchical Scheduling

Device Kernels (CPU, GPU, FPGA, Storage)

Vectorized Execution Engine (Fused Op Pipelines)

Sync/Async I/O Buffer/Memory Management

Local (embedded) and Distributed Environments (standalone, HPC, data lake, cloud, DB)

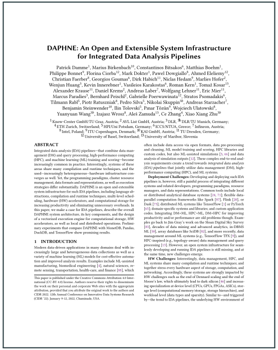
Extensible Infrastructure

Multi-level Compilation/ Runtime

Fine-grained Fusion and Parallelism

Integration w/ Resource Mgmt & Prog. Models

DAPHNE



DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines

Patrick Damme[1], Marius Birkenbach[10], Constantinos Bitsakos[6], Matthias Boehm[1], Philippe Bonnet[9], Florina Ciorba[12], Mark Dokter[1], Pawel Dowgiallo[8], Ahmed Eleliemy[12], Christian Faerber[8], Georgios Goumas[6], Dirk Habich[11], Niclas Hedam[9], Marlies Hofer[2], Wenjun Huang[3], Kevin Innerebner[1], Vasileios Karakostas[6], Roman Kern[1], Tomaž Kosar[13], Alexander Krause[11], Daniel Krems[2], Andreas Laber[7], Wolfgang Lehner[11], Eric Mier[11], Marcus Paradies[3], Bernhard Peischl[2], Gabrielle Poerwawinata[12], Stratos Psomadakis[6], Tilmann Rabl[5], Piotr Ratuszniak[8], Pedro Silva[5], Nikolai Skuppin[3b], Andreas Starzacher[7], Benjamin Steinwender[10], Ilin Tolovski[5], Pınar Tözün[9], Wojciech Ulatowski[8], Yuanyuan Wang[3b], Izajasz Wrosz[8], Aleš Zamuda[13], Ce Zhang[4], Xiao Xiang Zhu[3b]

[1] Know-Center GmbH/TU Graz, Austria; [2] AVL List GmbH, Austria; [3] DLR, [3b] DLR/TU Munich, Germany; [4] ETH Zurich, Switzerland; [5] HPI/Uni Potsdam, Germany; [6] ICCS/NTUA, Greece; [7] Infineon, Austria; [8] Intel, Poland; [9] ITU Copenhagen, Denmark; [10] KAI GmbH, Austria; [11] TU Dresden, Germany; [12] University of Basel, Switzerland; [13] University of Maribor, Slovenia

ABSTRACT

Integrated data analysis (IDA) pipelines—that combine data management (DM) and query processing, high-performance computing (HPC), and machine learning (ML) training and scoring—become increasingly common in practice. Interestingly, systems of these areas share many compilation and runtime techniques, and the used—increasingly heterogeneous—hardware infrastructure converges as well. Yet, the programming paradigms, cluster resource management, data formats and representations, as well as execution strategies differ substantially. DAPHNE is an open and extensible system infrastructure for such IDA pipelines, including language abstractions, compilation and runtime techniques, multi-level scheduling, hardware (HW) accelerators, and computational storage for increasing productivity and eliminating unnecessary overheads. In this paper, we make a case for IDA pipelines, describe the overall DAPHNE system architecture, its key components, and the design of a vectorized execution engine for computational storage, HW accelerators, as well as local and distributed operations. Preliminary experiments that compare DAPHNE with MonetDB, Pandas, DuckDB, and TensorFlow show promising results.

1 INTRODUCTION

Modern data-driven applications in many domains deal with increasingly large and heterogeneous data collections as well as a variety of machine learning (ML) models for cost-effective automation and improved analysis results. Examples include ML-assisted manufacturing, biomedical engineering [4], natural sciences, remote sensing, transportation, health-care, and finance [88], which often include data access via open formats, data pre-processing and cleaning, ML model training and scoring, HPC libraries and custom codes, but also ML-assisted simulations [3, 66] and data analysis of simulation outputs [12]. These complex end-to-end analysis requirements create a trend towards *integrated data analysis (IDA) pipelines* that jointly utilize data management (DM), high-performance computing (HPC), and ML systems.

Deployment Challenges: Developing and deploying such IDA pipelines is, however, still a painful process of integrating different systems and related developers, programming paradigms, resource managers, and data representations. Common tools include local or distributed analytical database systems [24, 71]; flexible data-parallel computation frameworks like Spark [87], Flink [20], or Dask [73]; distributed ML systems like TensorFlow [1] or PyTorch [65]; domain-specific systems and libraries; and custom application codes. Integrating DM+ML, HPC+ML, DM+HPC for improving productivity and/or performance are old problems though. Examples go back to Jim Gray's work on the Sloan Digital Sky Survey [83], decades of data mining and advanced analytics, in-DBMS ML [50], array databases like SciDB [82], and more recently, data management around ML systems (e.g., TensorFlow TFX [9]), and HPC-inspired (e.g., topology-aware) data management and query processing [13]. However, an open system infrastructure for seamlessly developing and running IDA pipelines is still missing, and at the same time, new challenges emerge.

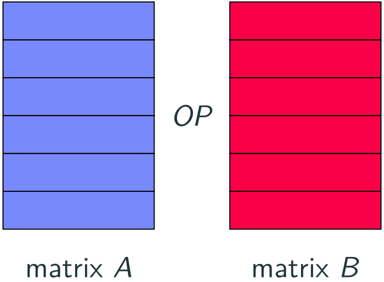
HW Challenges: Interestingly, data management, HPC, and ML systems share many compilation and runtime techniques; and together stress every hardware aspect of storage, computation, and networking. Accordingly, these systems are strongly impacted by HW challenges such as the end of Dennard scaling and the end of Moore's law, which ultimately lead to dark silicon [46] and increasing specialization at device level (CPUs, GPUs, FPGAs, ASICs), storage level (computational memory/storage, storage hierarchies), and workload level (data types and sparsity). Similar to—and triggered by—the trend to IDA pipelines, the underlying HW environment of

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

DaphneSched

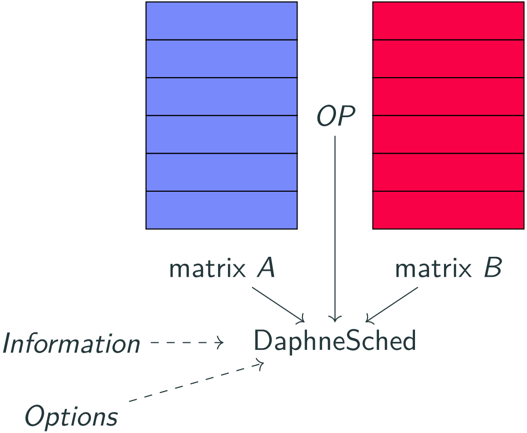
Local and Distributed Versions

DaphneSched – Local Scheduler – Walk-through

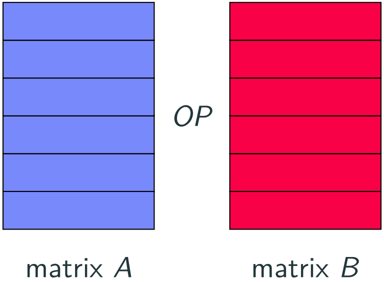


matrix A OP matrix B

DaphneSched – Local Scheduler – Walk-through

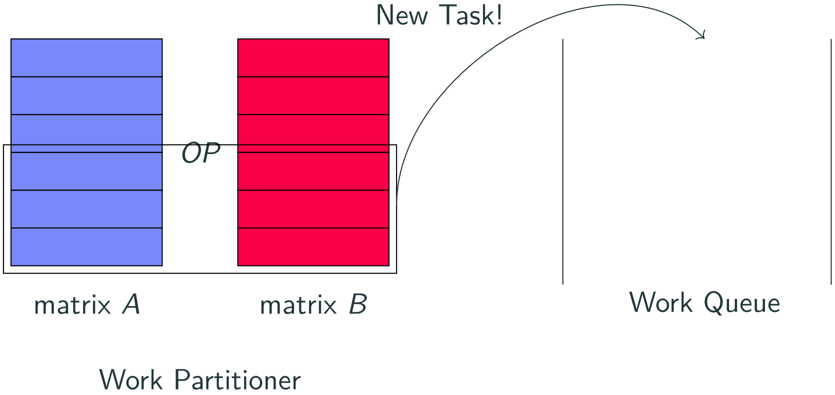


DaphneSched – Local Scheduler – Walk-through

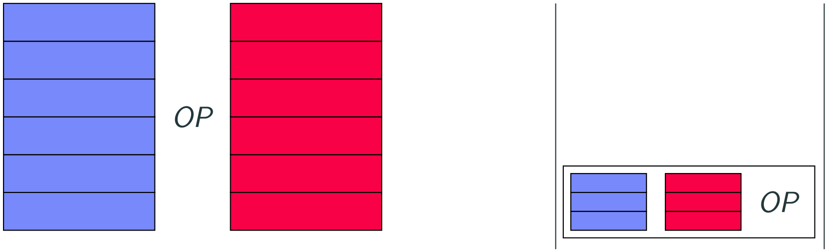


Work Partitioner

DaphneSched – Local Scheduler – Walk-through



DaphneSched – Local Scheduler – Walk-through



OP

matrix A

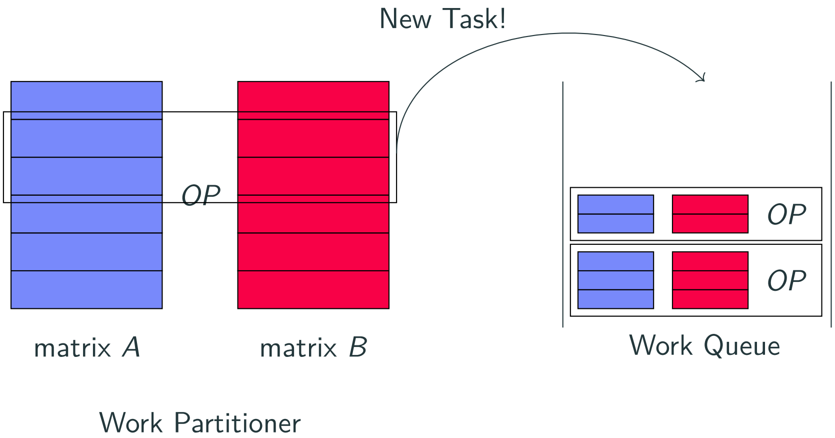
matrix B

Work Partitioner

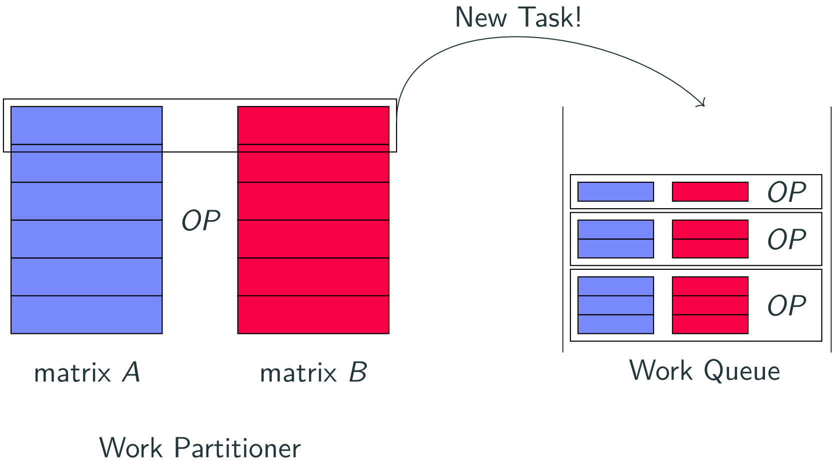
OP

Work Queue

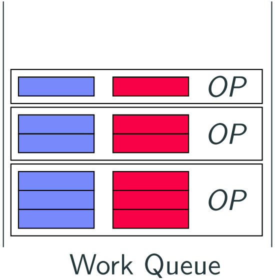
DaphneSched – Local Scheduler – Walk-through



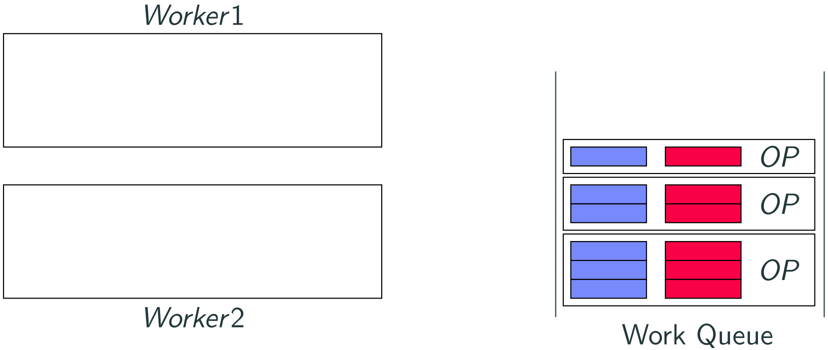
DaphneSched – Local Scheduler – Walk-through



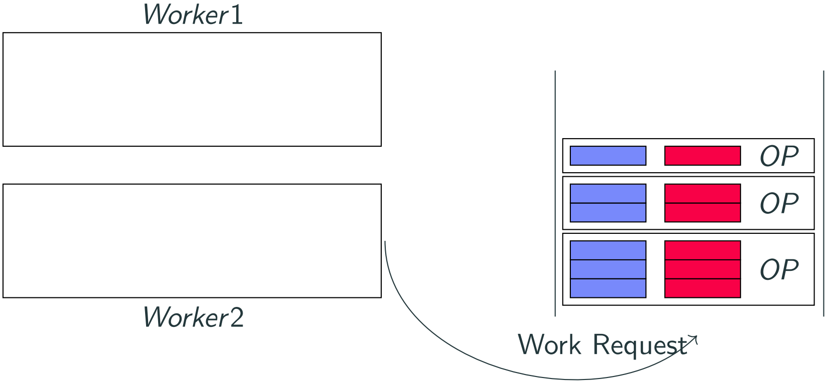
DaphneSched – Local Scheduler – Walk-through



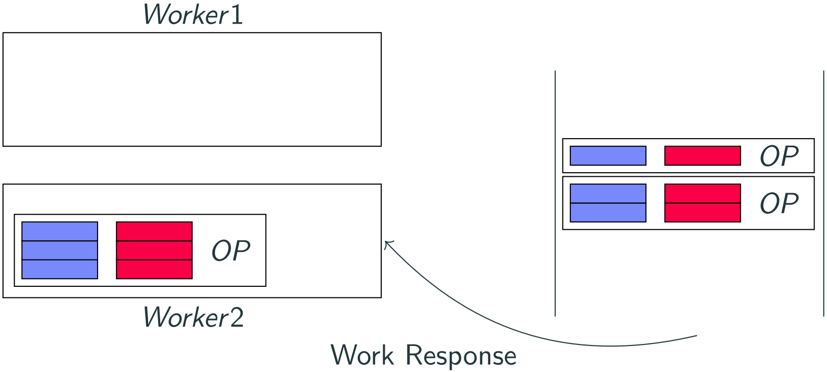
DaphneSched – Local Scheduler – Walk-through



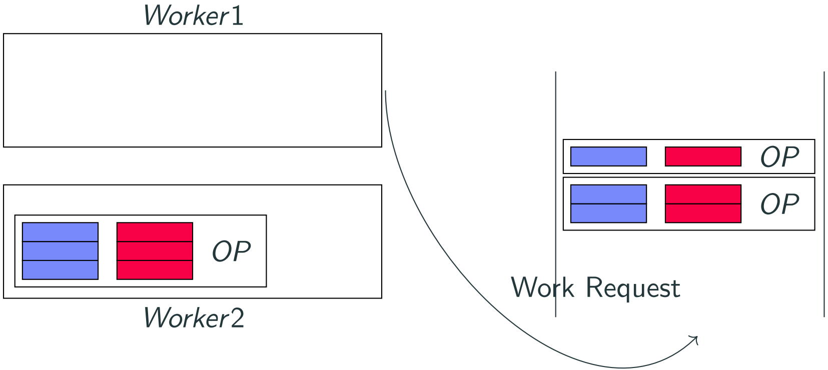
DaphneSched – Local Scheduler – Walk-through



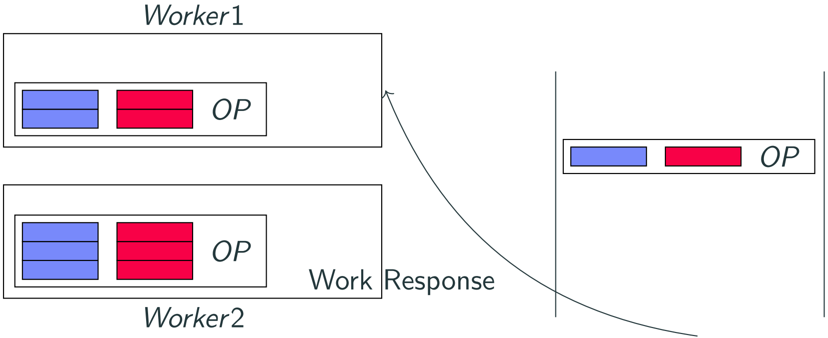
DaphneSched – Local Scheduler – Walk-through



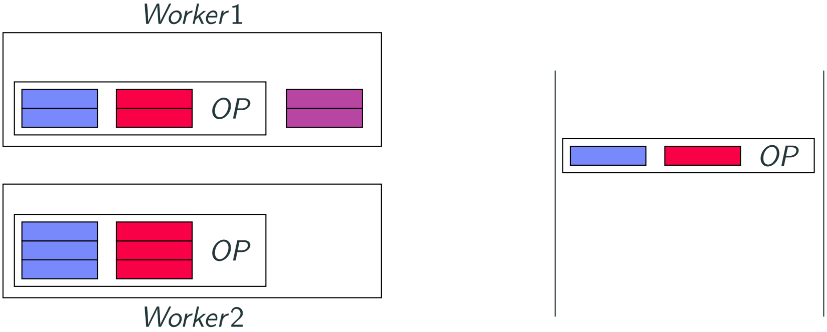
DaphneSched – Local Scheduler – Walk-through



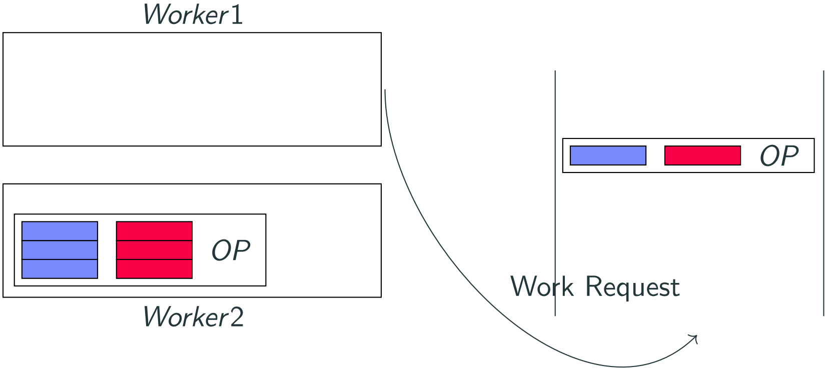
DaphneSched – Local Scheduler – Walk-through



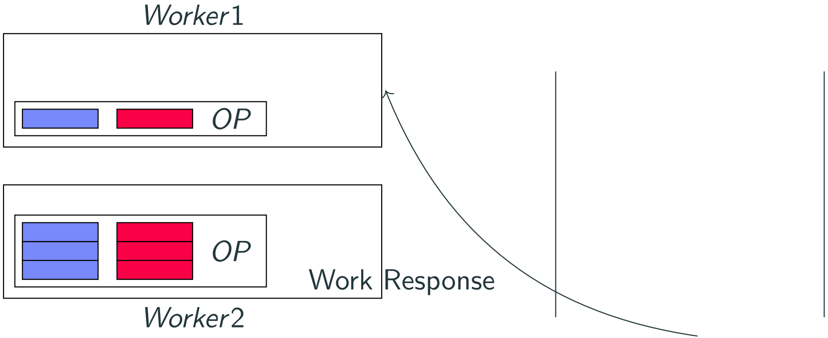
DaphneSched – Local Scheduler – Walk-through



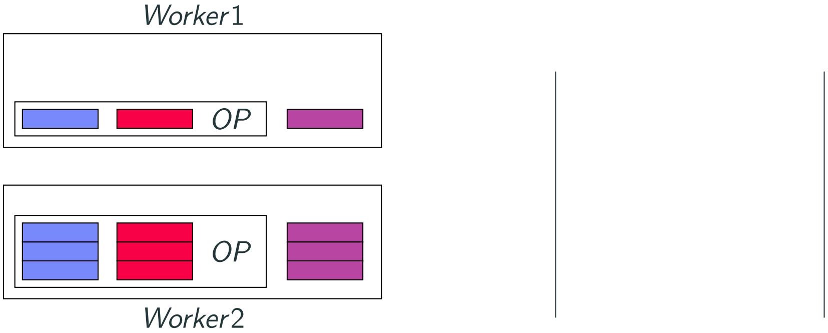
DaphneSched – Local Scheduler – Walk-through



DaphneSched – Local Scheduler – Walk-through



DaphneSched – Local Scheduler – Walk-through



DaphneSched – Local Scheduler (A. Eleliemy, F. M. Ciorba, ISPDC 2023)

12 Work Partitioners

STATIC (OpenMP static),

SS (OpenMP dynamic),

GSS (OpenMP guided),

TSS (in LLVM),

FAC2, TFSS, FISS, VISS, PLS,

MSTATIC, MFSC, PSS (in LB4OMP),

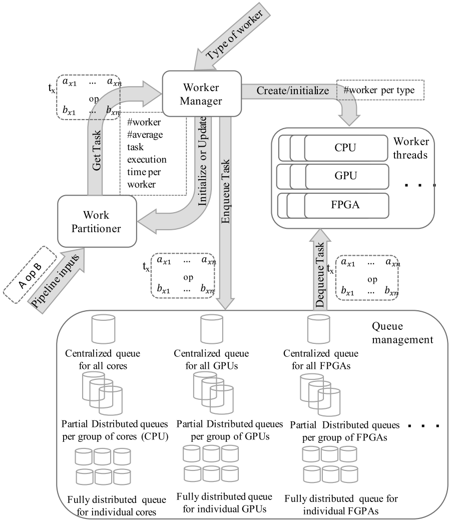
and AUTO

3 Queue Layouts

CENTRALIZED, PERGROUP*, PERCPU*

4 Work Stealing* Strategies

SEQ, SEQPRI, RND, RNDPRI


Type of worker
Worker Manager
Create/initialize
#worker per type
t_x
a_{x1} ... a_{xn}
op
b_{x1} ... b_{xn}
Get Task
#worker
#average task execution time per worker
Initialize or Update
Enqueue Task
CPU
GPU
FPGA
Worker threads
Work Partitioner
A op B
Pipeline inputs
Dequeue Task
Queue management
Centralized queue for all cores
Centralized queue for all GPUs
Centralized queue for all FPGAs
Partial Distributed queues per group of cores (CPU)
Partial Distributed queues per group of GPUs
Partial Distributed queues per group of FPGAs
Fully distributed queue for individual cores
Fully distributed queue for individual GPUs
Fully distributed queue for individual FGPAs

Local DaphneSched

DaphneSched – Local Scheduler (A. Eleliemy, F. M. Ciorba, ISPDC 2023)

12 Work Partitioners

`STATIC` (OpenMP `static`),

`SS` (OpenMP `dynamic`),

`GSS` (OpenMP `guided`),

`TSS` (in LLVM),

`FAC2`, `TFSS`, `FISS`, `VISS`, `PLS`,

`MSTATIC`, `MFSC`, `PSS` (in LB4OMP),

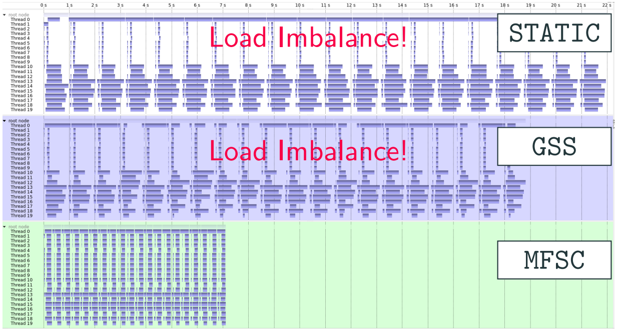
and `AUTO`

3 Queue Layouts

`CENTRALIZED`, `PERGROUP`*, `PERCPU`*

4 Work Stealing* Strategies

`SEQ`, `SEQPRI`, `RND`, `RNDPRI`



Connected Components in DaphneDSL, executed with DaphneSched and **sparse** input matrix (`wikipedia-20070206`) on a single node

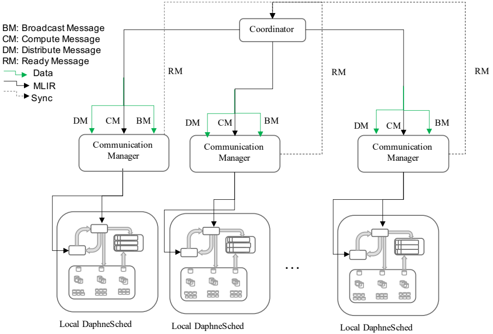
DaphneSched – Distributed Scheduler ⇒ Hierarchical Scheduling – NEW!

Coordinator (MPI Rank 0)

- Partitions work (data, ops) to the local DaphneSched instances
- Communicator Manager coordinates with local DaphneSched instances
- Different message types:
  - BM: Broadcast Message (Data)
  - CM: Compute Message (MLIR)
  - DM: Distribute Message (Data)
  - RM: Ready Message (Sync)

Workers (MPI Ranks 1 .. P-1)

- Listen incoming messages from coordinator
- Execute a local DaphneSched instance



Distributed DaphneSched

Experimental Evaluation

Design of Experiments

- Study the **effort and performance of scaling applications**
- Compare C++, Python, Julia, DaphneDSL
- **Connected Components** graph algorithm: broadcast, max, max (S. Beamer et al., GAP Benchmark suite)
- 2 Intel Broadwell E5-2640v4 CPUs, 10 cores each, 64GB of RAM
- 3 input matrices:

Matrix	Size	Density (%)
`amazon0601`	403'394 × 403'394	2.08×10^{-3}
`wikipedia-20070206`	3'566'907 × 3'566'907	0.354×10^{-3}
`ljournal-2008`	5'363'260 × 5'363'260	0.275×10^{-3}

Implementations – Connected Components (CC) Algorithm

Language (abbreviation)	External dependencies	Lines of code per implementation		
		Sequential	Local Parallel	Distributed Parallel
C++ (`cpp`)	Eigen	$\simeq 25$	$\simeq 25$	$\simeq 120$
Python (`py`)	Numpy, Scipy	$\simeq 10$	$\simeq 10$	$\simeq 100$
Julia (`jl`)	MatrixMarket.jl	$\simeq 25$	$\simeq 25$	$\simeq 100$
DaphneDSL (`daph`)	$\varnothing$	$\simeq$ **10**	$\simeq$ **10**	$\simeq$ **10**

- `CC`: broadcast (dense vector to CSR matrix), row-wise max, vector-wise max
- `cpp`, `jl`: Broadcast implemented by hand
- MPI: encapsulates the local parallel version, between a scatter of CSR matrix and `MPI_Allreduce` with user-defined function

Implementations – CC in DaphneDSL

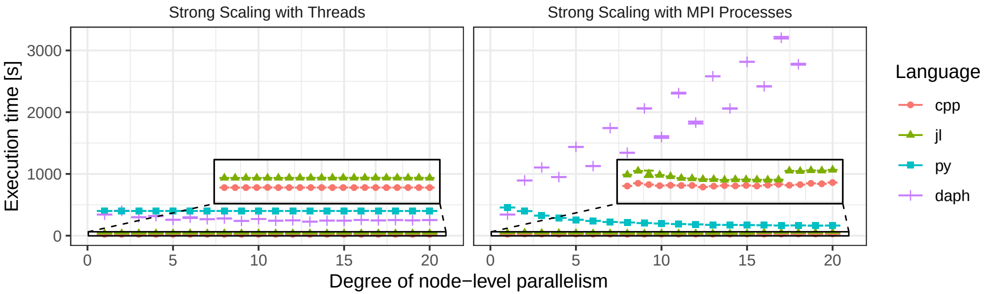
```
G = readMatrix($f); // read sparse matrix from CLI argument
maxi = 100; // maximum number of iterations
start = now();

c = seq(1.0, as.f64(nrow(G)), 1.0); // initialization

for(iter in 1:maxi) {
    c = max(aggMax(G * t(c), 0), c);
}

end=now();
print((end-start) / 1000000000.0);
```

Results – Local Strong Scaling with Threads or Processes?



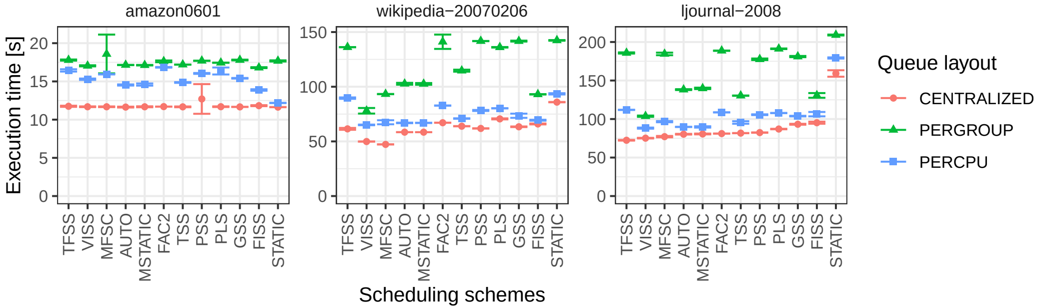
Strong scaling on a *single node* (20 total cores) with threads (left) and MPI processes (right) for CC with `wikipedia-20070206` as input.

Error bars: 95% confidence intervals.

DaphneSched: `CENTRALIZED` queue w/ `STATIC`.

- DaphneDSL scales with threads
- The others scale with processes

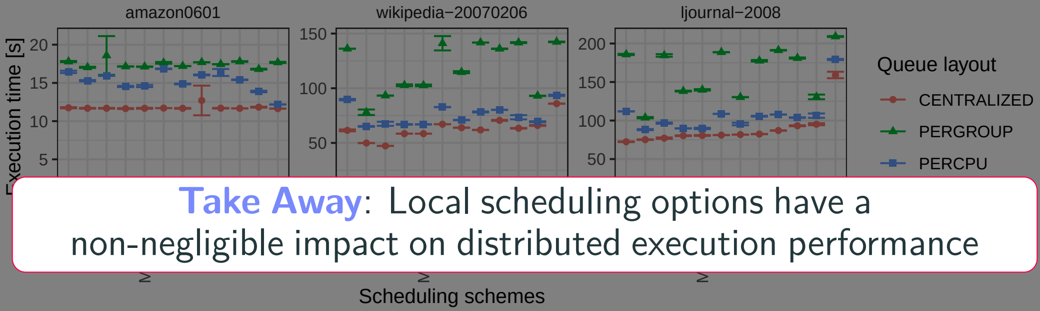
Results – Distributed Scheduling Schemes and Queue Layout ‖ DaphneDSL



Average execution time with 95% confidence intervals for CC with DaphneDSL and DaphneSched, for 12 scheduling schemes and 3 queue layouts, with 1 victim selection – `SEQPRI`, executed on 4 nodes and 1 MPI process / node.

Total degree of parallelism: (4 - 1) x 20 = 60 DAPHNE workers.

Results – Distributed Scheduling Schemes and Queue Layout ‖ DaphneDSL

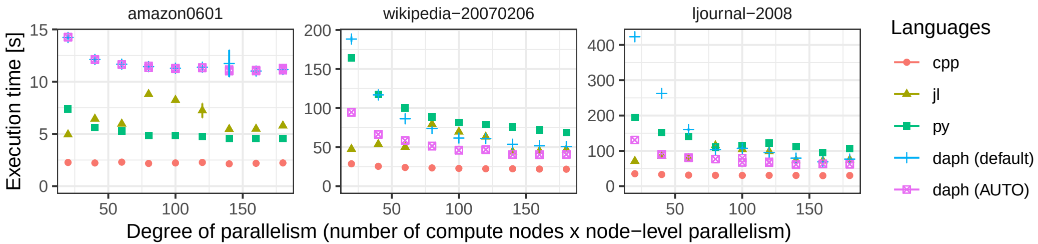


Take Away: Local scheduling options have a non-negligible impact on distributed execution performance

Average execution time with 95% confidence intervals for CC with DaphneDSL and DaphneSched, for 12 scheduling schemes and 3 queue layouts, with 1 victim selection – `SEQPRI`, executed on 4 nodes and 1 MPI process / node.

Total degree of parallelism: (4 - 1) x 20 = 60 DAPHNE workers.

Results – Distributed Strong Scaling For Different Inputs and Implementations



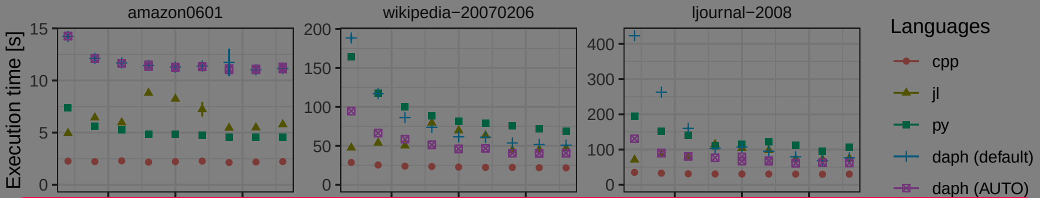
Strong scaling: 1-9 compute nodes. Inside each node, work is parallelized using MPI for C++, Julia, and Python, and threads for DaphneDSL.

DaphneSched: used `CENTRALIZED + STATIC` (default) and `CENTRALIZED + AUTO`

⇒ highlight the impact of scheduling on performance.

- DAPHNE is outperformed by others on small inputs ☹ ⇒ Impact of scheduling
- DAPHNE outperforms others on larger inputs ☺ ✓ **No additional effort!**

Results – Distributed Strong Scaling For Different Inputs and Implementations



Take Away: Seamlessly Scaling Applications with DAPHNE!

Strong scaling: 1-9 compute nodes. Inside each node, work is parallelized using MPI for C++, Julia, and Python, and threads for DaphneDSL.

DaphneSched: used `CENTRALIZED + STATIC` (default) and `CENTRALIZED + AUTO`
⇒ highlight the impact of scheduling on performance.

- DAPHNE is outperformed by others on small inputs ☹ ⇒ Impact of scheduling
- DAPHNE outperforms others on larger inputs ☺ ✓ No additional effort!

Conclusion and Future Steps

Conclusion and Future Steps

Conclusion

- Distributing applications → Difficult, substantial effort, expertise
- **DAPHNE scales seamlessly** (without additional effort) ☺
- Best performance is not always guaranteed ☹
- Interesting trade-off: **Performance vs. Ease of Development**

DAPHNE

Future Steps

- Dynamic partitioning by the coordinator in Distributed DaphneSched
- Communication/Stealing between the distributed workers
- Trade-off between colocating the coordinator with workers?
- Evaluation with full IDA pipelines

University of Basel

DAPHNE

DMI HPC

Seamlessly Scaling Applications with DAPHNE

COMPAS 2024, Nantes, France

Quentin GUILLOTEAU, Jonas H. Müller KORNDÖRFER, Florina M. CIORBA

2024-07-04

University of Basel, Switzerland

`{Quentin.Guilloteau, Jonas.Korndorfer, Florina.Ciorba}@unibas.ch`

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 957407

Additional Slides

Implementations – CC in DaphneDSL

```
G = readMatrix($f); // read sparse matrix from CLI argument
maxi = 100; // maximum number of iterations
start = now();

c = seq(1.0, as.f64(nrow(G)), 1.0); // initialization

for(iter in 1:maxi) {
    c = max(aggMax(G * t(c), 0), c);
}

end=now();
print((end-start) / 1000000000.0);
```

Implementations – CC in Python

```python
import sys
import time
from scipy.io import mmread
from scipy.sparse import csr_matrix, csr_array
import numpy as np

def cc(filename, maxi=100):
    G = csr_matrix(mmread(filename))
    n = G.shape[1]
    start = time.time()
    c = np.array([list(map(lambda i: float(i), range(1, n + 1, 1)))])
    for iter in range(maxi):
        x = G.multiply(c.transpose()).max(axis=0)
        c = np.maximum(c, x.todense())
    end = time.time()
    print(end - start)
```

Implementations – CC in Julia

```julia
using MatrixMarket
using SparseArrays
using SparseMatricesCSR

function G_broadcast_mult_c(G, c)
  cols = colvals(G)
  vals = nonzeros(G)
  m, n = size(G)
  maxs = zeros(n)
  for j = 1:m
     for i in nzrange(G, j)
        col = cols[i]
        val = vals[i]
        if val * c[j] > maxs[col]
          maxs[col] = val*c[j]
        end
     end
  end
```

Implementations – CC in C++

```cpp
#include <iostream>
#include <Eigen/SparseCore>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <Eigen/Core>
#include <unsupported/Eigen/SparseExtra>
#include <chrono>

typedef Eigen::SparseMatrix<double, Eigen::RowMajor> SpMatR;
typedef SpMatR::InnerIterator InIterMatR;

int main(int argc, char** argv) {
  if (argc != 3) {
    std::cout << "Usage: bin mat.mtx size" << std::endl;
    return 1;
  }
  std::string filename = argv[1];
```