TTG TEMPLATE TASK GRAPHS

FIND OUT MORE AT https://github.com/TESSEorg/ttg

TEMPLATE TASK GRAPHS

The Templated Task Graph API / DSL has been developed to enable a straightforward expression of the parallelism for algorithms that work on irregular and unbalanced data sets. Combining our experience with MADNESS, TiledArray and PaRSEC, the DSL employs C++ templates to build an internal representation of the Distributed DAG of Tasks.

OVERALL OBJECTIVE

- Provide an intermediate-level expression of data-dependent irregular algorithms while leveraging a powerful micro-task runtime to manage dependencies, scheduling, and data motion within the data flow.
- Encourage programs that avoid non-essential barriers and intermediates, express available concurrency without drowning the developer in detail, and reap most benefits of fusion within a more general framework
- An extensible, robust and scalable directed acyclic graph (DAG) execution model supported by an intelligent and dynamic runtime that can adapt to changing requirements presented by the evolving numerical theories and HPC platforms.

TTG KEY CONCEPTS

1 Parameterize each task that will execute the operation by a key or index (e.g. a loop index making a separate task for each iteration; the label of each node in a tree being traversed; a pair of indices labelling a matrix sub-block).

2 Avoid describing / observing the entire task graph at once (avoid memory clogging).

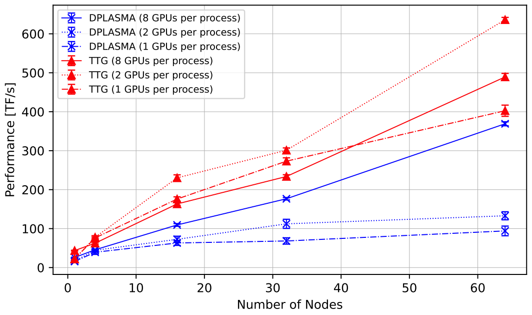
3 Fully parallelized distributed task graph discovery at scale.

4 Data labeled by a key to match with consuming task.

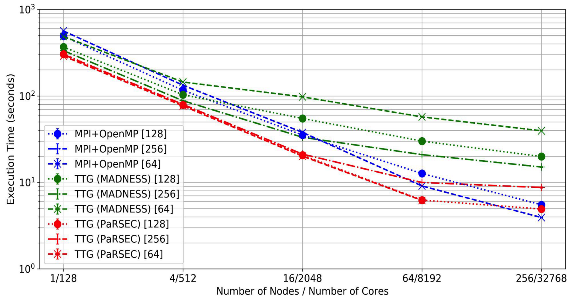
5 Through each output, a task can send data to a specific successor (identified by its key), or broadcast to multiple successors (keys).

PERFORMANCE

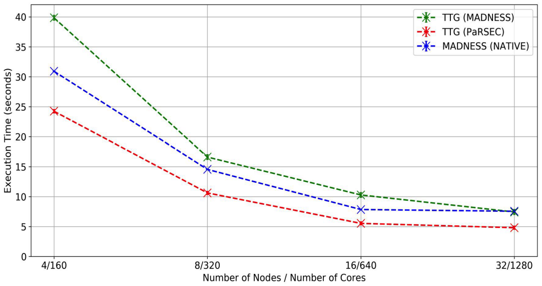
Dense Linear Algebra:
Cholesky Factorization (strong scaling)



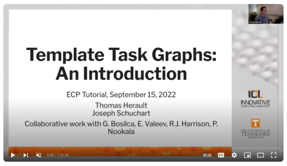
Dense Stencil-Like operation:
Floyd-Warshall (strong scaling)



Irregular, Data-Dependent Operation:
Multi-Resolution Analysis (strong scaling)



TUTORIAL



Template Task Graphs:
An Introduction

ECP Tutorial, September 15, 2022

Thomas Herault
Joseph Schuchart

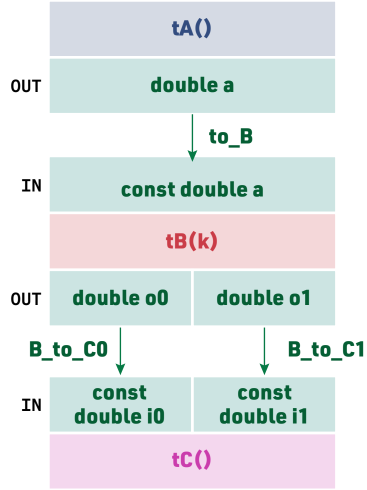
Collaborative work with G. Bosilca, E. Valeev, R.J. Harrison, P. Nookala

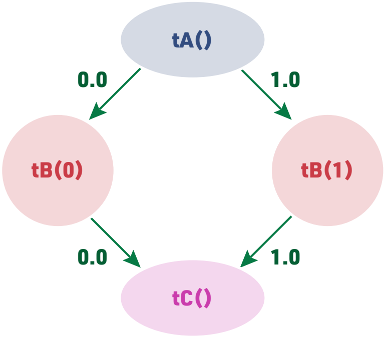
Watch on YouTube:
https://youtu.be/BOavGaSviqQ

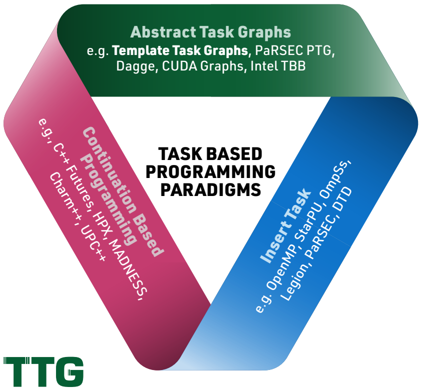
TTG TEMPLATE TASK GRAPHS

FIND OUT MORE AT **https://github.com/TESSEorg/ttg**

```
ttg::Edge<int, double> to_B;
ttg::Edge<void, double> B_to_C0;
ttg::Edge<void, double> B_to_C1;

auto ta = ttg::make_tt<void>(
 []() {
    ttg::send<0>(0, 0.0);
    ttg::send<0>(1, 1.0);
 },
 ttg::edges(), ttg::edges(to_B),
 "tA");

auto tb = ttg::make_tt(
 [=](const int &k,
    const double &a) {
     if(0 == k) ttg::sendk<0>(a);
     if(1 == k) ttg::sendk<1>(a);
 },
 ttg::edges(to_B),
 ttg::edges(B_to_C0, B_to_C1),
 "tB");

auto tc = ttg::make_tt<void>(
 [](const double &i0,
    const double &i1) {
     ttg::print(
       "This is task C()",
       " it received values ",
       i0," and ", i1);
 },
 ttg::edges(B_to_C0, B_to_C1),
 ttg::edges(),
 "tC");

ttg::make_graph_executable(ta);
if(ta->get_world().rank() == 0) {
   ta->invoke();
}
ttg::execute();
ttg::fence(tb->get_world());
```







Abstract Task Graphs
e.g. Template Task Graphs, PaRSEC PTG, Dagge, CUDA Graphs, Intel TBB
Continuation Based Programming
e.g., C++ Futures, HPX, MADNESS, Charm++, UPC++
TASK BASED PROGRAMMING PARADIGMS
Insert Task
e.g. OpenMP, StarPU, OmpSs, Legion, PaRSEC, DTD

TTG

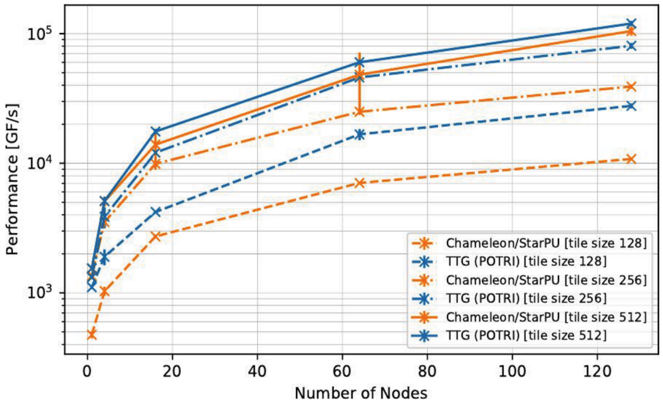
AN ABSTRACT TASK GRAPHS PARADIGM

- Abstract Task Graphs rely on a representation of the folded graph of task classes which allows a fully distributed and scalable discovery of the DAG of tasks at runtime
- Continuations require to manage a collection of futures whose size is linear in the number of edges in the entire DAG of tasks.
- Insert Task builds the DAG of tasks transparently by deducing it from the apparent order of data access in a sequential discovery of tasks, linear in the problem size.

HIGHLIGHT

Modular Programming in TTG: fine-grain task composition

Inverse Cholesky Factorization: the POTRI operation can be seen as the composition of two parallel operations: TRTRI and LAUUM. Both operations are expressed in a task-based algorithm, over two Task-Based programming paradigms: Chameleon/StarPU uses the Insert Task paradigm, and all tasks of TRTRI need to be discovered before the first task of LAUUM can be discovered and performed. The Abstract Task Graph approach of TTG allows to provide the same level of fine-grain composition, while removing this limitation, as both graphs of (template) tasks are entirely expressed at the beginning, in a problem-size independent way.



IN COLLABORATION WITH

Stony Brook University

VIRGINIA TECH.

SPONSORED BY

NSF National Science Foundation

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE