Nested Task Parallel Workflows for Scientific Applications

“Data movement, rather than computational processing, will be the constrained resource at exascale.”

– Dongarra et al. 2011

Three-stage workflow converting particles into a density image
The comparison of a molecular dynamics simulation with high-energy X-ray microscopy of the same material system requires integrating computational and experimental workflows. Challenge: combining workflows across facilities.

Computational Science: Complex Combination of Multiple Tasks

- 3 categories levels of tasks are common
  - One parallel operation in shared or distributed memory of a program (L1)
  - One program in HPC a workflow (L2)
  - One workflow in a distributed federation (L3)
- Different models for each level
  - L1 distributed and shared memory
  - L2 in situ workflows
  - L3 scripts
- Lack of interoperability between levels
  - Unnecessary complexity
  - Sacrificed performance and scalability

Three-level hierarchy of integrated materials science problem.
Background and Progress to Date

Decaf: Dataflows for Workflows
Workflow Model: Directed Graph

- A directed graph of tasks and communication between them
- Graph nodes are the tasks
  - Large tasks (ie., programs), not small tasks (ie., threads)
  - Programs are parallel
- Graph links are the communication
  - Also parallel

A directed workflow graph. The graph does not have to be acyclic (digraph, not DAG).
Dataflow Model: Parallel Communication in a Workflow

- For a pair of nodes connected by a directed edge (link) in the workflow graph,
- Dataflow is communication over the links in a workflow
- Workflow is in terms of tasks; dataflow is in terms of ranks

Dataflow between 2 workflow graph nodes. One may decompose any complex graph into a set of producer-consumer groups.
Links are Parallel

- Translate task-level put/get calls into rank-level MPI calls
- Links (can) have resources too
  - Run a parallel program, (almost) just like a node task

The dataflow can do more than just transmit messages. By assigning resources and running code in the dataflow, data transformations needed to connect producer to consumer are performed. We call this dataflow a “link.”
Common Redistribution Patterns

Different ways to split/merge a data model on M producer ranks to N consumer ranks

- Bounding Box
- Round robin
- Contiguous
- Block

- Bounding box redistribution of 3 producer ranks to 2 consumer ranks
- Block redistribution of 4 producer ranks to 2 consumer ranks
- Contiguous redistribution of 3 producer ranks to 2 consumer ranks
- Round robin redistribution 3 producer ranks to 2 consumer ranks

- Contiguous redistribution of 3 producer ranks to 2 consumer ranks
Data Rates Can Vary

Space Division Coupling

Space division coupling has to accommodate varying data rates of producer and consumer

Space Division Coupling with Flow Control

Multiple levels of memory/storage can act as burst buffers to allow a fast producer to couple to slow consumer
Python Workflow Definition

- Define in Python
- Add nodes and edges
- Add attributes to nodes and edges
- Convert to JSON
- Read JSON in application

```python
# initialize
import networkx as nx
import os
import imp
wf = imp.load_source('workflow', os.environ['DECAF_PREFIX'] + '/python/workflow.py')
mod_path = os.environ['DECAF_PREFIX'] + '/examples/direct/modify_cycle_4nodes.so

# define workflow graph
w = nx.DiGraph()

w.add_node("node_b", start_proc=5, nprocs=1, func='node_b')
w.add_node("node_d", start_proc=9, nprocs=1, func='node_d')
w.add_node("node_c", start_proc=7, nprocs=1, func='node_c')
w.add_node("node_a", start_proc=0, nprocs=4, func='node_a')

w.add_edge("node_c", "node_d", start_proc=8, nprocs=1, func='dflow', path=mod_path, prod_dflow_redist='count', dflow_con_redist='count')
w.add_edge("node_a", "node_b", start_proc=4, nprocs=1, func='dflow', path=mod_path, prod_dflow_redist='count', dflow_con_redist='count')
w.add_edge("node_a", "node_c", start_proc=6, nprocs=1, func='dflow', path=mod_path, prod_dflow_redist='count', dflow_con_redist='count')
w.add_edge("node_b", "node_a", start_proc=10, nprocs=1, func='dflow', path=mod_path, prod_dflow_redist='count', dflow_con_redist='count')

# convert the graph into a JSON config file
wf.workflowToJson(w, mod_path, "cycle.json")
```
Decaf: Decoupled Dataflows

Decaf generates dataflows for workflows.
- Decoupled workflow links with configurable dataflow
- Data redistribution patterns
- Flow control
- Resilience

If Decaf is the only workflow software, then it needs to do manage the workflow.
- Input workflow graph definition (Python)
- Instantiate the graph (Decaf constructor)
- Launch the tasks (run the nodes)
- Communicate (run the links)
- Terminate cleanly (when one of the tasks does)

But Decaf doesn’t have to be the top level workflow manager: e.g., Swift or Damaris or FlowVR or ADIOS or PyCOMPSs
- In such cases, Decaf just runs the links
Three Highlights
We applied the Decaf redistribution library to the Gromacs molecular dynamics code in order to visualize isosurfaces from molecular density. Code complexity was reduced dramatically, while maintaining performance improved.

Three different redistributions are performed while computing an isosurface from an MD simulation of 54,000 lipids (2.1M particles). [Dreher et al. 2014]
Computational Steering in Molecular Dynamics

The FepA protein is a channel on the surface of a cell through which iron can pass. FlowVR visualization of the steering progress.

Steering workflow with Decaf, Damaris, and FlowVR. Decaf and FlowVR tasks are in space-division mode while Damaris is in time-division mode.

Strong scaling of the steering pipeline without visualization. The measured time is the average over 100 iterations.

Strong scaling of the end-to-end workflow shows good efficiency. The analysis time effectively overlaps the simulation time.

Density estimation: Tessellations as intermediate representations enable accurate regular grid density estimators.

Output of in situ cosmology analysis workflow at four time steps

Nested Task Parallel Workflows
Nested Task Parallelism: A Coherent Model

- **Existing**
  - Various incompatible L2 workflow tools
  - L2 dataflow (Decaf)
  - L1 and L3 task parallel tools

- **Missing**
  - L2 task parallel workflow

- **Challenges and gaps**
  - Usability
  - Dynamicity
  - Scalability
  - Interoperability

A nested task-parallel abstraction for scientific computing from a federation of distributed facilities down to a single compute core (example: cosmology)
Usable Workflows

- Problem: Workflows are hard to construct and error-prone
- Solution: Workflow task typing, data model typing, workflow graph templates
Workflow Task Typing

- Dedicated task types
  - Fork
  - Join
  - Buffer
  - Compute
- Benefits
  - Usability: less user code
  - Security: more type checking
  - Performance: minimum data exchanged

Top: A thin layer of workflow code surrounds majority of user code. Bottom: Task typing requires less user code because typed tasks provide the common functions.
Workflow Graph Templates

- Add templates the same as individual tasks
  - Dynamically
- Predefined templates
  - Pipeline
  - Fork
  - Join
  - Cycle
- User-defined templates
- Star entities
  - Recursive templates
- Uses
  - (Re)usability
  - Benchmarking
  - Training

Workflows can be built from pipeline, fork, join, cycle, and user-defined building blocks.

“Star” nodes can be a generic compute task, any of the above building blocks, or a virtual task in the nested hierarchy.
Dynamic Workflows

• Problem: Time-evolving phenomena require dynamically changing workflows
• Solution: Resource changes, global graph changes, local graph changes
Resource and Graph Changes

- Resource changes
  - Number of compute nodes
  - Types of compute nodes
  - Time and space division
- Graph changes
  - Global changes: stop and restart
  - Local changes: continue execution

Different coupling options for a producer and consumer task in the same workflow graph.
Scalable Workflows

• Problem: Workflows need to perform well on a variety of h/w and s/w systems
• Solution: Exascale hardware, Post-Moore hardware, DOE system software
Hardware and System Software Support

- Pre-exascale machines
  - Cori, Theta, Aurora, Summit
- Exascale machines
  - ALCF and OLCF 2021
- Post-Moore architectures
  - Quantum
  - Neuromorphic
  - FPGA, SOC
- System software
  - ASCR research projects
  - ECP software projects
  - NNSA ASC projects
  - Decaf abstraction layer

L2 workflow software will utilize existing software funded by other research projects through an abstraction layer that will be added to Decaf.
Nested Workflows

- Problem: Workflows are single-level and do not cooperate across levels
- Solution: Virtual tasks, cross-level integration, nested usability, dynamism, scalability
Virtual Tasks

- Virtual task: Child level that appears as a single task in the parent level
- Protocol for communicating between parent and child
  - Science data
  - Control data
- Usability across levels
  - Implicit vs. explicit task parallelism
- Dynamics across levels
  - Decision making
- Scalability across levels
  - Virtual tasks extend scalability

Communication across the parent-child virtual task when the virtual task is a gray box includes bidirectional science and control data.
Early Results and Applications

Interoperability examples and application engagement
Early Examples of L3-L2 Nesting

- **PyCOMPSs**
  - PyCOMPSs extended so that a Decaf dataflow can become a task of a PyCOMPSs workflow

- **Pegasus**
  - Pegasus can define a Decaf dataflow in its .DAX XML configuration file

- **Swift**
  - Swift can communicate over a Decaf dataflow

- All of these examples are static, and “hard-coded”

```
@constraint(computingUnits="4")
@decaf(dfScript= 'decaf/linear_2nodes.py',
       wfExecutor= 'linear2.sh', computingNodes=2)
@task(arrayfile=FILE_INOUT)
def array_increment_pipeline(fileprefix, arrayFile):
    pass
```

“Decorator” for Decaf dataflow definition and execution in PyCOMPSs configuration file. (Courtesy R. Badia and J. Ejarque, BSC)
Early Examples of L2-L1 Nesting

Decaf L2 dataflow

L1 programming models
- HACC: MPI
- TESS: DIY + MPI
- Dense: DIY + MPI + OpenMP

L2 dataflow nested with various L1 programming models to estimate dark matter density in cosmology simulations
Application Engagement

- Superconductivity
  - TDGL code, feature detection and tracking
- Cosmology
  - HACC code, compare simulations and sky surveys
- Photon science
  - LAMMPS code, compare APS experiments with simulations

TDGL simulation of superconductors

HACC cosmology simulation

Gold nanocrystal high-energy X-ray images
Nested task parallelism is a coherent abstraction for L2 workflows.

- **Improved usability**
  - Workflow task typing, data model typing, templates

- **Workflow dynamism**
  - Run-time workflow definition and modification
  - Local and global resource and graph changes

- **Performance and scalability**
  - Task types and templates for exascale and Post-Moore
  - Dataflow abstraction layer for DOE software

- **Nested interoperability**
  - Parent-child communication
  - Usability, dynamism, scalability across levels

“A task-parallel in situ workflow system for L2 that will expose a task-parallel user interface, backed by a dynamically scheduled runtime, and will feature a nested task-parallel interface to L1 and L3.”
References
Further Reading

Acknowledgments

Facilities
Argonne Leadership Computing Facility (ALCF)
Oak Ridge National Center for Computational Sciences (NCCS)
National Energy Research Scientific Computing Center (NERSC)

Funding
DOE SDMAV Exascale Initiative

People
Franck Cappello (ANL), Matthieu Dreher (ANL), Jay Lofstead (SNL),
Patrick Widener (SNL)