Composition of Algorithmic Building Blocks in Template Task Graphs

Thomas Herault, **Joseph Schuchart** (schuchart@icl.utk.edu), George Bosilca
The University of Tennessee, Knoxville
Edward F. Valeev
Virginia Polytechnic Institute and State University

PAW/ATM'22, held in conjunction with SC'22

Task Graphs and their Composition

Composition is essential for scalable software development

Task Graphs and their Composition

Composition is essential for scalable software development

We know how to compose functions and libraries
(through well-defined interfaces and black-box functions)

Task Graphs and their Composition

Composition is essential for scalable software development

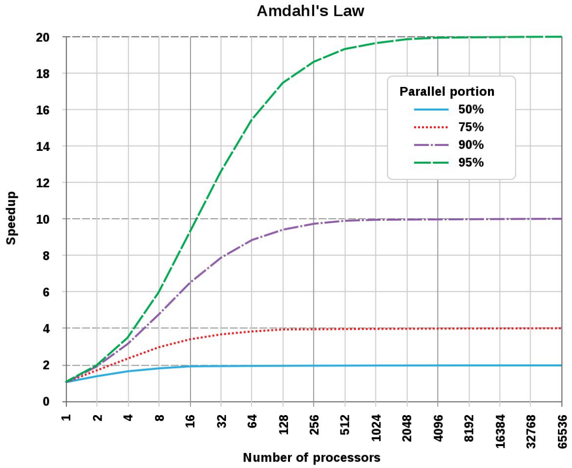
We know how to compose functions and libraries
(through well-defined interfaces and black-box functions)

Do we know how to compose task graphs?

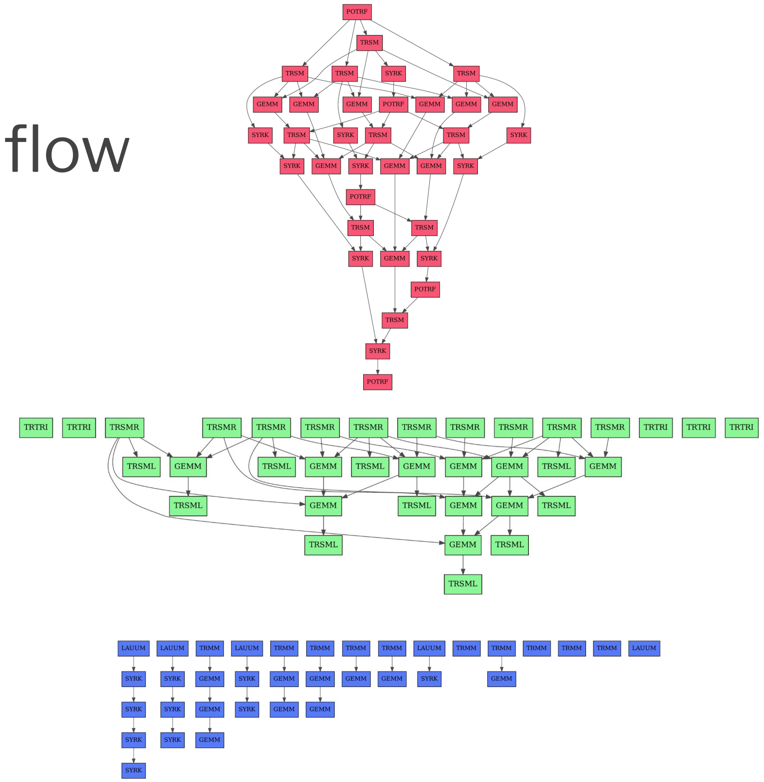
Why Task Graph Composition

- Assume N consecutive functions implemented using the same task programming model
  - N forks and joins without composition
  - Write-back to data structures instead of direct flow
  - Gene Amdahl says that's bad

https://upload.wikimedia.org/wikipedia/commons/1/1a/Gene_Amdahl_on_a_classic_grey_Ferguson_tractor_at_Amdahl.JPG

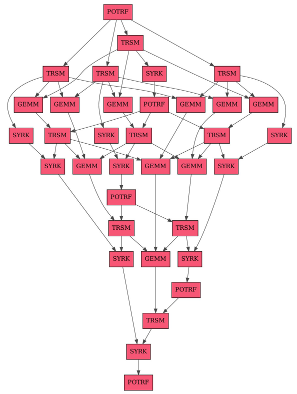


https://en.wikipedia.org/wiki/Amdahl%27s_law#/media/File:AmdahlsLaw.svg

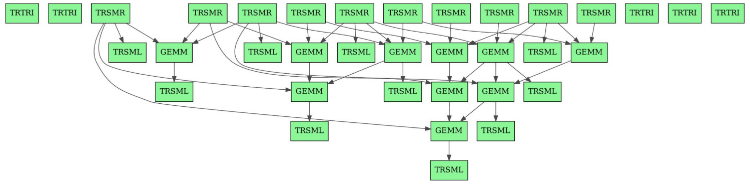


Example: Cholesky Matrix Inversion

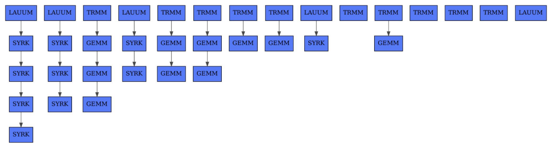
- Cholesky Factorization (POTRF) followed by matrix inversion
    - Given A, compute A^{-1}
    - A: Hermitian positive-definite matrix



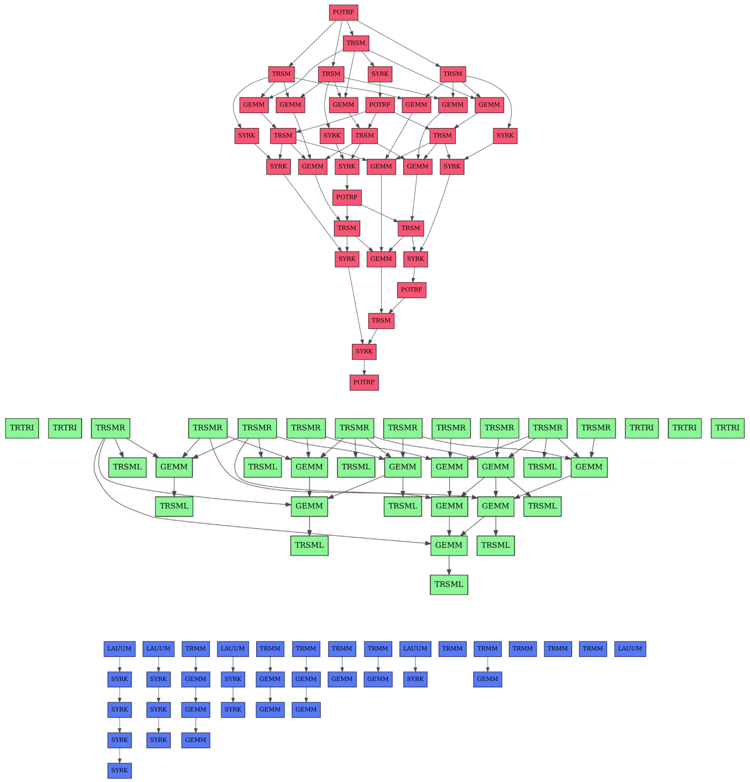
- Inversion: Given L from POTRF
    - Compute L^{-1} from L (TRTRI)
    - Compute $A^{-1} = (L^{-1})^T L^{-1}$ (LAUUM)
    - POTRI = TRTRI ⊕ LAUUM

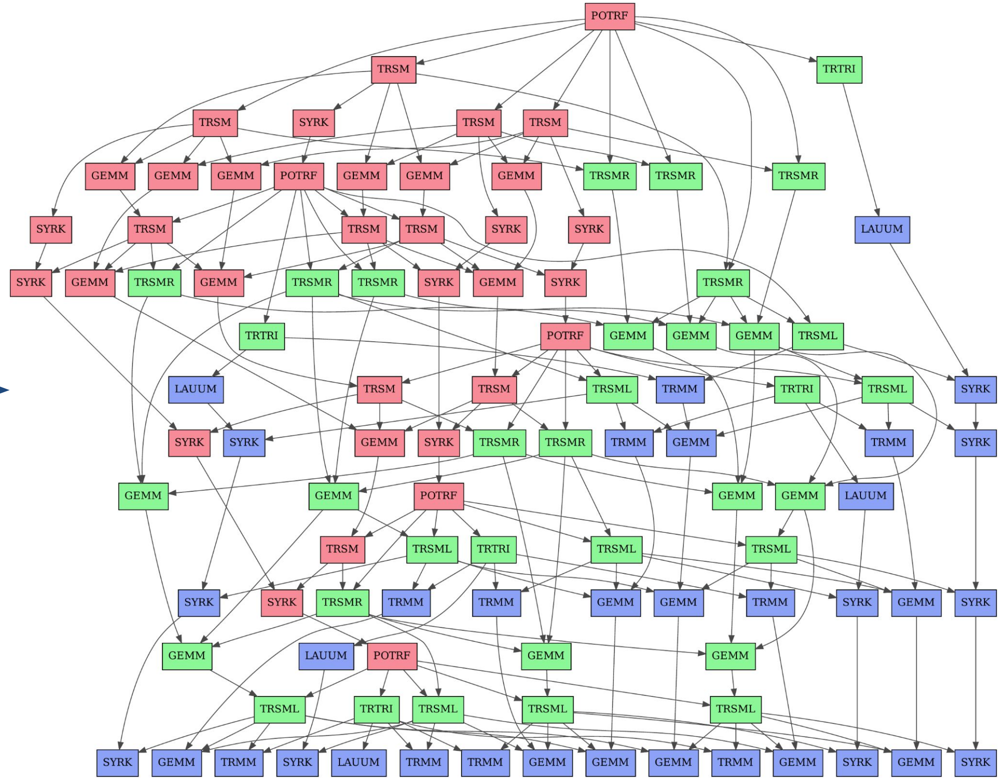


- POINV = POTRF ⊕ POTRI
  = POTRF ⊕ TRTRI ⊕ LAUUM



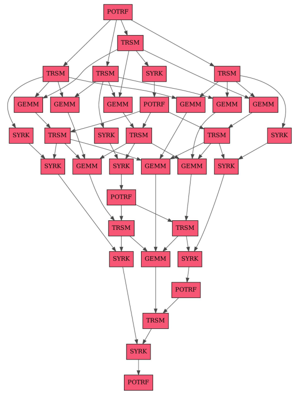
Task Graph Composition at Work

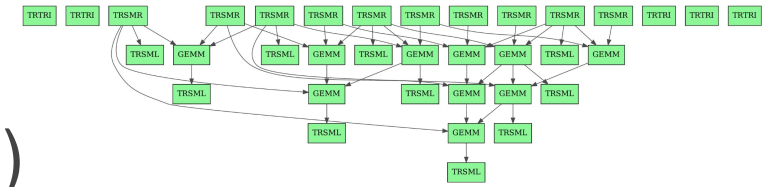


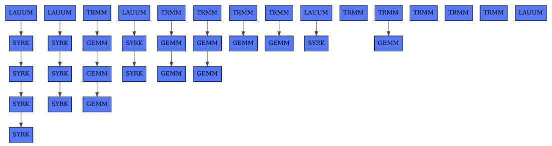


Existing Approaches to Task Graph Composition

- **Dependency**-based approaches
  - OpenMP, OmpSs, StarPU, PaRSEC DTD, ...
  - Sequential discovery of tasks and their dependencies
  - Limited discovery windows
  - Limited scalability
- **Handle**-based approaches (HPX)
  - Composition through future passing
  - One future per element
  - Significant efforts for LA required
- **Parameterized Task Graphs** (PaRSEC PTG)
  - Flow within a graph
  - Operates exclusively on data collections
  - Missing: external interfaces

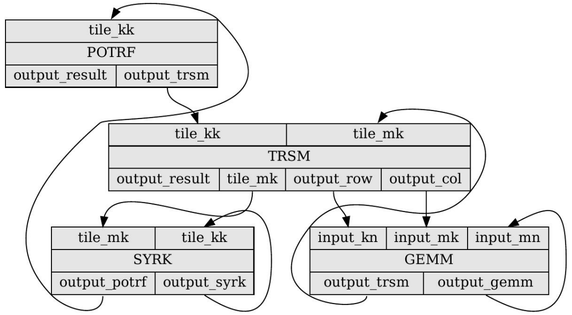


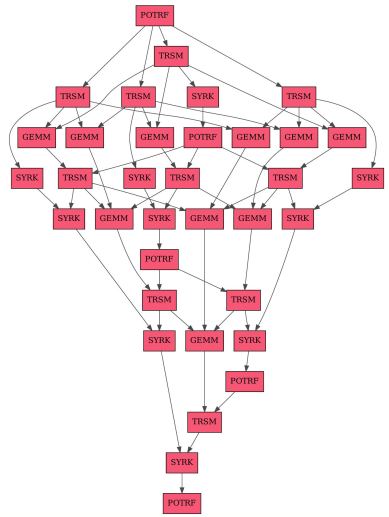




Introducing: Template Task Graphs

- Algorithms expressed as **abstract graphs**
  - All possible edges in task graph declared *a priori*
  - Template Tasks (TT) with input and output terminals
  - **Tasks are instances** of TTs identified by IDs
  - **Edges** represent **sets of values** flowing between tasks
  - Data movement through the graph handled by TTG
- Template Task Graph unrolled during execution
  - Fully distributed graph discovery



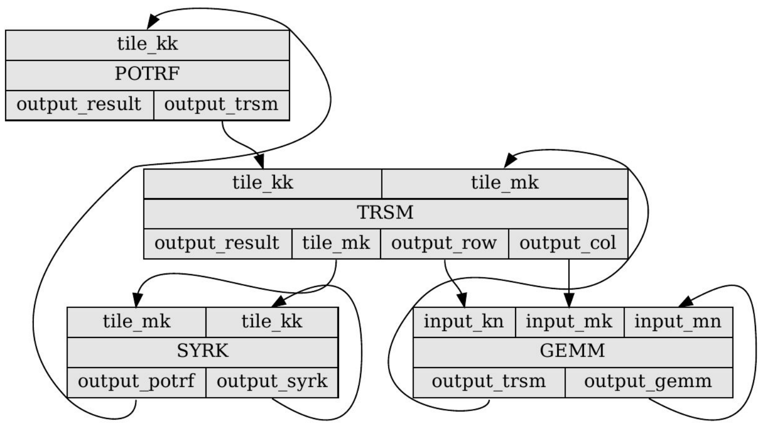


G. Bosilca, R. J. Harrison, T. Herault, M. M. Javanmard, P. Nookala and E. F. Valeev, "The Template Task Graph (TTG) - an emerging practical dataflow programming paradigm for scientific simulation at extreme scale," *2020 IEEE/ACM Fifth International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*, 2020.

Edges in Template Task Graphs

- **Edges** connect template tasks through **terminals**
- Tasks decide on which output terminal(s) to send data

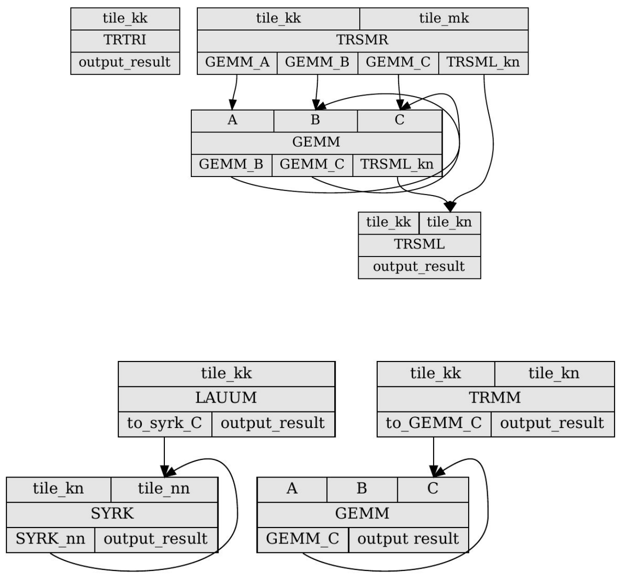
- Edges connect
  - A single output terminal (its input)
  - Multiple input terminals (its output)



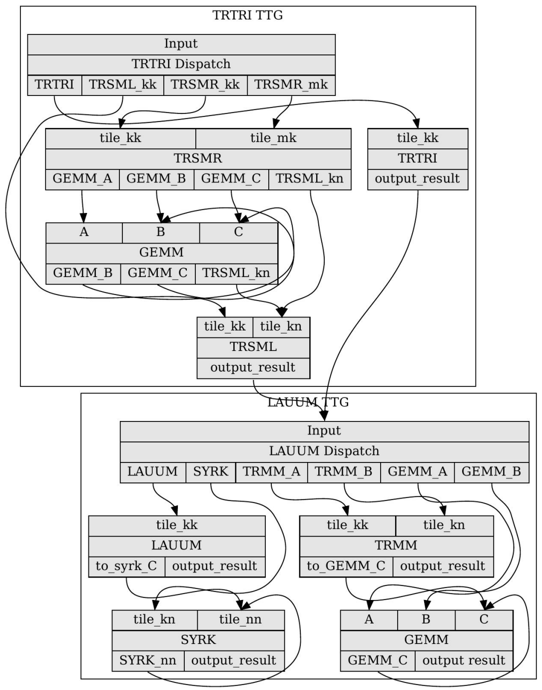
- Can we use Edges to connect Task Graphs?*

Connecting Graphs: Edges as Composition Devices

- Introduce **Dispatch Tasks** to Task Graphs
  - Dispatch incoming data to internal tasks
  - Internal edges and tasks not visible to outside
  - Single output edge, to connect to successors

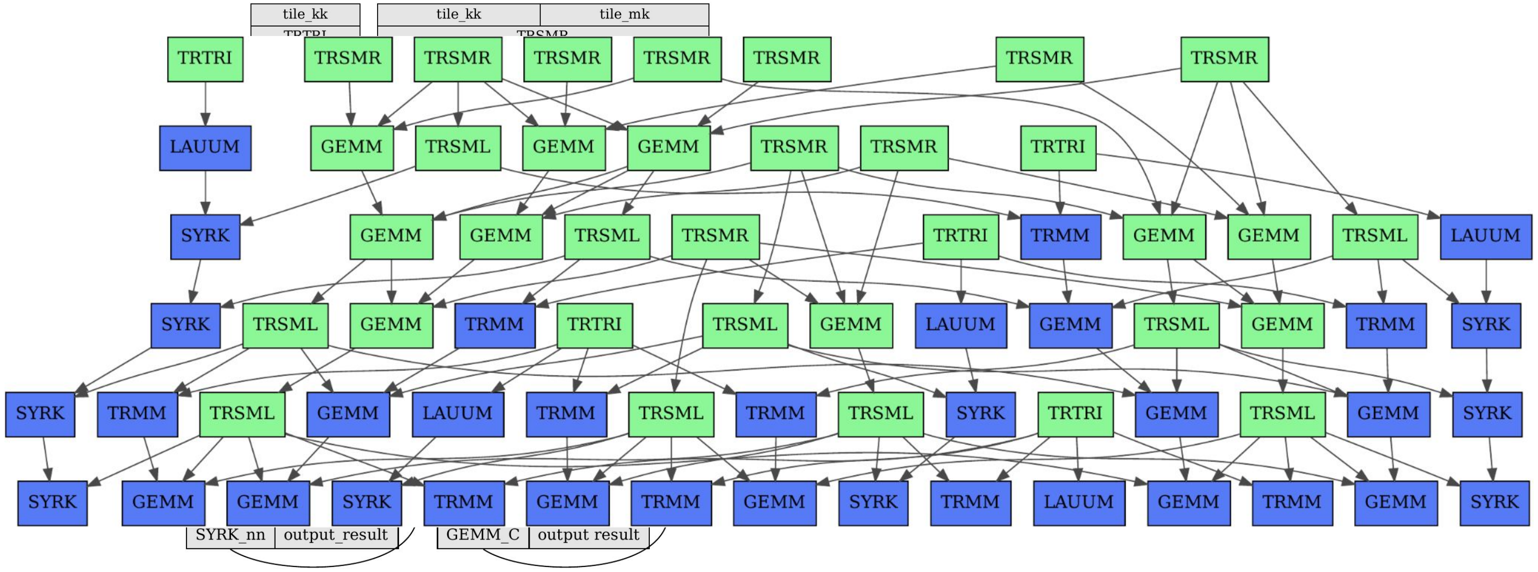


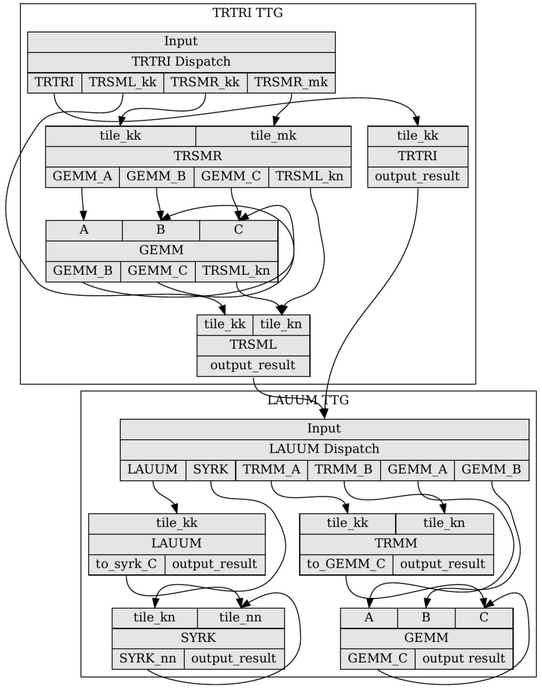
Composition



Connecting Graphs: Edges as Composition Devices

- Introduce **Dispatch Tasks** to Task Graphs
  - Dispatch incoming data to internal tasks
  - Internal edges and tasks not visible to outside
  - Single output edge, to connect to successors





Task Graphs as Black Box Functions

- **Edges**: well-defined interface between graphs
- Internal structure of a task-graph irrelevant to caller

- Composing TRTRI & LAUUM into POTRI

```cpp
using namespace std;
auto make_potri_ttg(ttg::Edge<Key2, MatrixTile>&input,
                    ttg::Edge<Key2, MatrixTile>&output)
{
  ttg::Edge<Key2, MatrixTile>
                    trtri_to_lauum("trtri_to_lauum");

  auto ttg_trtri = make_trtri_ttg(input, trtri_to_lauum);
  auto ttg_lauum = make_lauum_ttg(trtri_to_lauum, output);

  auto ins  = make_tuple(ttg_trtri->template in<0>());
  auto outs = make_tuple(ttg_lauum->template out<0>());
  vector<unique_ptr<ttg::TTBase>> ops(2);
  ops[0] = std::move(ttg_trtri);
  ops[1] = std::move(ttg_lauum);

  return ttg::make_ttg(std::move(ops),
                       ins, outs, "POTRI TTG");
}
```

Input Edge

Output Edge

Internal Edge

Internal Subgraphs

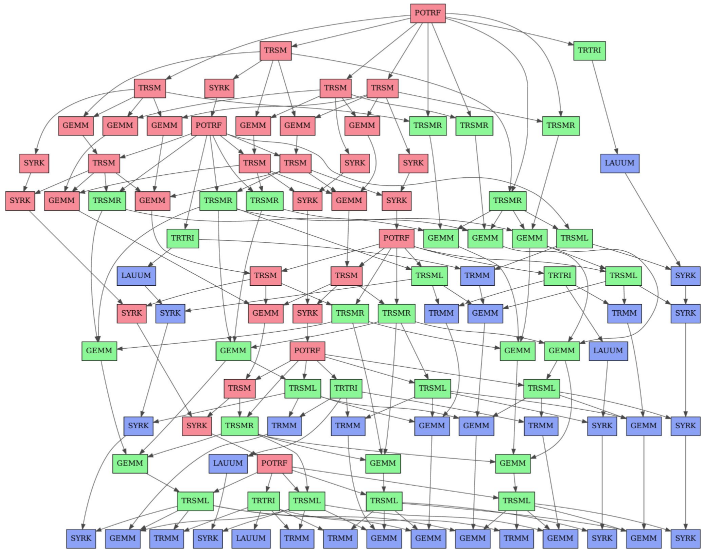
New Template Task Graph

Benefits of Task Graph Composition

- Minimized serial application parts

- Application-level **depth-first execution** (memory reuse)
  - Cache reuse
  - Device data transfer minimization (for out-of-core computation)

- **Flexibility**: reusing intermediate results becomes trivial
  - Example: tiles of a Cholesky factorized matrix
    - As input for POTRI
    - Used together with POTRI results
  - TTG will manage tile copies

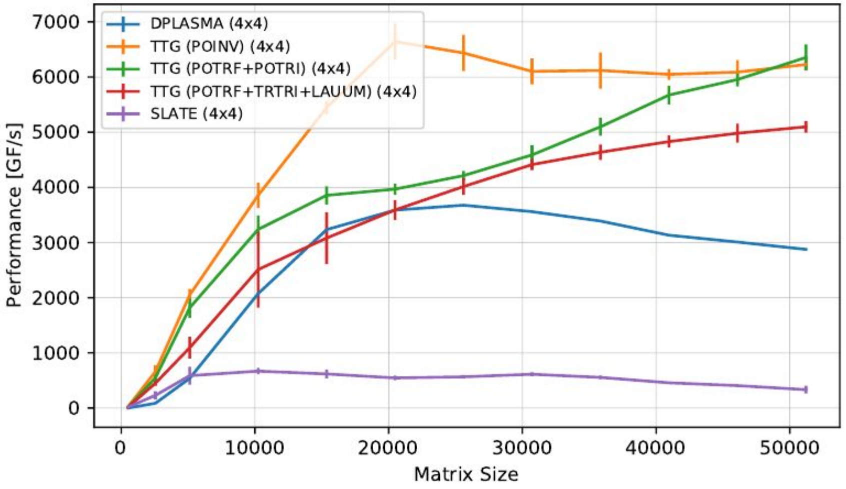


Experimental Evaluation: Test System Setup

- All experiments executed on Hawk
  - Installed at HLRS in Stuttgart, Germany
  - 2x64core AMD EPYC nodes, 2.2 GHz
  - Mellanox ConnectX-6
- Open MPI 4.1.0
- GCC 10.2.0

POINV Composition

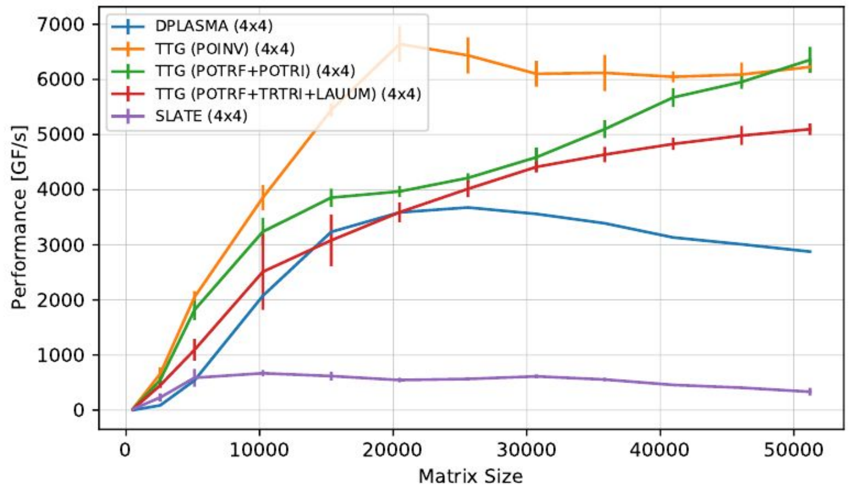
- 16 nodes on Hawk, 64 threads each
- Full composition beneficial for small tile sizes
  - Fine-grain composition helps hide communication latency
  - Beats both DPLASMA (based on PaRSEC PTG) and SLATE



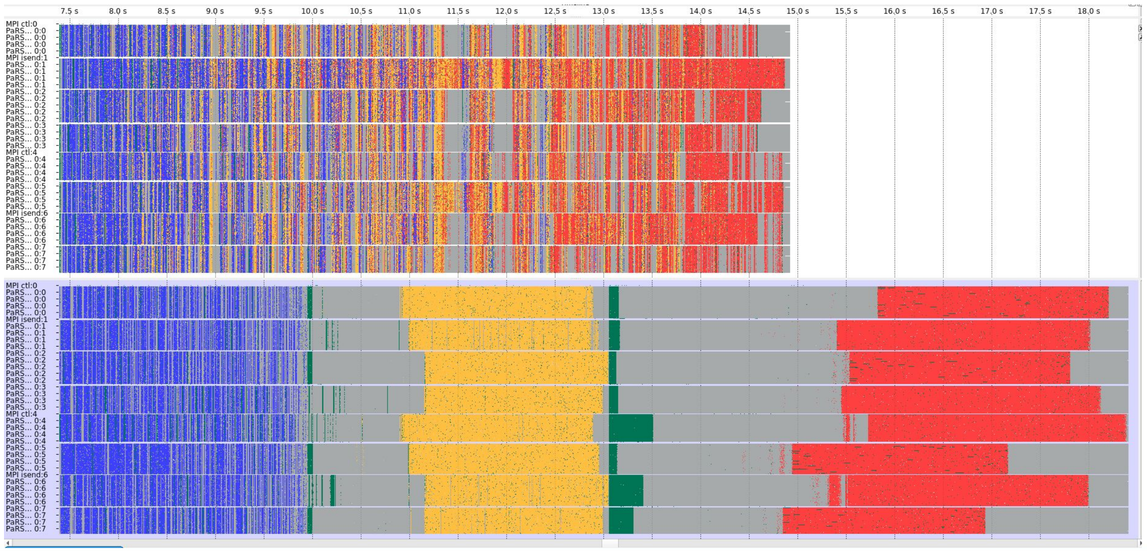
(a) Tile size 128.

POINV Composition

- 16 nodes on Hawk, 64 threads each
- Full composition beneficial for small tile sizes
  - Fine-grain composition helps hide communication latency
  - Beats both DPLASMA (based on PaRSEC PTG) and SLATE

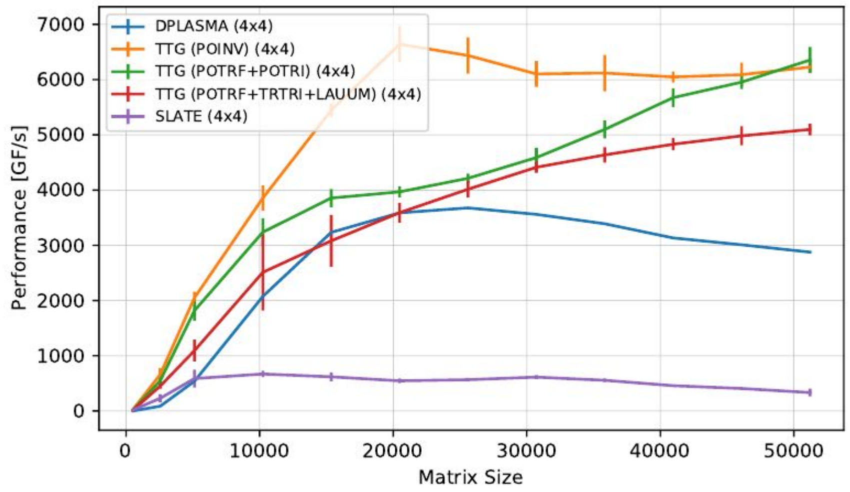


(a) Tile size 128.

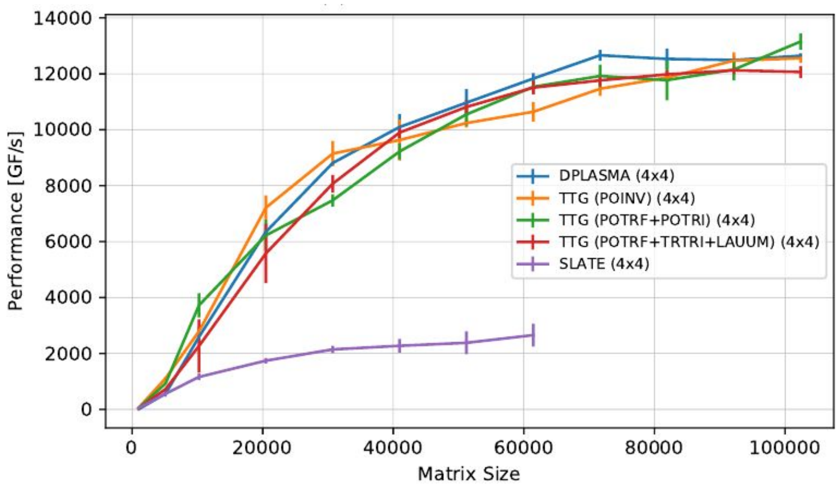


POINV Composition

- 16 nodes on Hawk, 64 threads each
- Full composition beneficial for small tile sizes
  - Fine-grain composition helps hide communication latency
  - Beats both DPLASMA (based on PaRSEC PTG) and SLATE



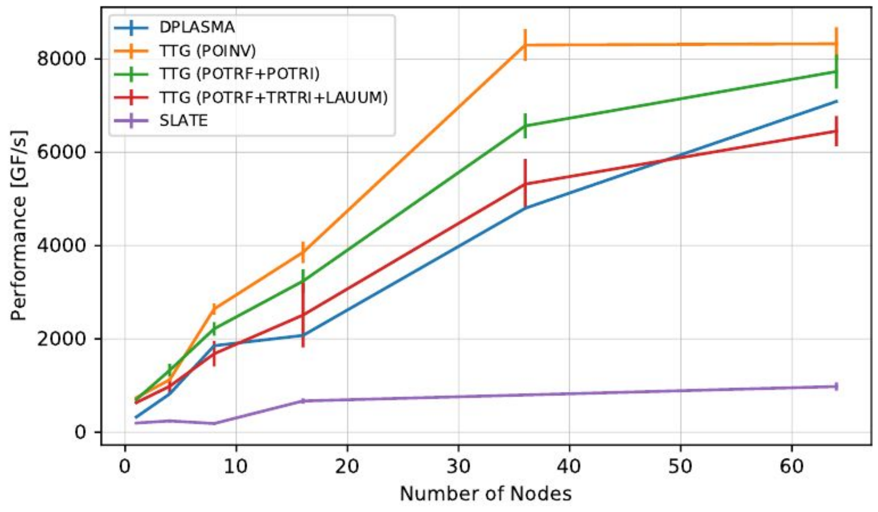
(a) Tile size 128.



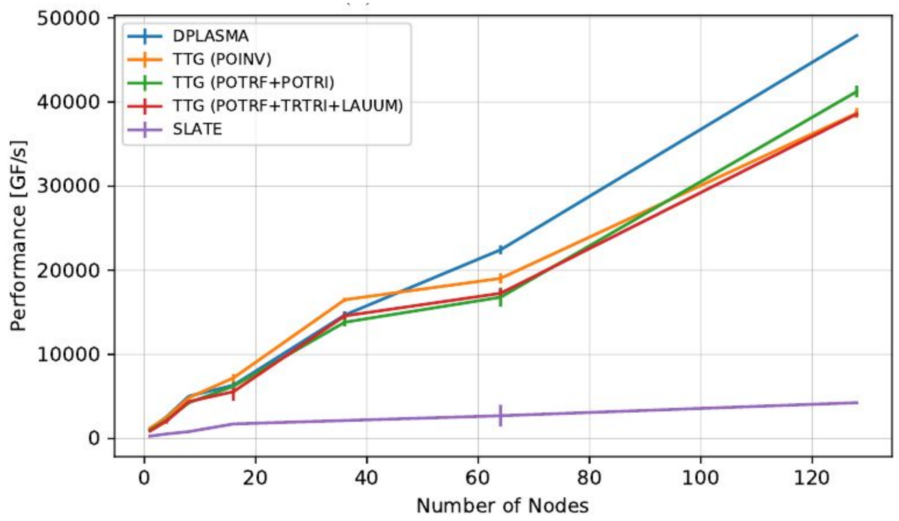
(b) Tile size 256.

POINV Composition

- Scaling from 1-64 nodes on Hawk
  - Full composition beneficial for smaller tasks



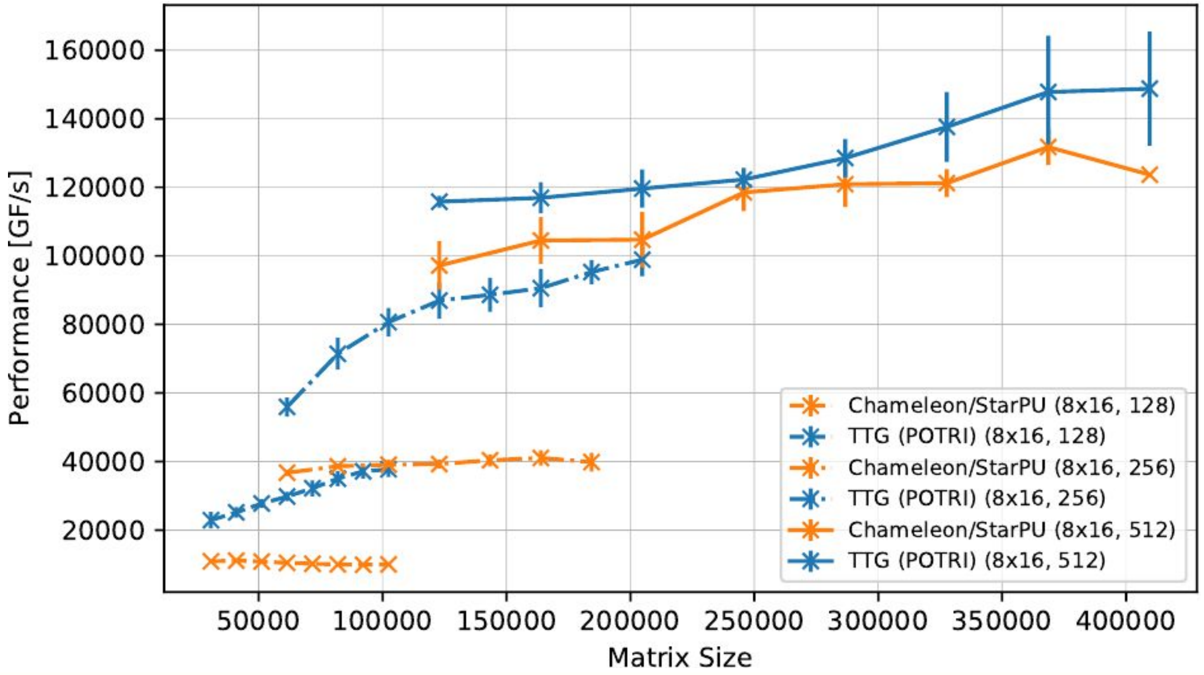
(a) Tile size 128.



(b) Tile size 256.

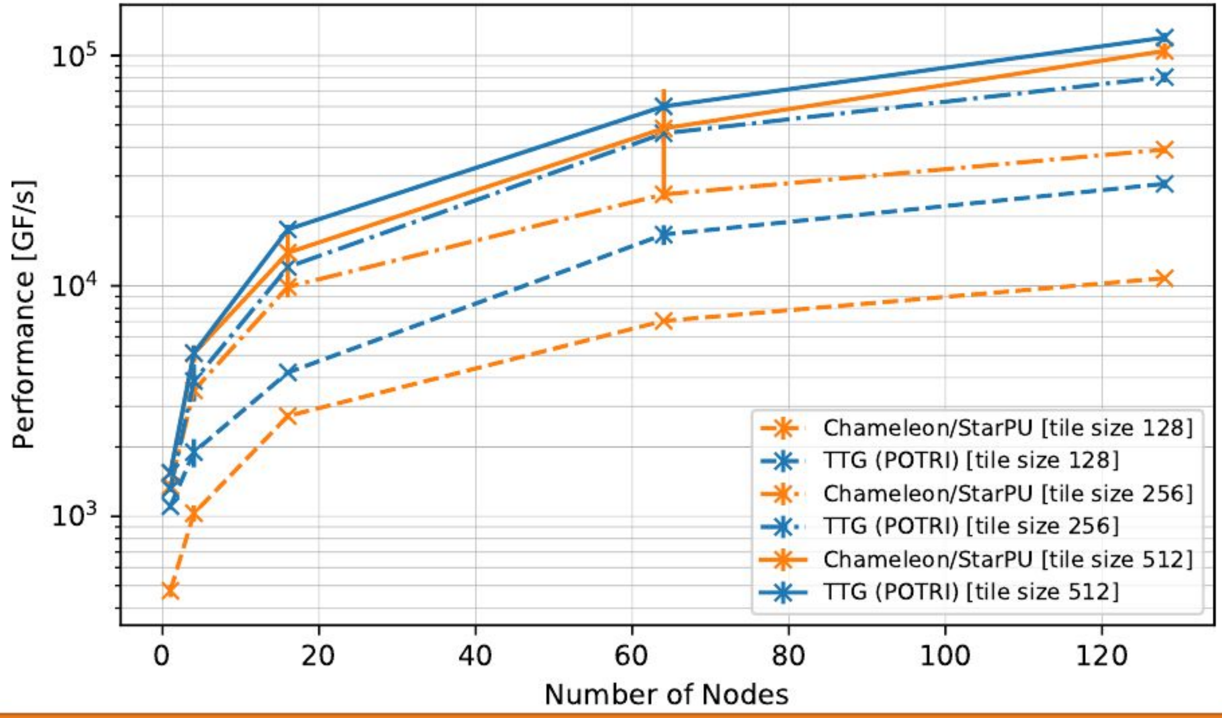
POTRI: Comparison with Chameleon

- 128 nodes on Hawk
- Chameleon (v1.1.0, using StarPU 1.3.9)
- POTRI : TRTRI ⊕ LAUUM
- TTG performance benefits
  - Depth-first execution
  - Parallel distributed task discovery



POTRI: Comparison with Chameleon

- 1 - 128 nodes on Hawk
- Chameleon (v1.1.0, using StarPU 1.3.9)
- POTRI : TRTRI ⊕ LAUUM
- TTG performance benefits
  - Depth-first execution
  - Parallel distributed task discovery

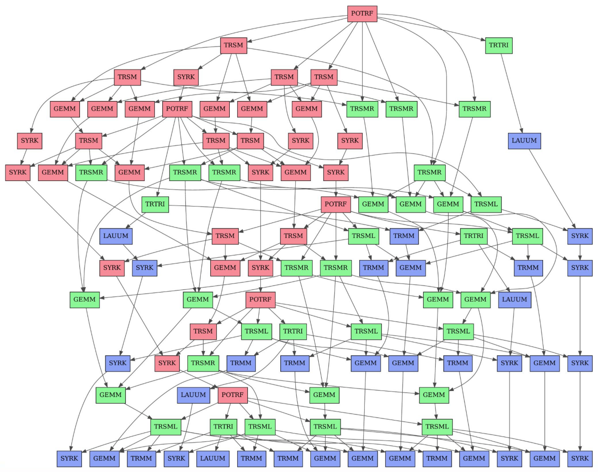


Conclusions

- Task graph composition helps hide communication latencies

- **Edges represent sets of future values**
  - Distribute values to multiple subscribers (input terminals)
  - Provide a clean interface to encapsulate black-box functionality
  - Coupling of graphs through single input, single output Edge

- Full-scale application composition without breaking abstraction barriers



Future Work

- GPU support (beyond Unified Memory, 2023)
- Inverse data flow: pulling data from the bottom of the task graph
- Coupling of different task-based programming models
  - Example: TiledArray & TTG

ICL@SC22

Questions?

UNIVERSITY OF TENNESSEE **BOOTH #613**

@ICL_UTK ON TWITTER

FOLLOW US AT **https://icl.utk.edu/sc22**

Check us out on Github:
github.com/TESSEorg/ttg/

This research was supported partly by NSF awards #1931347 and #1931384, and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. We gratefully acknowledge the provision of computational resources by the High-Performance Computing Center (HLRS) at the University of Stuttgart, Germany.