Exploiting dynamic sparse matrices for performance portable linear algebra operations

Christodoulos Stylianou[1,a] Michèle Weiland[1]

[1]EPCC, The University of Edinburgh, UK

[a]c.stylianou@ed.ac.uk

2022 International Workshop on Performance, Portability & Productivity in HPC

13 November 2022

Introduction

- Sparse matrices essential concept in computational science and engineering
- Sparse matrix storage format are different in-memory representations of sparse matrices
  - Each designed to exploit strengths of the different hardware architectures or sparsity pattern of the matrix
- More than 70 formats have been developed over the years - still no single one performs best across:
  - Different sparsity patterns
  - Different target architectures
  - Different operations
- Most code-bases today still use a single format (CSR)
  - Adapting the data structure at run-time offers new optimization opportunities

Sparse Matrix Storage Formats

	0	1	2	3	4
0	1	2		11	
1		3	4		
2		5	6	7	
3				8	
4				9	10

Dense Matrix

row	0	0	0	1	1	2	2	2	3	4	4
col	0	1	3	1	2	1	2	3	3	3	4
val	1	2	11	3	4	5	6	7	8	9	10

COO Representation

row offset	0	3	5	8	9	11

col	0	1	3	1	2	1	2	3	3	3	4
val	1	2	11	3	4	5	6	7	8	9	10

CSR Representation

Diagonal Offsets	-1	0	1	3

val

*	1	2	11
0	3	4	0
5	6	7	*
0	8	0	*
9	10	*	*

DIA Representation

col

0	1	3
1	2	*
1	2	3
3	*	*
3	4	*

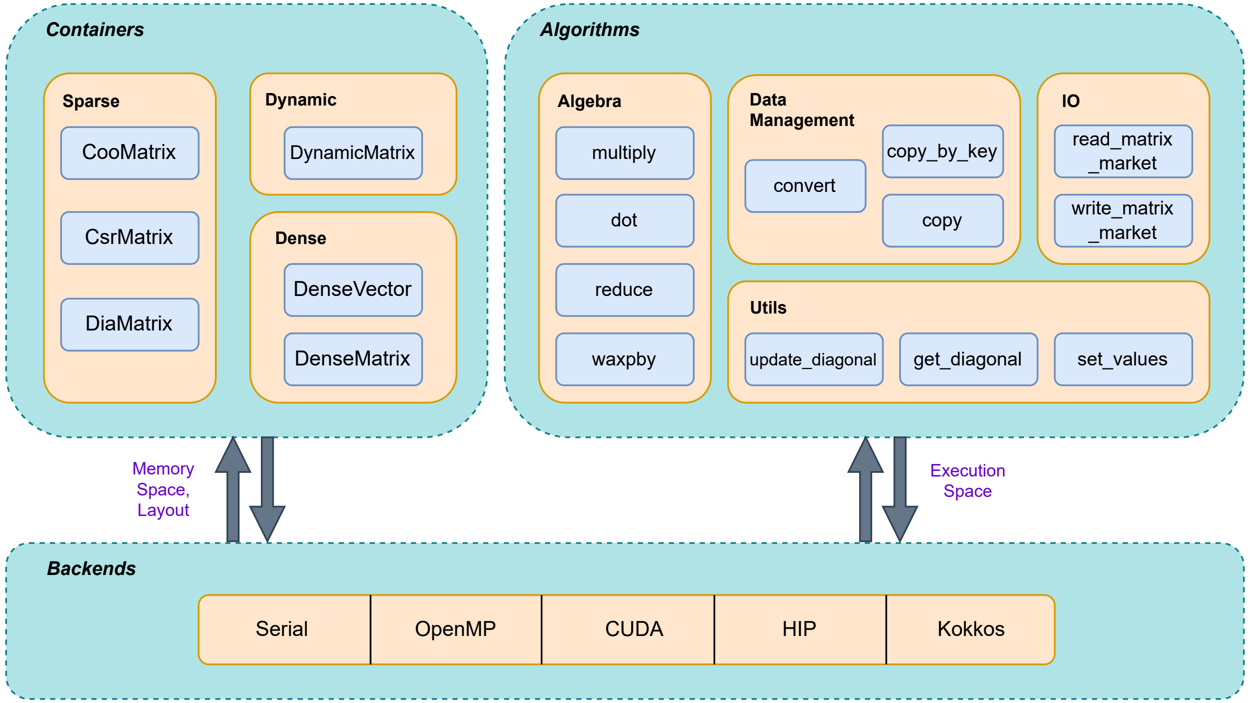
val

1	2	11
3	4	*
5	6	7
8	*	*
9	10	*

ELL Representation

Morpheus: A Library for Dynamic Sparse Matrices

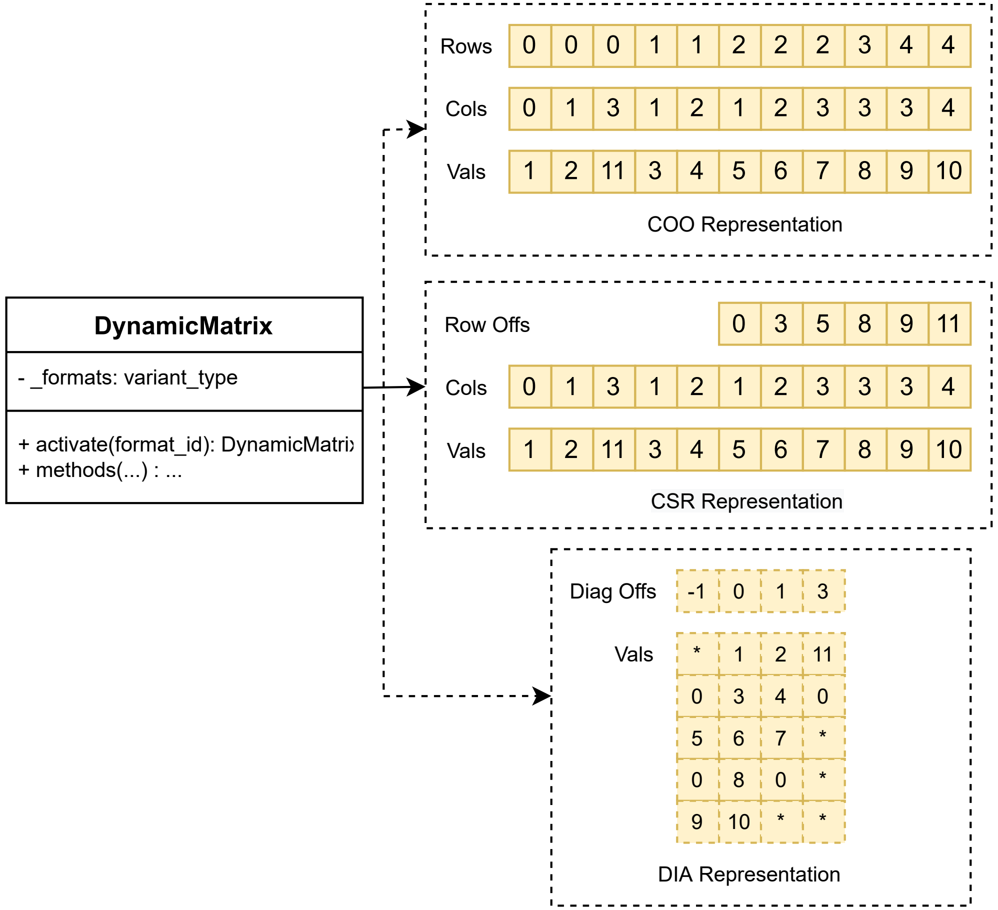
- Templated C++ library
- Functional Design
  - Containers & Algorithms
- Data Management
- Support for Heterogeneous Platforms
  - Host-Device Model
  - Mirroring
- Efficient dynamic switching
- Continuous addition of new formats and backends
  - Increased life-time of software



Link to *Morpheus*: https://github.com/morpheus-org/morpheus

DynamicMatrix Container

- Composition of all the available formats
- Type safe union (*std::variant*)
- All formats are known *apriori*
- Dispatch at run-time examining its active state
  - Low latency & run-time overheads
- Abstract matrix representation
  - Encapsulates internal implementation of each format
  - Single interface for users to use
- Transparent format switching through:
  - *activate()* member function
  - Convert routine (in-place)



Link to *Morpheus*: https://github.com/morpheus-org/morpheus

Integrating Morpheus into Applications

1. Converting user-defined data structures:
   - Convert to containers supported by Morpheus
   - Containers can also be “unmanaged” - *aliasing*
   - Sparse containers only constructed through element-wise conversion
2. Enabling GPU support:
   - No automatic data transfers between spaces
   - Containers either used for general housekeeping or in an algorithm
   - User must handle the data transfers between device containers and mirrors
3. Enabling Dynamic switching:
   - Convert Morpheus Sparse Container to *DynamicMatrix*
   - Both containers share same interface – *No Further changes* are required

Morpheus-enabled HPCG

1. Port *Vector* data structure
   - Morpheus *DenseVector* aliases HPCG Vector – No data management *yet*!
   - Morpheus-enabled *DOT, WAXPBY* operations
2. Port *SparseMatrix* data structure
   - Convert between HPCG CSR Variant to Morpheus *CsrMatrix* container
   - Morpheus-enabled *SpMV*
3. Enable GPU Backend:
   - Data-management of *ExchangeHalo* in SpMV
   - Morpheus-enabled *ZeroVector* and *Copy*
4. Enable Dynamic Switching:
   - Convert Morpheus *CsrMatrix* to *DynamicMatrix*

HPCG solves the Poisson differential equation:
- *on a regular 3D grid*
- *discretized with a 27-point stencil*

using:
- *Preconditioned Conjugate Gradient (PCG) algorithm*
- *Symmetric Gauss-Seidel as a preconditioner*

Link to *Morpheus-enabled* HPCG: https://github.com/morpheus-org/morpheus-hpcg

epcc

Experiment Setup Description

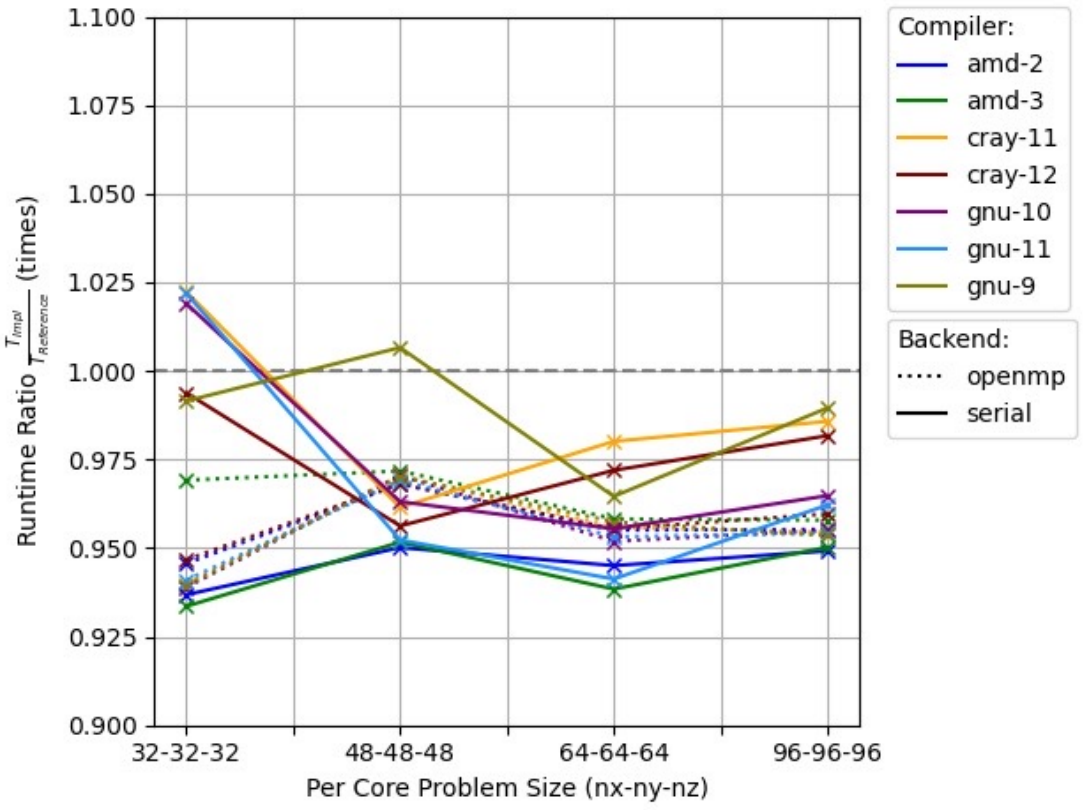
1. Overhead Comparison from the adoption of *DynamicMatrix*
   - Comparison of the original HPCG w.r.t. the Morpheus-enabled HPCG
2. Single-node performance of available formats in Morpheus-enabled HPCG
   - Over several problem sizes, compilers & architectures
3. Multi-node performance of Morpheus-enabled HPCG
   - Split matrix to *local* and *remote* parts
   - Over several architectures
   - Versions:
     - Morpheus (Local matrix changes format on each process)
     - Ghost (Remote matrix changes format on each process)
     - Multi-format (Both change format on each process)
     - Original HPCG (Reference)

PLATFORM	CIRRUS (GPU Node)	CIRRUS (CPU NODE)	ARCHER2
CPU	INTEL XEON GOLD 6248 (X2)	INTEL XEON E5-2695 (X2)	AMD EPYC 7742 (X2)
GPU	NVIDIA TESLA V100 SXM2-16GB (X4)	N/A	N/A

Node configurations for the systems used in the experiments.

Overhead Comparison

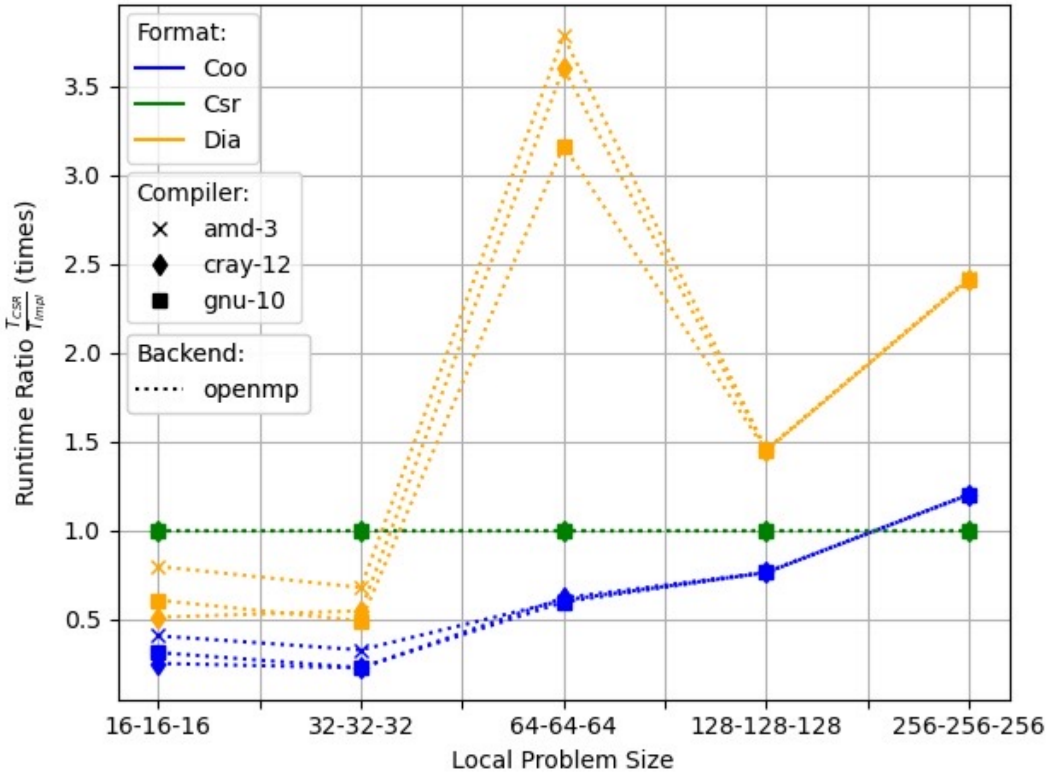
- Run-time of *DynamicMatrix* (switched at CSR)
  - w.r.t. Original HPCG
- OpenMP backend uses 16 cores (1 chiplet)
- Overall negligible overheads
- Overheads reduced as problem size grows
- Similar behavior for *Intel* hardware



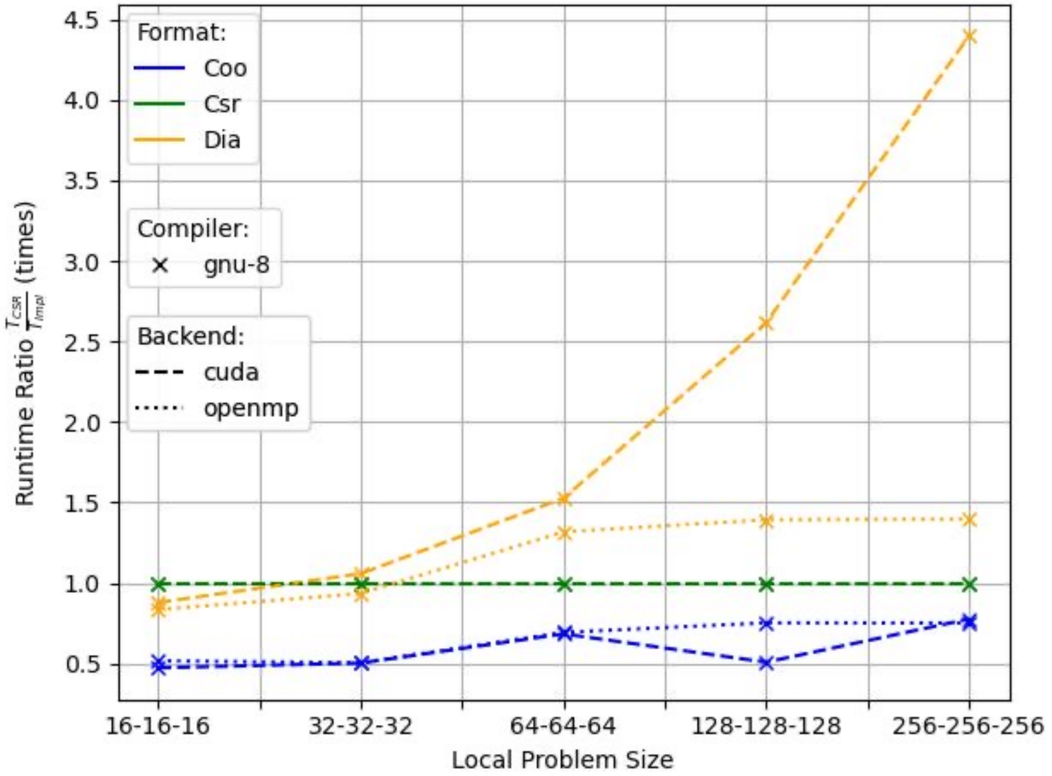
Archer2

Run-time Ratio = $\frac{SpMV\ run{-}time\ of\ Morpheus{-}enabled\ HPCG\ (CSR)}{SpMV\ run{-}time\ of\ original\ HPCG}$ *times*

Single-node Performance



Archer2



Cirrus

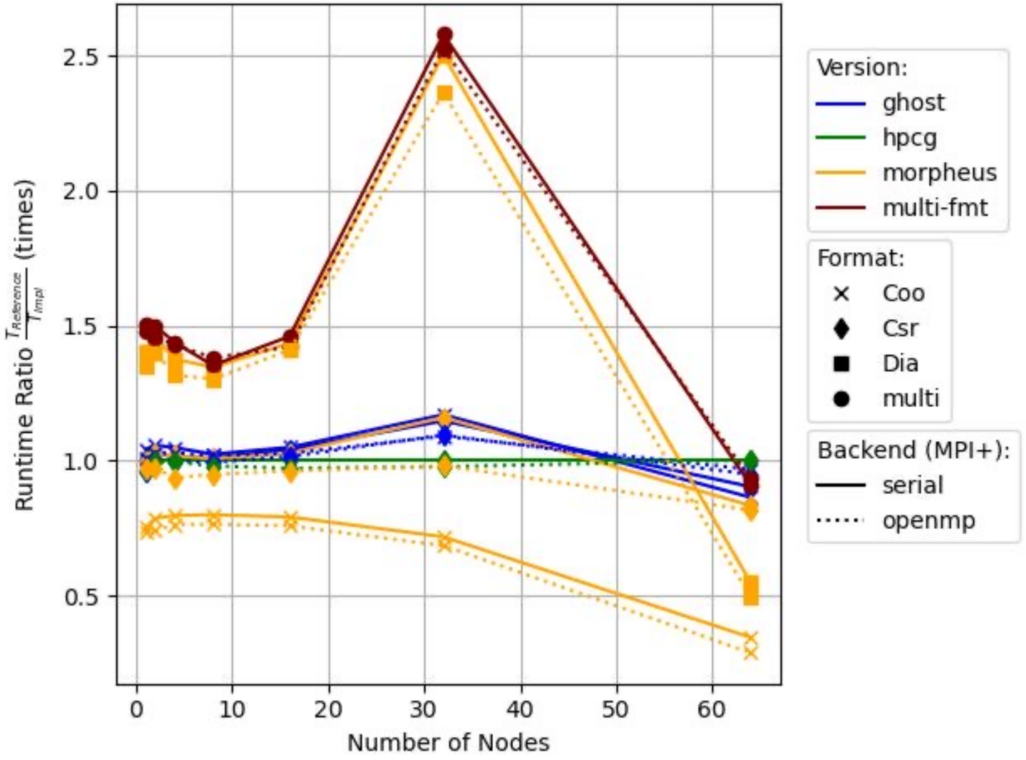
$$\text{Run-time Ratio} = \frac{SpMV\ run{-}time\ of\ DynamicMatrix\ (CSR)}{SpMV\ run{-}time\ of\ DynamicMatrix\ (COO, CSR\ or\ DIA)}$$ *times (higher is better)*

Multi-node Performance – Multi-format Version

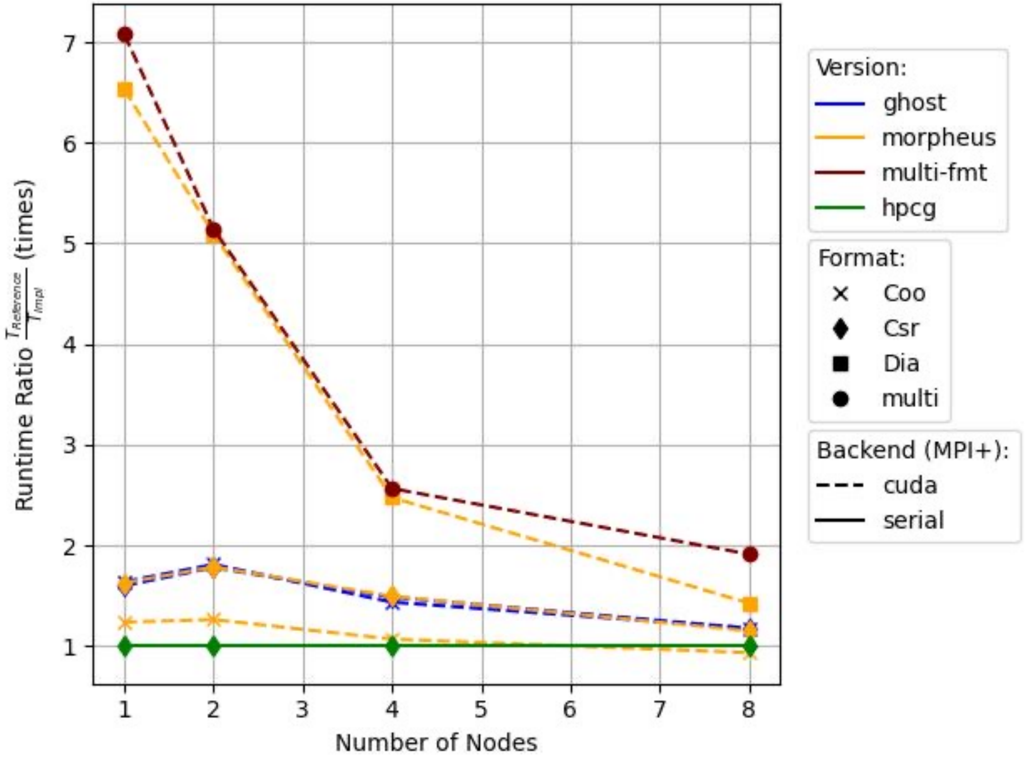
- For each local & remote part on each process we:
  - Perform profiling runs of HPCG
  - Measure per-process timings for each format
  - Generate an input file with optimal format configuration
- Use generated file to switch format on each process
  - Achieving the optimum format per-process and per matrix part (local & remote)
- Global Problem Size for Strong Scaling:
  - ARCHER2 – 512x512x256 (per Process size on Weak Scaling)
  - Cirrus – 384x256x128 (per Process size on Weak Scaling)

Multi-node Performance – Strong Scaling

Versions:
- Morpheus (Local matrix changes format on each process)
- Ghost (Remote matrix changes format on each process)
- Multi-format (Both change format on each process)
- Original HPCG (Reference)



Archer2



Cirrus

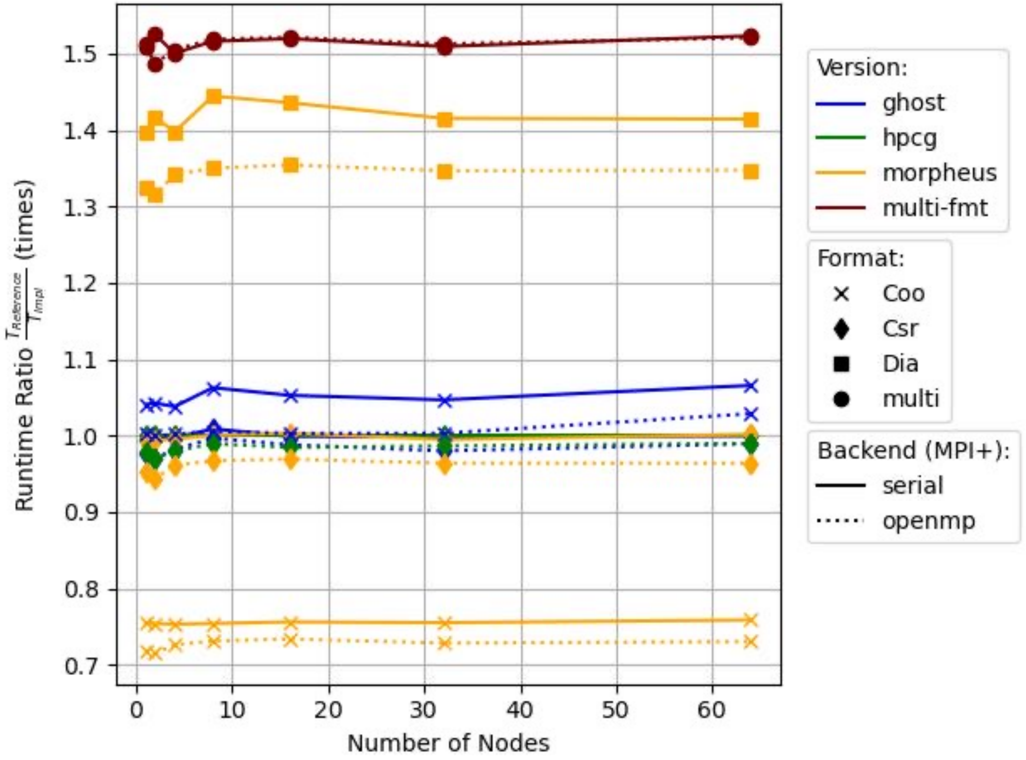
$$\text{Run-time Ratio} = \frac{SpMV\ run\!-\!time\ of\ Reference\ HPCG}{SpMV\ run\!-\!time\ of\ Version}$$ *times (higher is better)*

epcc

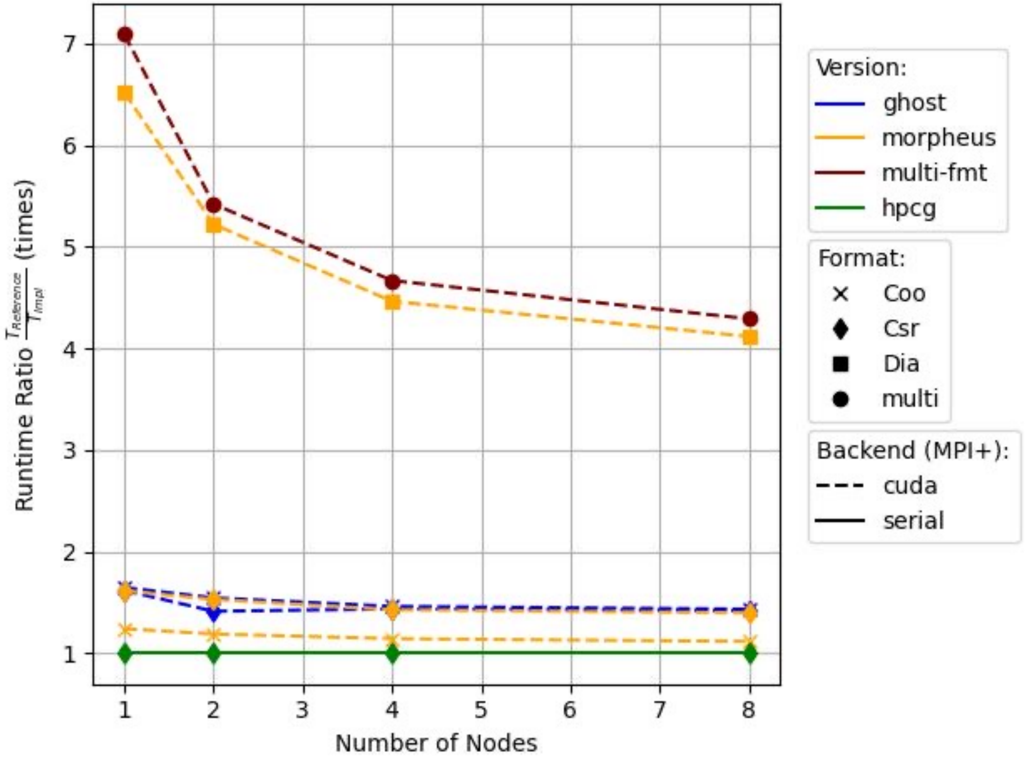
Multi-node Performance – Weak Scaling

Versions:

- Morpheus (Local matrix changes format on each process)
- Ghost (Remote matrix changes format on each process)
- Multi-format (Both change format on each process)
- Original HPCG (Reference)



Archer2



Cirrus

Run-time Ratio = $\frac{SpMV\ run-time\ of\ Reference\ HPCG}{SpMV\ run-time\ of\ Version}$ *times (higher is better)*

Conclusions

- No format can perform optimally across different operations, sparsity patterns and target architectures.
- Dynamically changing the underlying data structure offers a range of optimization opportunities.
- One of them is using a different format per process in distributed setting.
- By porting Morpheus in applications users can now:
  - Target new architectures (GPUs)
  - Optimize their code through format switching without further code modifications.
  - Increase software lifetime as new formats and architectures are added.
- Performance of SpMV kernel is improved up to:
  - *2.5x* on CPUs
  - *7x* on GPUs

through runtime selection of the best format on each MPI process

Link to *Morpheus*: https://github.com/morpheus-org/morpheus

epcc