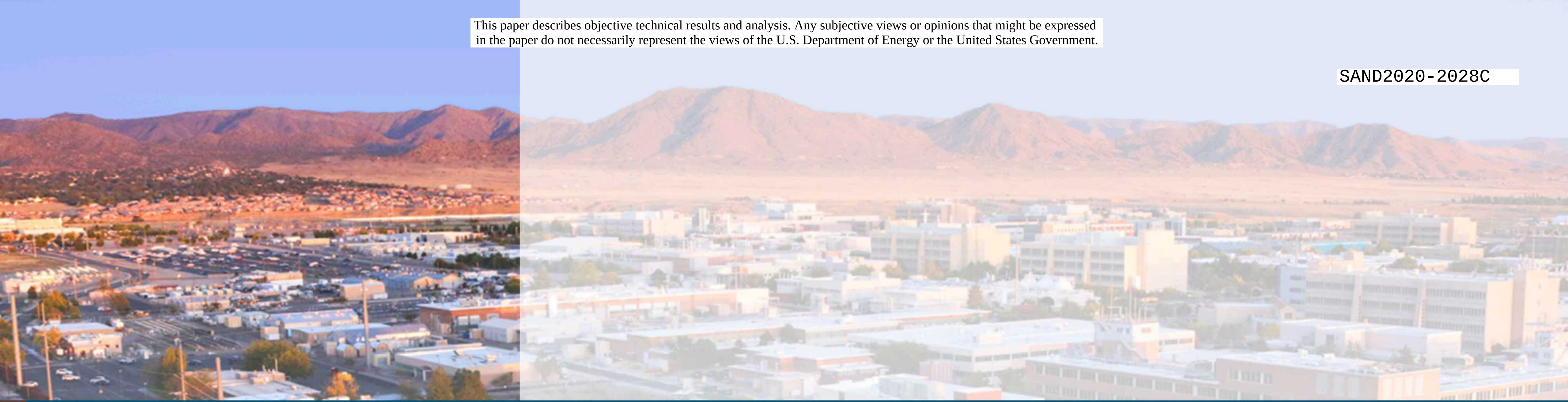
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2020-2028C



Oversubscription and Your Data, How User Level Scheduling Can Increase Data Flow

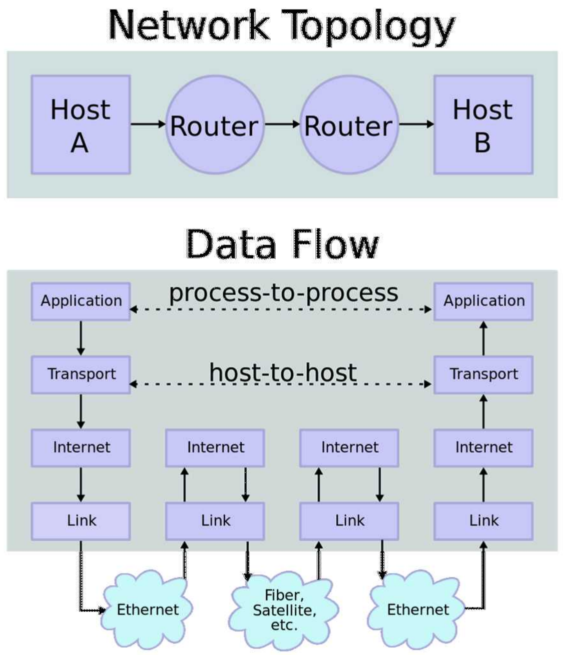
Noah Evans

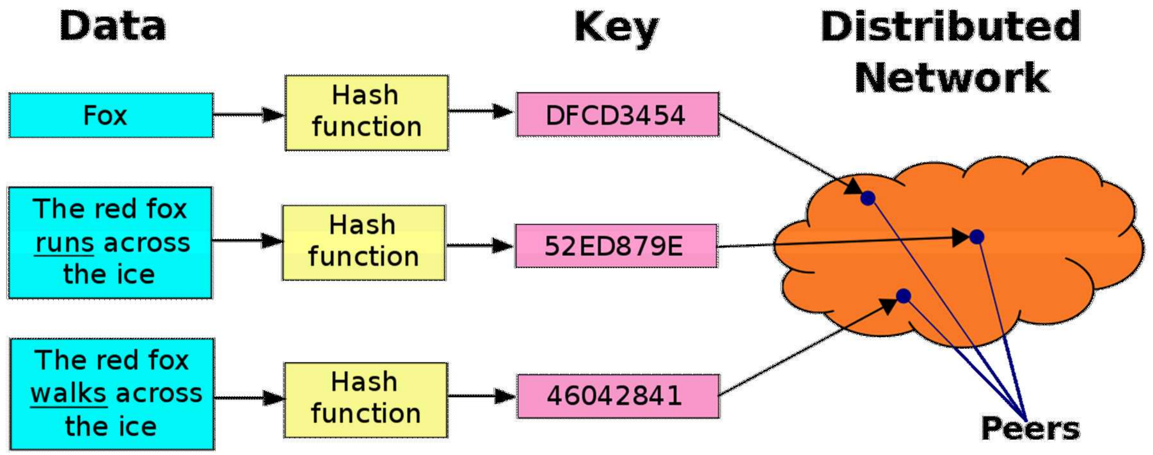
Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
(SAND2018-11134 C)

A brief and mostly wrong history of Data Centricity

- From the networking world
- Term has picked up baggage
- Don't address "where" (hosts, processes)
- Address "what" (data)
- This is not new at HPC at all --off node
- Barney will go into greater detail



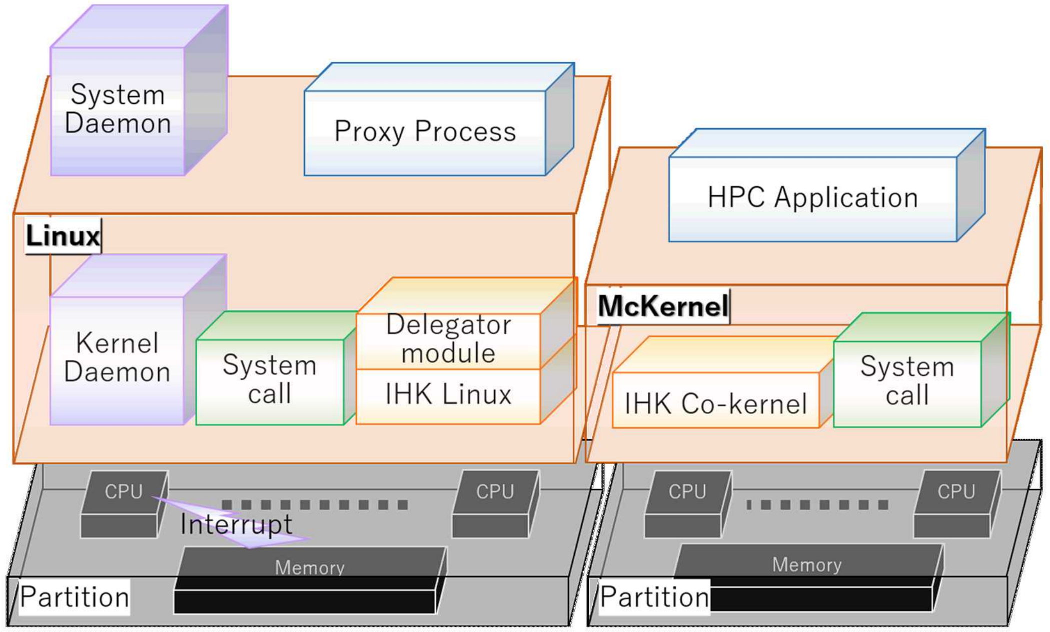


Data Centrism in Runtimes

- Modern HPC programming models a mix between *process* centric (MPI ranks) and data centric (OpenMP/GPU dataflow)
- You end up with a mix between runtimes
- This leads to "thread heterogeneity"
- Lots of work to integrate all of these programming models.
- Scheduling from above, wrapping MPI calls (OMPSs, QUO)
- Scheduling from below, using user level threading (Argobots, Qthreads, MPIC, FGMPI)
- All of these provide scheduling problems.
- Fundamentally a data flow problem. Progress threads/engines -> MPI ranks -> data parallel runtime -> communication

Example Problem: thread oversubscription on the McKernel

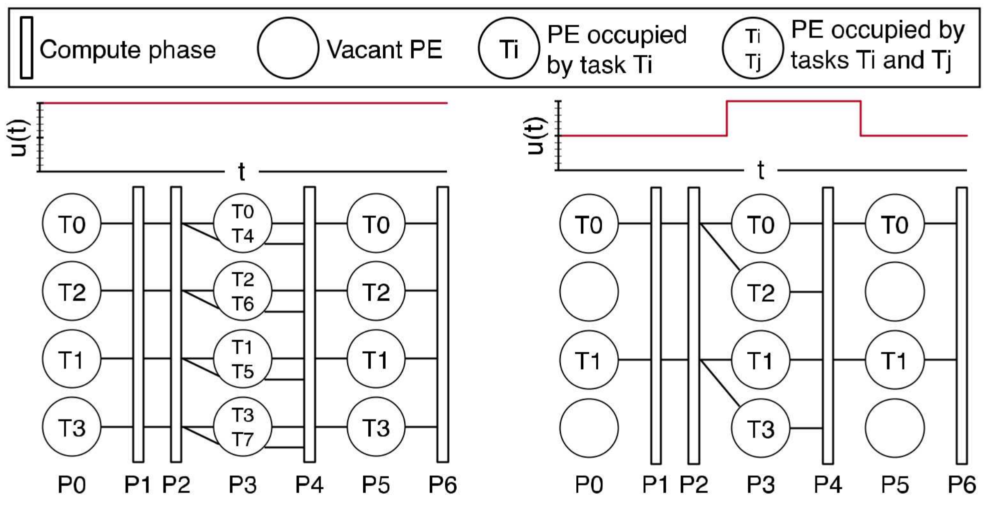
- We are running OpenMPI on the McKernel (thanks Balazs)
- OpenMPI is very promiscuous with its threads, 2 threads for every rank.
- Big performance hit, need to provide noise cores
- Want to be able to manage this in a much more efficient way.
- Is it possible to do this in an easier way without modifying the application?



Current state of the Art: QUO (LANL)

- Attempts to control “thread heterogeneity” (different threads used by different runtimes, which can conflict)
- Modifies application
  - First does an All to All collective to find the MPI Process topology
  - Then ”pushes” and “pops” a stack of thread affinities and quiesces unused threads
  - Effectively becomes a user level scheduler for pthreads

QUO gives Data Centric control over and undersubscription in HPC Application phases



Compute phase
Vacant PE
Ti PE occupied by task Ti
Ti Tj PE occupied by tasks Ti and Tj
u(t)
t
T0
T2
T1
T3
T4
T6
T5
T7
P0
P1
P2
P3
P4
P5
P6

(a) Time evolution of a static over-subscribed MPI+X configuration.

(b) Under-subscribed MPI+X with typical *wide* binding policy.

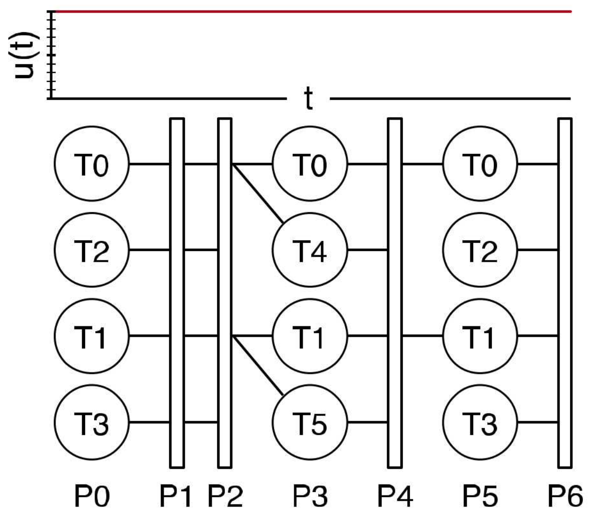


Figure 5: Illustration of compute resource utilization by tasks over time $u(t)$ for a QUO-enabled MPI+X configuration.

Gutiérrez, Samuel K., et al. "Accommodating thread-level heterogeneity in coupled parallel applications." 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2017.

Talk theme: Is it possible to make existing systems MPI+X Data Centric?

Massive investment in the simulation codes that are still MPI+X

Is it possible to use runtime composition techniques to be data centric?

Traditional User Level Scheduling Memory/CPU oriented

Typically a scheduler is attached to a physical device (a per CPU run-queue, a NUMA domain).

This allows *resource* based scheduling, but it’s tied to a fundamentally static idea of system resources.

Can schedule tasks, but scheduler, rather than data driven for the most part

Modern HPC runtimes (Legion, SDFG) provide data centric mechanisms, would require porting effort

We want to be able to do this without modifying applications.

Substrate-based runtime composition

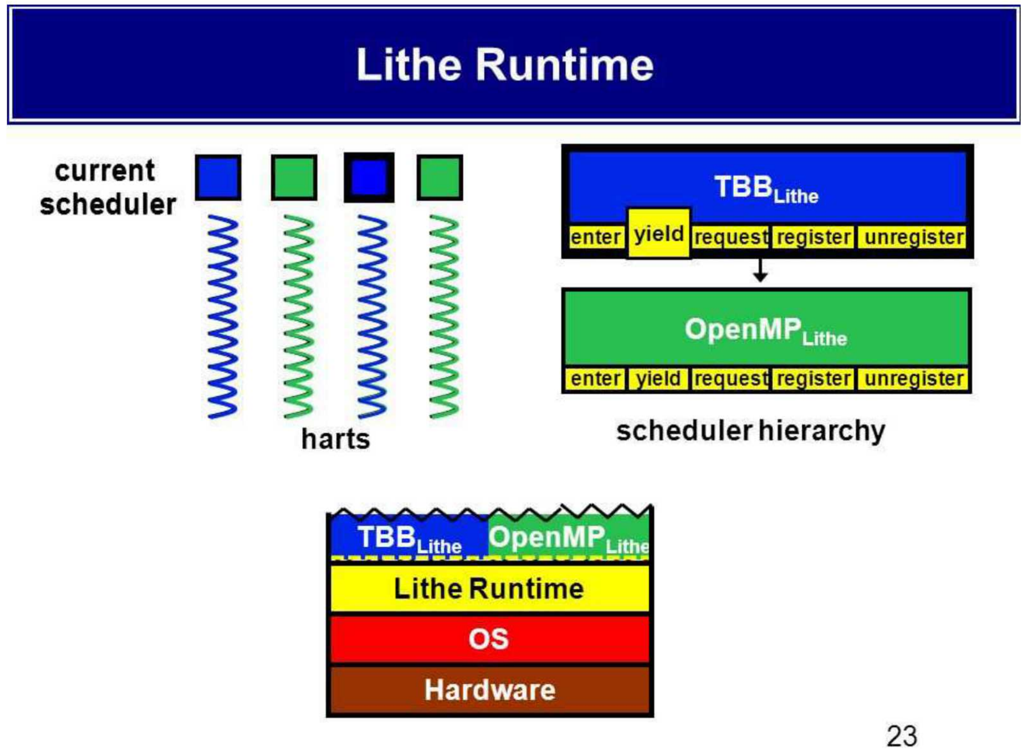
Substrate based runtime composition provides a common scheduling mechanism for runtimes.

Runtimes implement the policies

Still use the regular API but you have a substrate (under the covers)

Typically you need to have a way to schedule runtimes *cooperatively*.

Relies on good runtime behavior



Pan, Heidi. Cooperative hierarchical resource management for efficient composition of parallel software. Diss. Massachusetts Institute of Technology, 2010.

Substrates are usually CPU Centric

E.g. Lithe is based on multiplexing runtimes over HARTs (HARdware Threads == CPUs/Hyperthreads)

Can schedule on blocking I/O but no concept of data driven scheduling (i.e. scheduling over a particular data allocation)

This is fundamentally lossy, you don’t know your data flow

Is it possible to annotate that dataflow into substrate scheduling?

CCR
Center for Computing Research

Proposal: Data Driven Substrate Scheduling

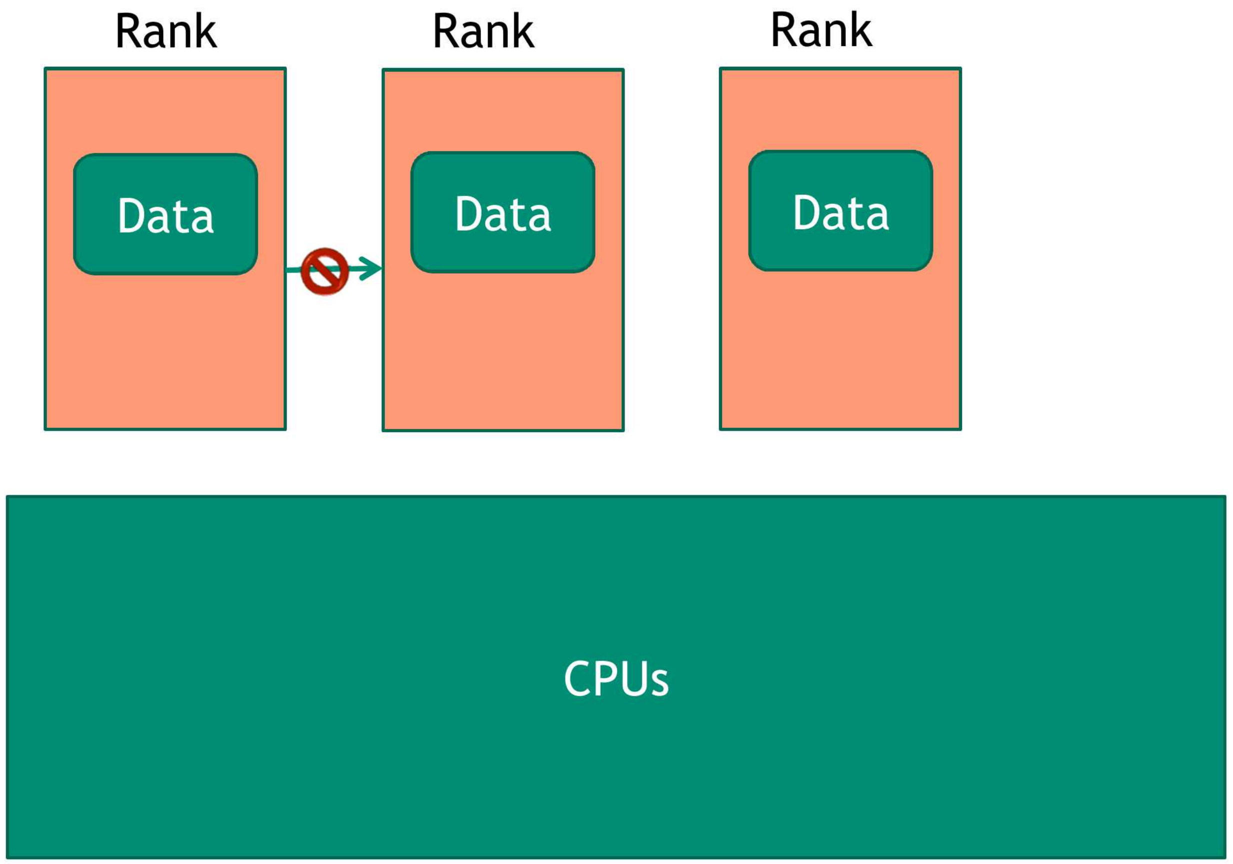
Add another layer to hierarchical scheduling: Data provenance

When data of interest (e.g. MPI Buffers) are allocated, attach a run-queue to the root scheduler.

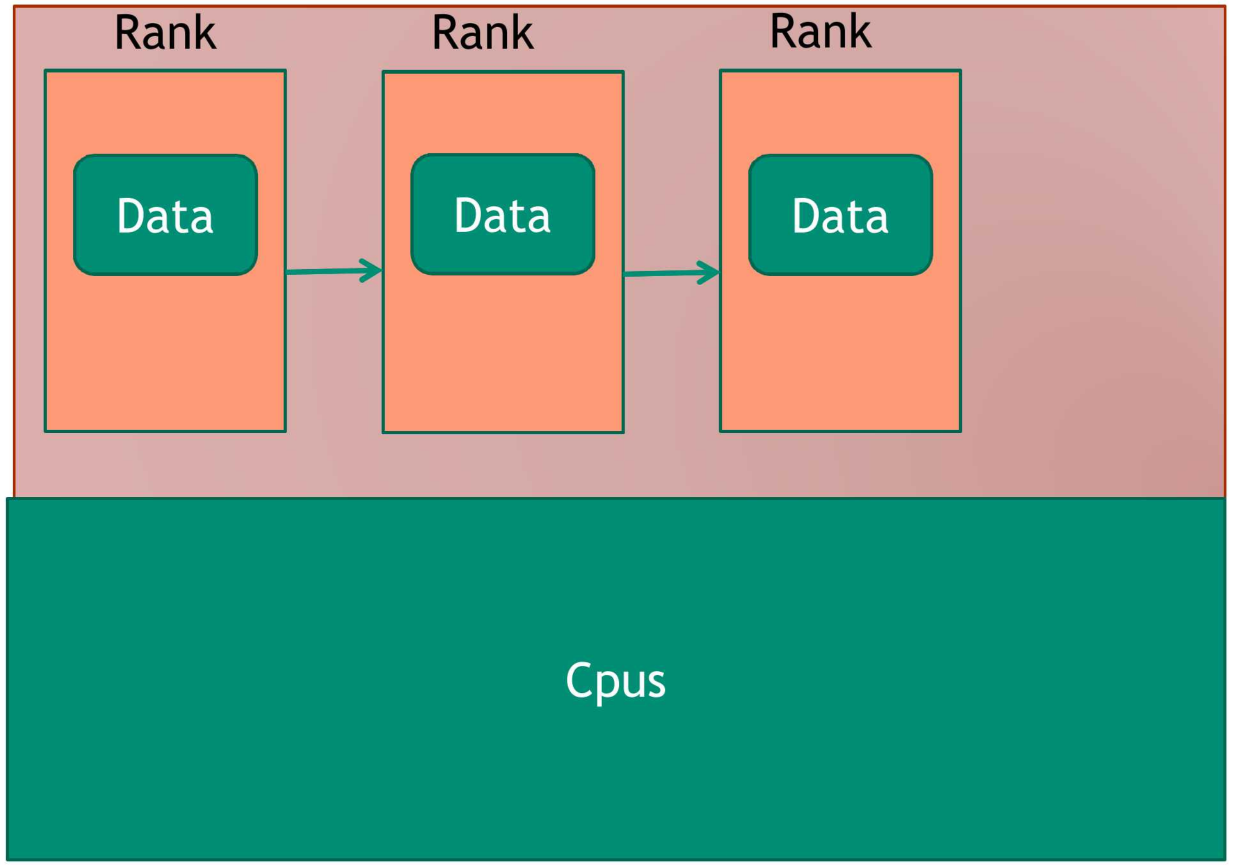
This allows Progress Threads/MPI/OpenMP to know the provenance of their data as they schedule.

However, some obvious problems with neat solutions.

Process: Process address spaces make data scheduling harder



Solution: Process In Process: Share Address Spaces



Problem: Implementing push and pop semantics

Quo relies on global knowledge of thread state to push and pop affinities after switching between runtimes.

Traditional CPU based schedulers don’t have this knowledge.

Know what’s on your run-queue, but don’t know the global state of the system.

To fully subscribe the system you need to know everything that’s going on.

Proposed Solution: Gang Scheduling with Data Inheritance

Schedule based on data regions, round robin.

- Each data region gang schedules cpu threads before passing to next region.

Ensures no conflicts because one region

Allows precise control of allocations and locality dynamically.

Potentially slower because you're scheduling globally.

Current Progress

Have a mechanism to replace OpenMPI's threading model to allow composition (currently a pull request in OMPI master)

- Uses OpenMPI's modular component architecture to decouple particular threading libraries from implementation.
- New threading models are shared libraries loaded at runtime.

Have a separate version of MPI and OpenMP that coschedule using Lithe.

- Baked in (earlier than thread MCA)
- Allows MPI ranks, progress threads, Orte and OpenMP to coschedule.
- Barrier synchronous is painfully slow, early results show better performance >96 nodes on KNL

Future Work

Add Data Queues via PiP’s glibc and Data Inheritance Scheduling to Lithe

Progress threads: Current approach to scheduling (Progress->MPI->OpenMP->MPI) is very static. Figuring out good join points to allow asynchronous progress to occur (during gang scheduling?)

Explore overdecomposed approaches: Current approach is very QUOish, come up with scheduling disciplines amenable to over decomposition.

Integrate Lithe into the new OpenMPI MCA and more modern OpenMP’s

Conclusions

Proposed a new form of scheduling based on Data Inheritance.

Gives explicit provenance to data and computation, making it possible to control oversubscription and dataflow in legacy applications.

Requires changing runtimes but not the applications.

Not possible in traditional scheduling models, processes don't have enough information about global state of the system.

By using Data Inheritance Scheduling over MPI Ranks represented as Processes in Process you can maintain the global knowledge of the system you need.

The runtime substrate is implemented.

Data Inheritance scheduling + PiP integration are future work

CCR
Center for Computing Research