Adaptive Parallelism for Coupled, Hybrid Programs

CW2020

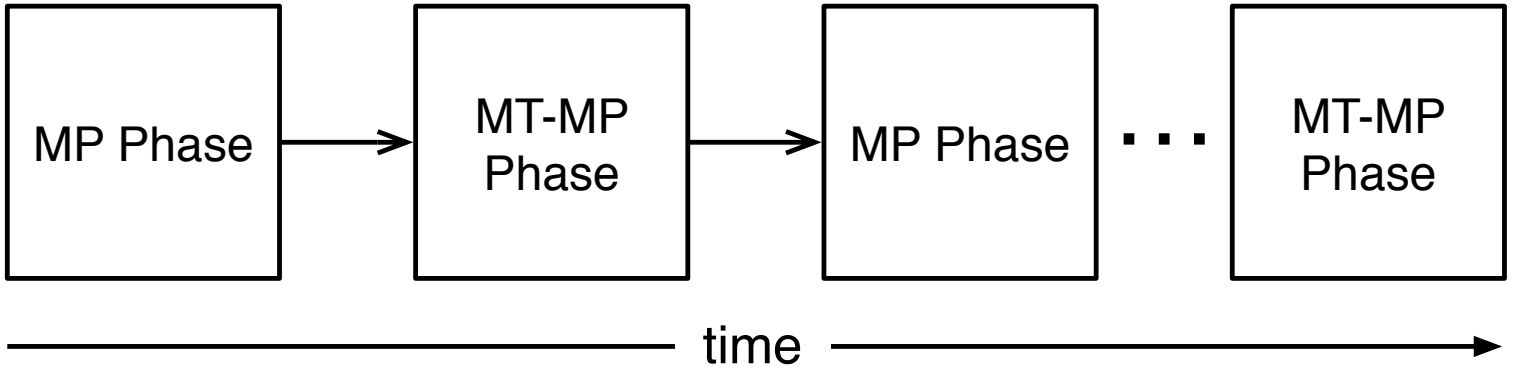
Samuel K. Gutiérrez

September 22, 2020

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

Typical Hybrid Programming Models in HPC

- Multi-threaded message-passing (MT-MP) applications becoming more popular (e.g., MPI+OpenMP)
- MP + MT-MP commonplace in *coupled applications*
  - MP (e.g., MPI Everywhere) and MT-MP libraries linked together with interleaved execution

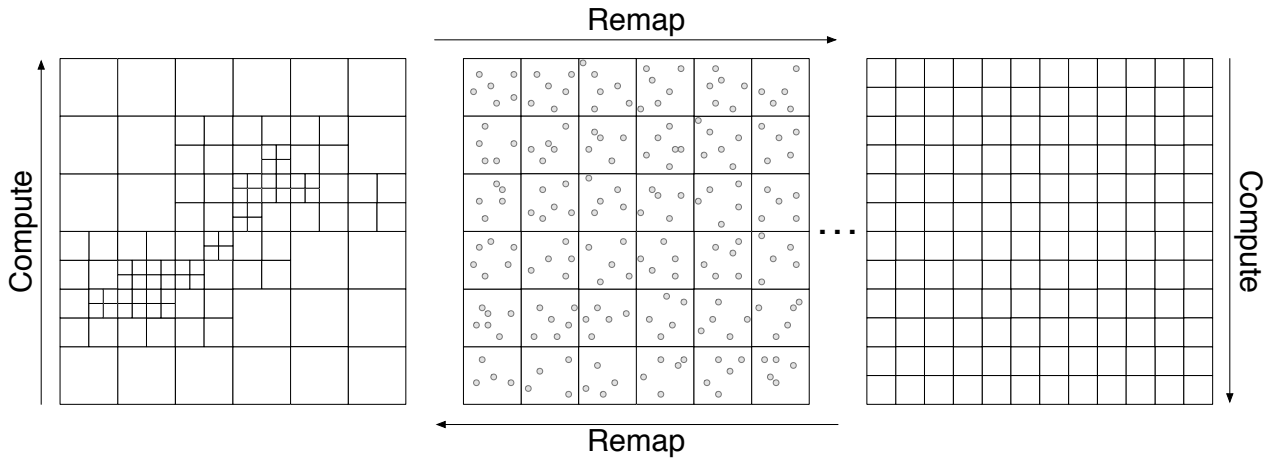


Where does this Type of Coupling Occur?

Parallel and distributed multi-physics applications

- Crucial in science and engineering
- Often interdisciplinary effort
  - Built by integration (or *coupling*) of independently developed and tuned software libraries linked into a single application

Phases of single, coupled application →



Challenges of Coupling Disparate Libraries

Each library should execute based on its design and tuning

- Each has own optimal *runtime configuration*
  - E.g., number and placement of *tasks* (processes and threads)
  - Poor library configuration → poor library performance → poor application performance
- Static parameters–often found manually, heuristically, and offline
- *Configuration conflicts* arise when optimal library configurations conflict with each other

State of the Practice

- Task-to-hardware bindings can improve performance
- Parallel launchers (e.g., srun, orterun) support only static binding and placement
  - Launch-time configuration persists for entire execution
- Two basic, static configuration options:
  - *Over-subscription* or *under-subscription*

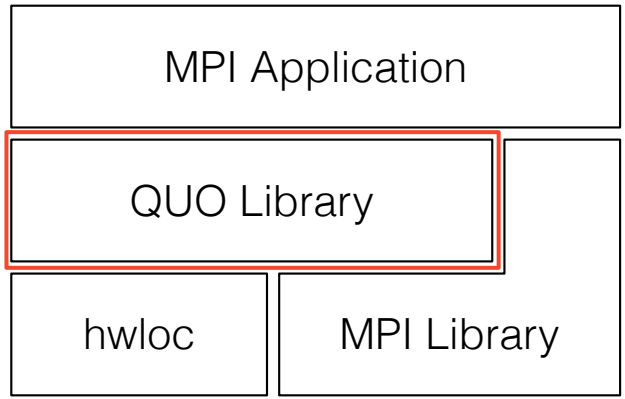
Quo Approach: Efficient Dynamic MPI+X

- Dynamic, run-time configuration conflict resolution for performant coupled thread-heterogeneous MP apps.

- Specifically, dynamic, coupled applications based on the Message Passing Interface (MPI)
  - MPI+X, where X is a Pthread-based runtime library
    - E.g., MPI-everywhere plus MPI+OpenMP or MPI+std::thread

Quo Features

- Programmer driven
- Hardware/software state queries (hardware topology, task config.)
- Efficient task reconfiguration (task placement, task affinity)
- Composable and general
- Open-source library (github.com/lanl/libquo)
- C and Fortran interfaces



Quo Main Concepts

- Contexts (instance handles)

QUO_create(&ctx, MPI_COMM_WORLD)

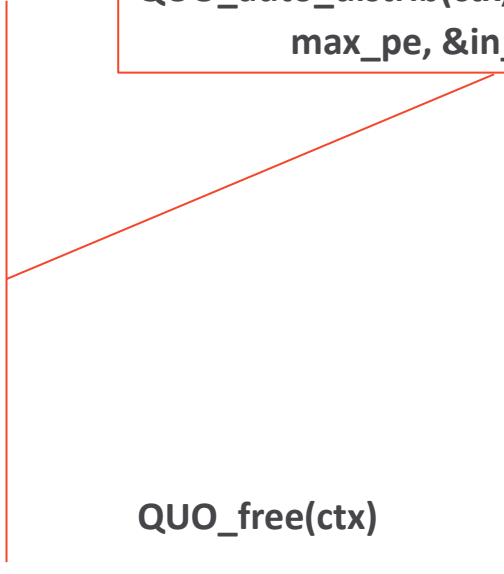
QUO_free(ctx)

Quo Main Concepts

- Contexts (instance handles)
- Hardware and software environment queries

```
// Hardware topology queries
QUO_nobjs_by_type()
QUO_nobjs_in_type_by_type()
// Intra-task state queries
QUO_cpuset_in_type()
QUO_bound()
// Inter-task state queries
QUO_qids_in_type()
```

```
QUO_create(&ctx, MPI_COMM_WORLD)
// Dynamically determine target resource
tres = QUO_OBJ_NUMANODE
// Query hardware/software at run-time
QUO_auto_distrib(ctx, tres,
                 max_pe, &in_dset)
```



```
QUO_free(ctx)
```

Quo Main Concepts

- Contexts (instance handles)
- Hardware and software environment queries
- Dynamic process affinities
  - Stack-based semantics

```
QUO_create(&ctx, MPI_COMM_WORLD)
// Dynamically determine target resource
tres = QUO_OBJ_NUMANODE
// Query hardware/software at run-time
QUO_auto_distrib(ctx, tres,
                 max_pe, &in_dset)
```

```
QUO_bind_push(ctx, tres)
A_library_call(in_args, &result)
QUO_bind_pop(ctx)
```

```
QUO_free(ctx)
```

Quo Main Concepts

- Contexts (instance handles)
- Hardware and software environment queries
- Dynamic process affinities
  - Stack-based semantics
- Efficient node-local process quiescence

```
QUO_create(&ctx, MPI_COMM_WORLD)
// Dynamically determine target resource
tres = QUO_OBJ_NUMANODE
// Query hardware/software at run-time
QUO_auto_distrib(ctx, tres,
                 max_pe, &in_dset)
```

```
QUO_bind_push(ctx, tres)
A_library_call(in_args, &result)
QUO_bind_pop(ctx)

QUO_barrier(ctx)

QUO_free(ctx)
```

Quo Main Concepts

- Contexts (instance handles)
- Hardware and software environment queries
- Dynamic process affinities
  - Stack-based semantics
- Efficient node-local process quiescence
- Data dependencies

```
QUO_create(&ctx, MPI_COMM_WORLD)
// Dynamically determine target resource
tres = QUO_OBJ_NUMANODE
// Query hardware/software at run-time
QUO_auto_distrib(ctx, tres,
            max_pe, &in_dset)
// Satisfy outstanding data dependencies
if (in_dset)
        QUO_bind_push(ctx, tres)
        A_library_call(in_args, &result)
        QUO_bind_pop(ctx)

QUO_barrier(ctx)
// Satisfy outstanding data dependencies
QUO_free(ctx)
```

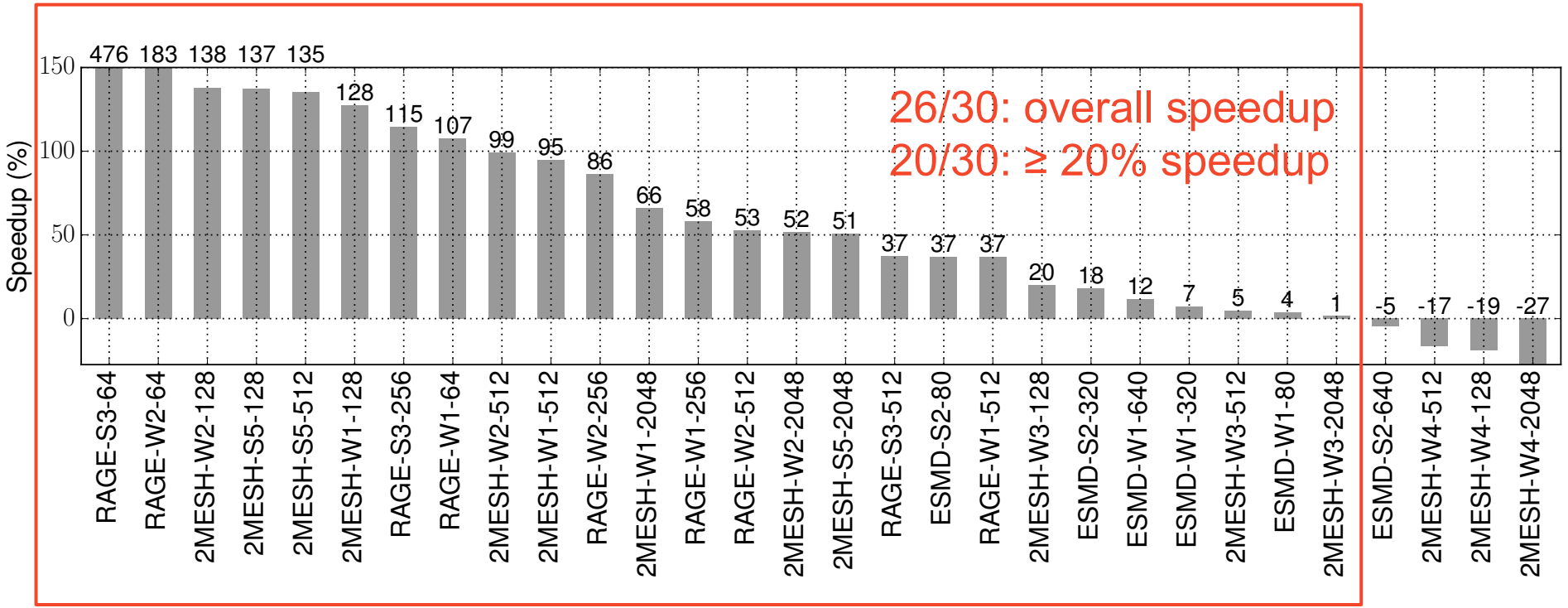
Quo Main Concepts

- Contexts (instance handles)
- Hardware and software environment queries
- Dynamic process affinities
  - Stack-based semantics
- Efficient node-local process quiescence
- Data dependencies
- Policy management

```
QUO_create(&ctx, MPI_COMM_WORLD)
// Dynamically determine target resource
tres = QUO_OBJ_NUMANODE
// Query hardware/software at run-time
QUO_auto_distrib(ctx, tres,
            max_pe, &in_dset)
// Satisfy outstanding data dependencies
if (in_dset)
        QUO_bind_push(ctx, tres)
        A_library_call(in_args, &result)
        QUO_bind_pop(ctx)

QUO_barrier(ctx)
// Satisfy outstanding data dependencies
QUO_free(ctx)
```

Quo's Efficacy



*Details Available in: Gutiérrez et al. *Accommodating Thread-Level Heterogeneity in Coupled Parallel Applications*. (IPDPS 2017). ieeexplore.ieee.org/abstract/document/7967136

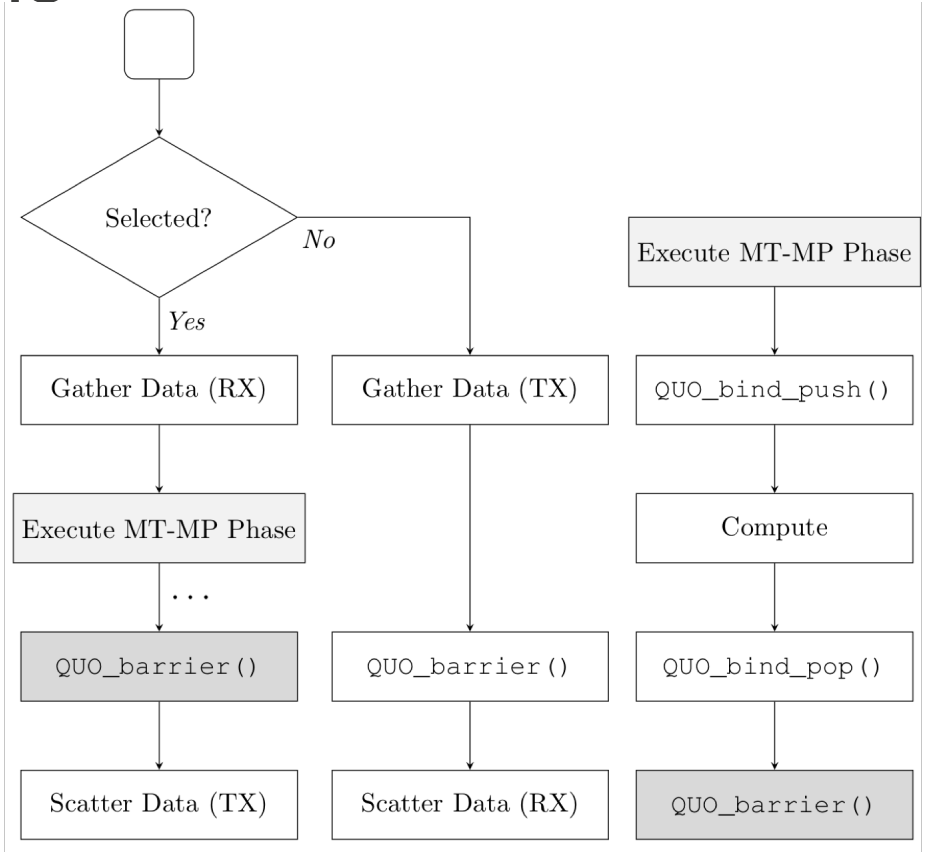
Practical Considerations

Increased code complexity

- Task quiescing + resumption → data remapping

Encapsulating code regions

- QUO_bind_push() + QUO_bind_pop()



*Details Available in: Samuel K. Gutiérrez. *Adaptive Parallelism for Coupled, Multithreaded Message-Passing Programs*. (2018). digitalrepository.unm.edu/cs_etds/95

In Summary

Coupled applications increasingly comprise MP libraries with differing preferred degrees of threading

Static execution environments are often suboptimal for resolving dynamic, phased configuration conflicts

Quo's approach can help improve overall application performance with dynamic execution environments

Thank You

Questions?

This research was supported by the Advanced Simulation and Computing Program of the U.S. Department of Energy's NNSA 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.