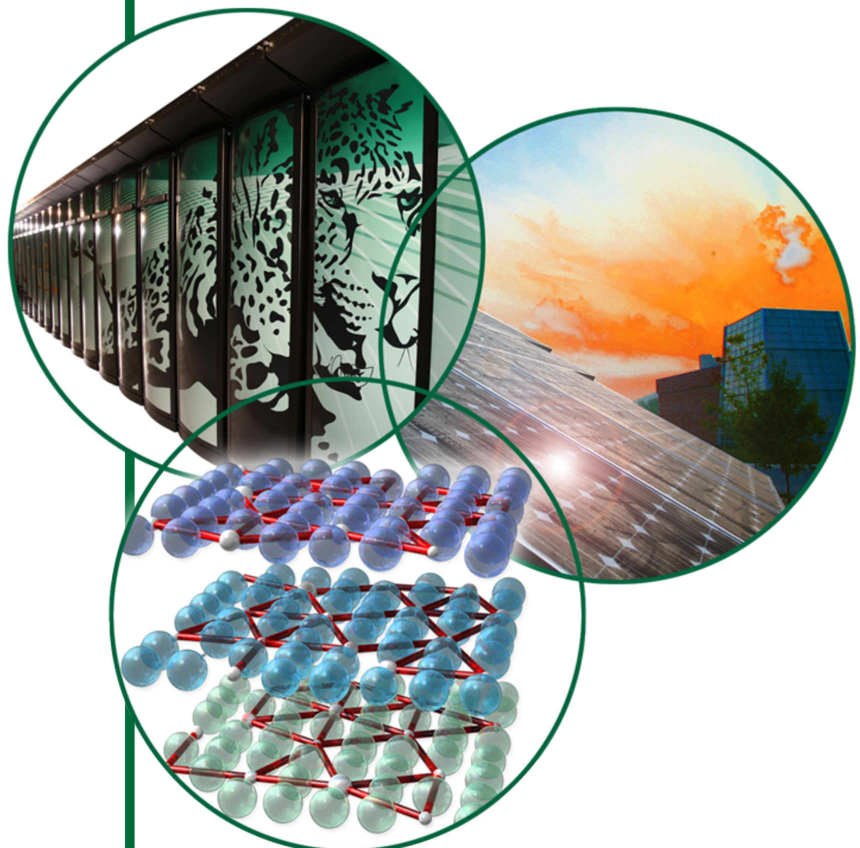
A Dependency-Driven Formulation of Parareal: Parallel-in-Time Solution of PDEs as a Many-Task Application



Wael R. Elwasif

Oak Ridge National Laboratory

elwasifwr@ornl.gov

U.S. DEPARTMENT OF ENERGY

SciDAC Scientific Discovery through Advanced Computing

OAK RIDGE NATIONAL LABORATORY

MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

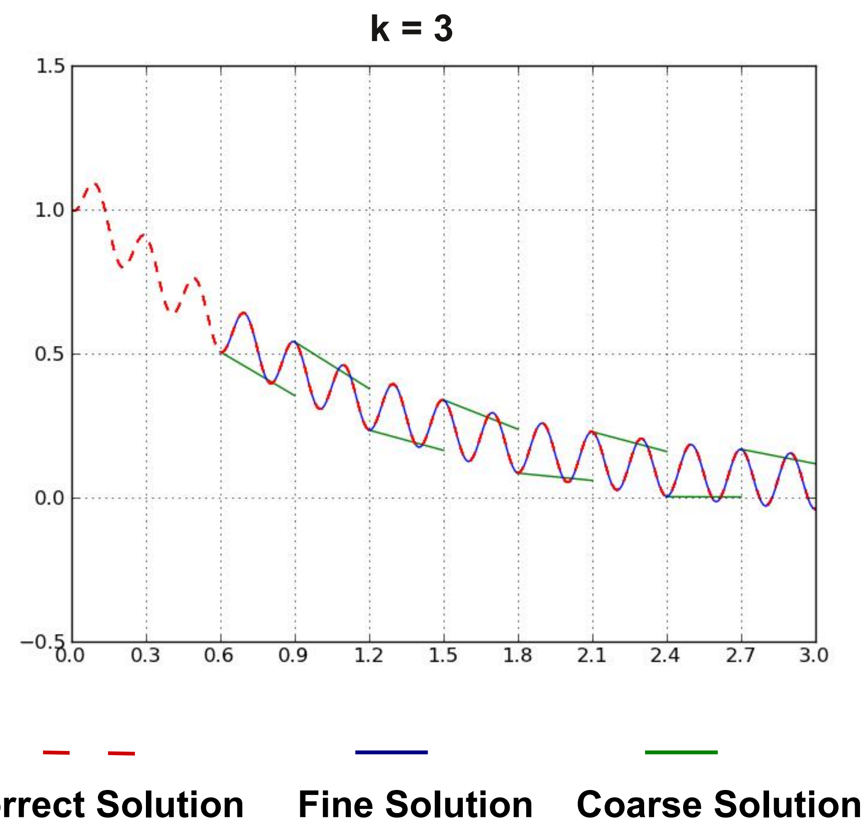
First: Shout-out to Coauthors

- **Samantha Foley, David Bernholdt, and Lee Berry**
  Oak Ridge National Laboratory
- **Debasmita Samaddar**
  ITER Organization
- **David E. Newman**
  University of Alaska, Fairbanks
- **Raul Sanchez**
  Universidad Carlos III de Madrid

Parareal: Trading Flops For Time

- Predictor-Corrector, iterative method for time-dependent PDE's
- Lions, Maday, and Turnici, 2001
- "Advance system state from initial condition λ_0 at time t_0 to time t_f, using ***N*** time "***slices***" (sub-intervals), each of size Δt, where $T = t_f - t_0 = N \Delta t$
- ***Fine,*** accurate (expensive) solver, ***F*** compute "***true***" solution
- ***Coarse***, approximate (fast) solver, ***G***, compute approximate solution
- Convergence tester, ***C***
- Guaranteed convergence in $K \leq N$ iterations
  - "Good" scenarios have $K << N$

$$F \equiv du/dt - \lambda u = sin(10\pi t)$$
$$G \equiv du/dt - \lambda u = 0 \qquad N = 10$$



Parareal: The Classic Algorithm

```
first_slice = 1
num_converged = 0
for iteration = 1, max_iterations
   for slice = first_slice..num_slices
      coarse_solve(iteration, slice)
   forall slice = first_slice..num_slices
      fine_solve(iteration, slice)
   for slice = first_slice..num_slices
      test_convergence(iteration, slice)
   num_converged +=
      first_non_converged_slice - first_slice
   if (num_converged == num_slices)
      end // SUCCESS
   else
      first_slice = first_non_converged_slice
end //Failed to converge in max_iteration
```

→ ***Sequential Phase*** (coarse_solve loop)

→ ***Parallel Phase*** (fine_solve forall loop)

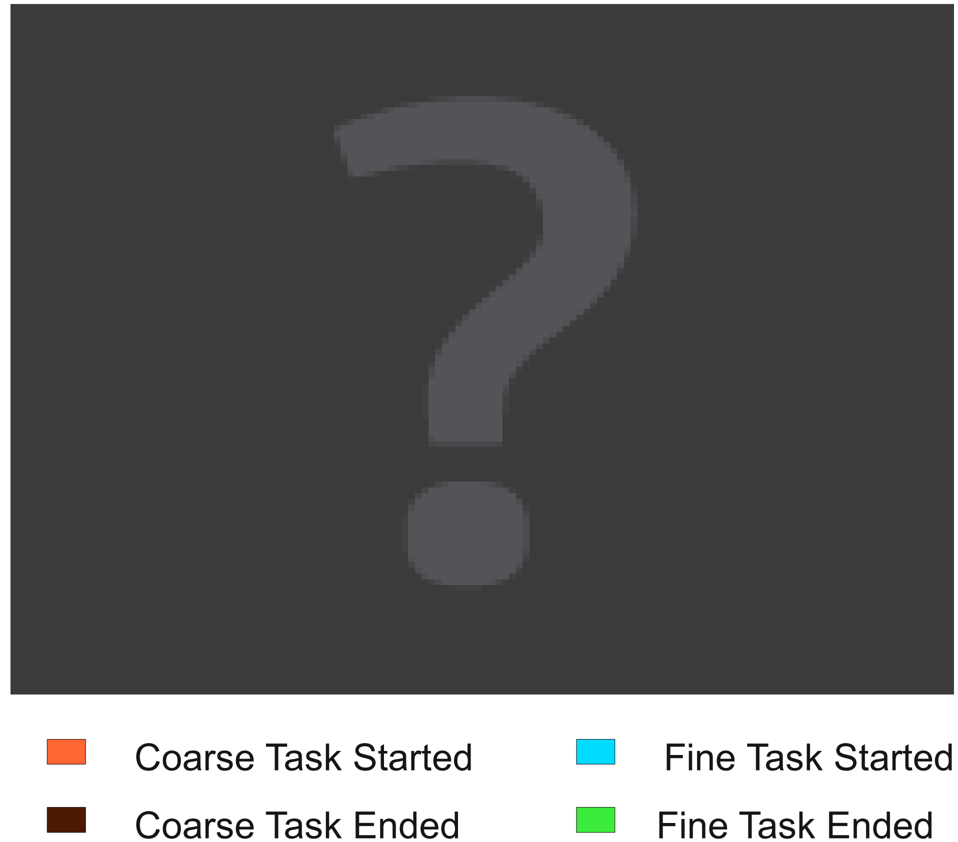
→ ***Computationally Cheap*** (test_convergence loop)

Motivational Problem

- Fusion Plasma Turbulence application (BETA) for **160** time slices on **1024** cores
- Fine solver uses **VODPK** adaptive integrator
- Coarse solver:
  - Reduced spatial resolution
  - Less accurate 4th order Runge-Kutta solver
- Implemented as a ***Many-Task*** problem
  - ***Separate MPI invocation per (coarse/fine) solve task per time slice per iteration***
  - ***File system used for inter-task data exchange***

Many Task Classic Parareal

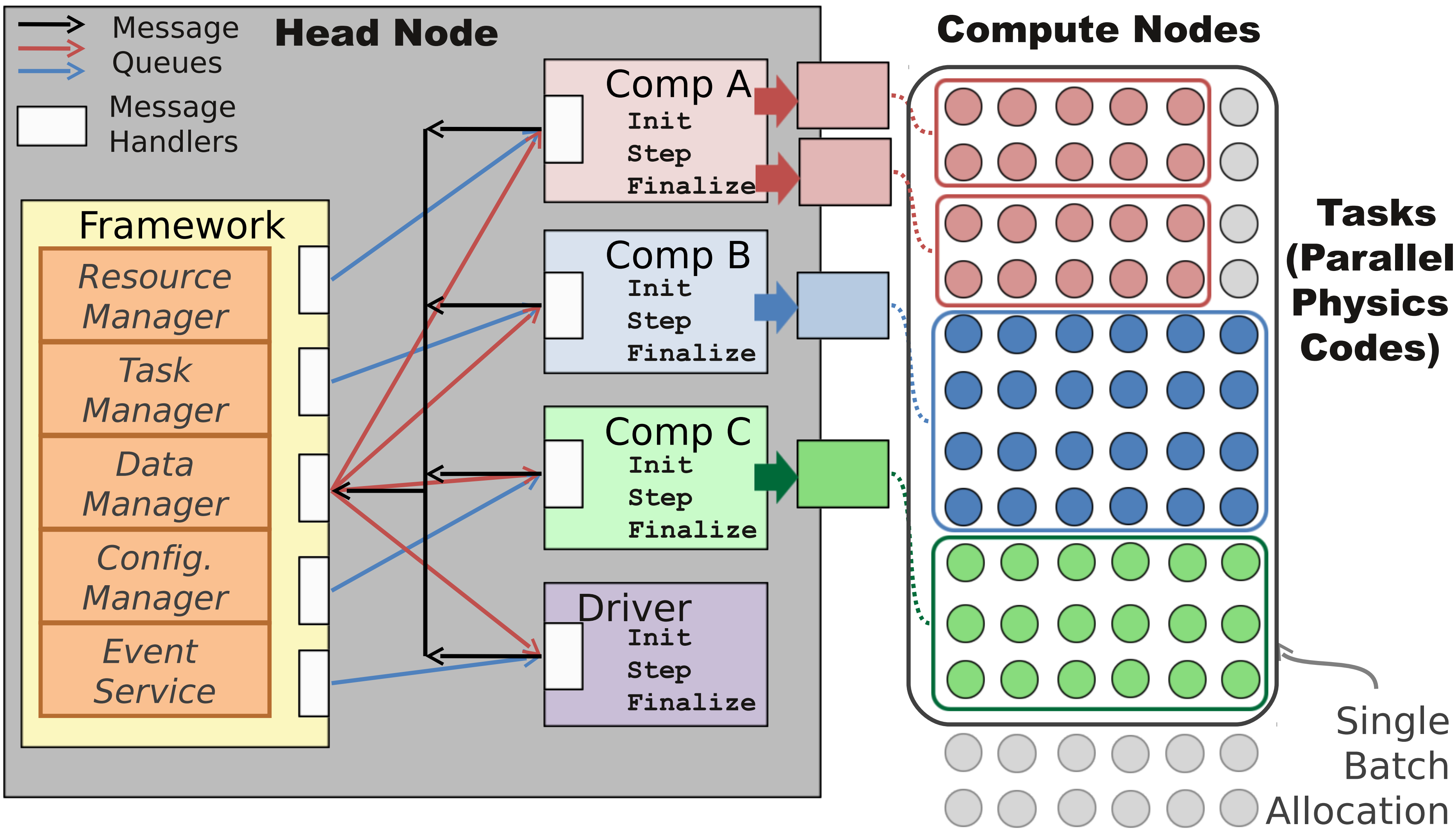
- Coarse tasks executed sequentially during each iteration
- All fine tasks for a given iteration are initiated as a single ***task pool***
  - Maximum of **64** fine solve tasks can be concurrently active on **1024** cores
- The order of task execution within a task pool is determined by the underlying execution framework
- Task Statistics:
  - Coarse Task : **16** Cores, **3.45 s**
  - Fine Task : **16** Cores, **223.15 s**
- Effective utilization: ***31.6 %***



The Underlying Framework
The Integrated Plasma Simulator (IPS)

- Component-based Python framework for loosely coupled simulations
- Originally designed for time-stepped fusion simulations
  - Flexibility allows use in other domains, and other control-flow paradigms
- Major features:
  - Thin component layer in Python that wraps stand-alone executables
  - Inter-component data exchange using the file system
  - Framework services used to assemble a simulation
    - **Resource management** - **Task management**
    - **Data management** - **Asynchronous event services**
  - Simulations execute within a single batch allocation

IPS Architecture & Execution Model



Concurrency in the IPS

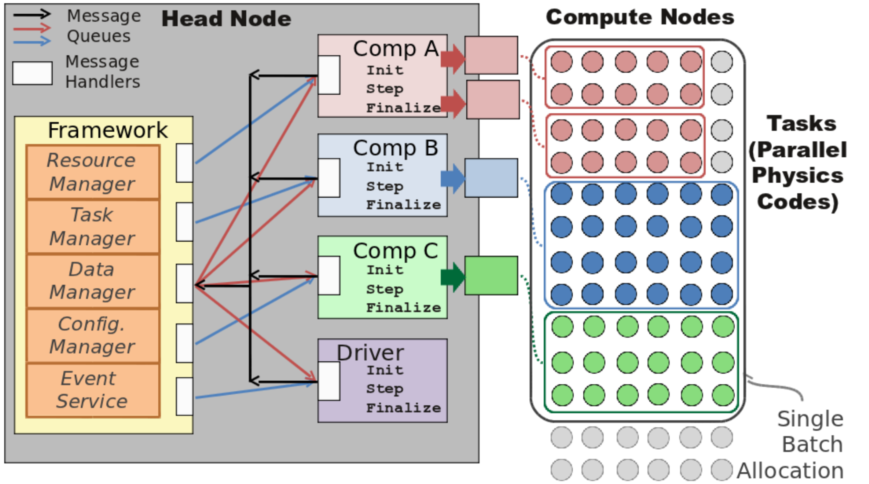
- Components launch parallel tasks
- Multiple concurrent tasks can be launched by the same component

Used to implement Classic Parareal

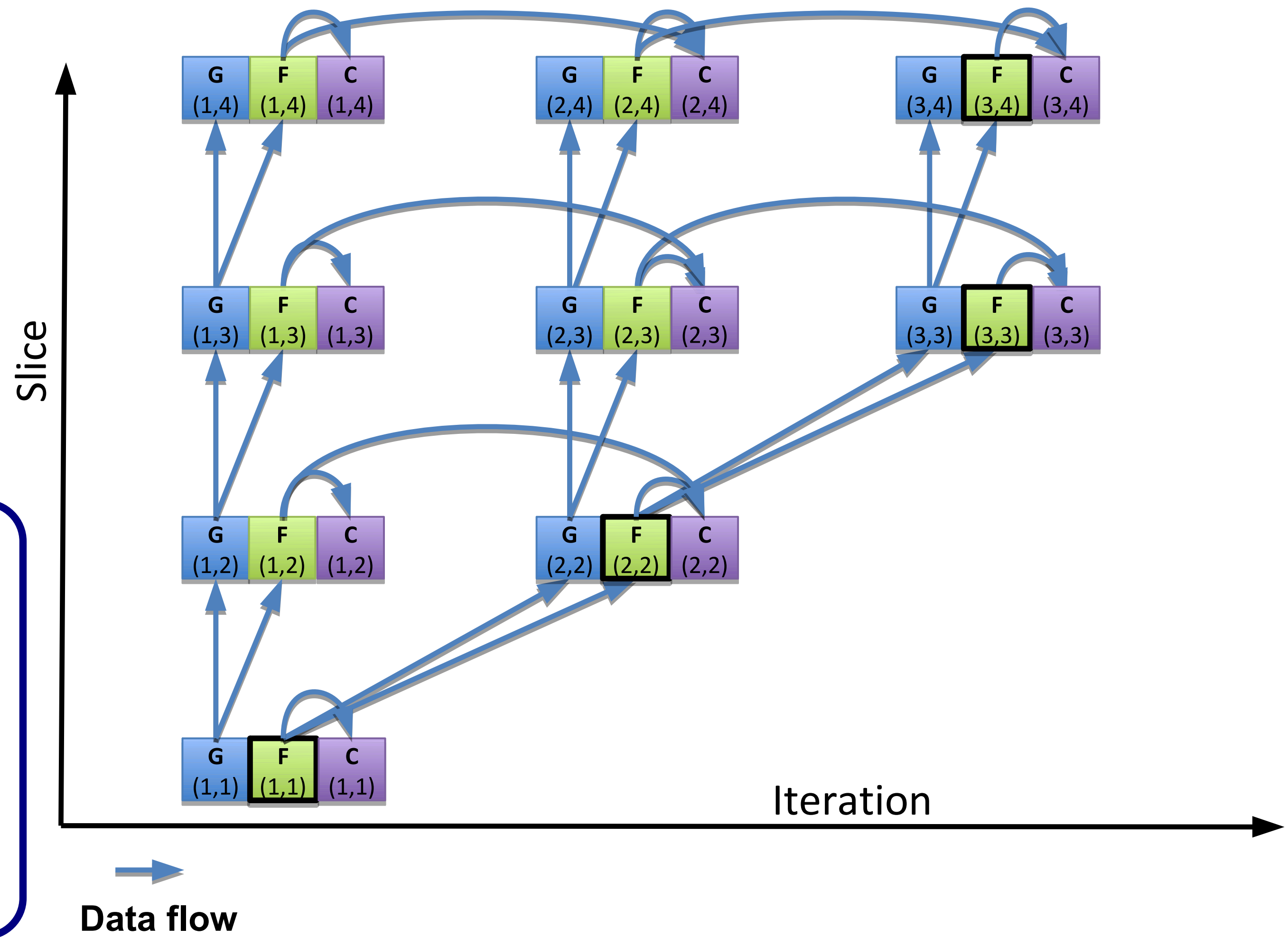
- Multiple components can be concurrently active

Used to implement Dependency-Driven Parareal

- Multiple simulations can execute within a single framework instance
  - Original IPS many-task application (see our MTAGS10 paper)



Task Dependencies in Parareal (1)

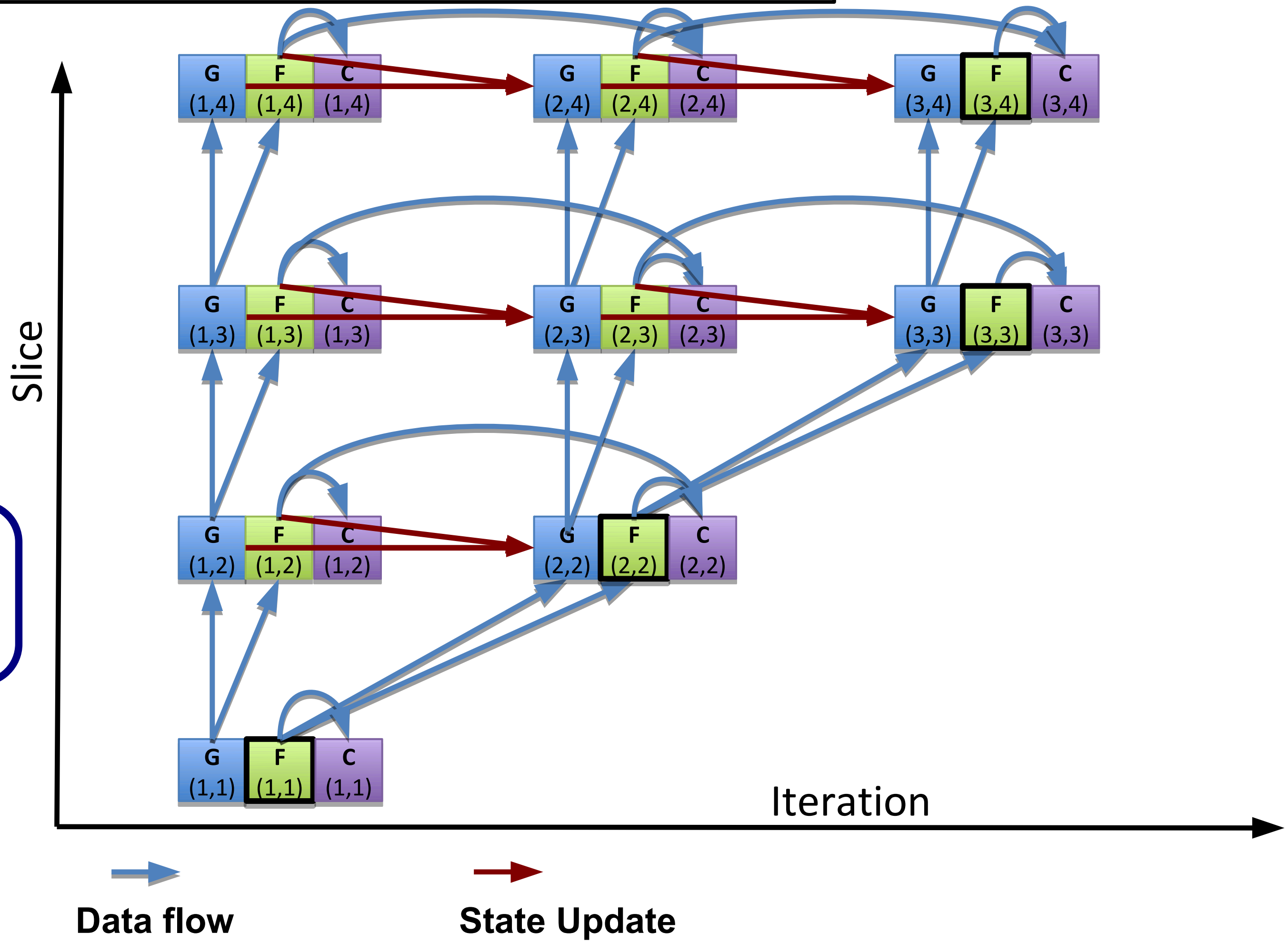


Basic ***data flow dependencies*** for computing intra-iteration coarse and fine states, and checking for convergence

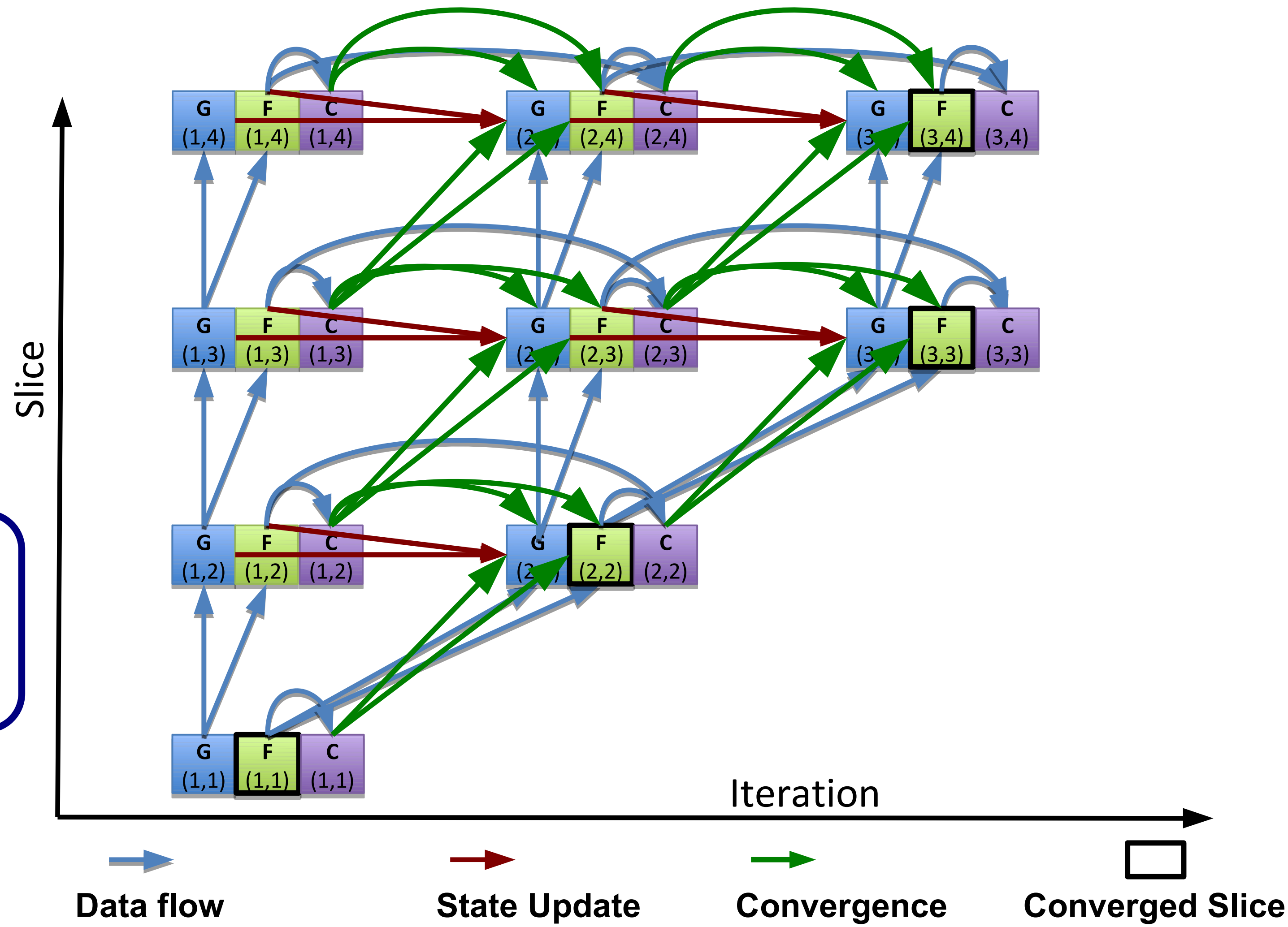
Task Dependencies in Parareal (2)

Parareal State Update $\Lambda_{k,i} = \lambda^G_{k,i} - \lambda^G_{k-1,i} + \lambda^F_{k-1,i}$

State updates "***correct***" output from coarse tasks



Task Dependencies in Parareal (3)



Convergence results control the execution of tasks in next iteration

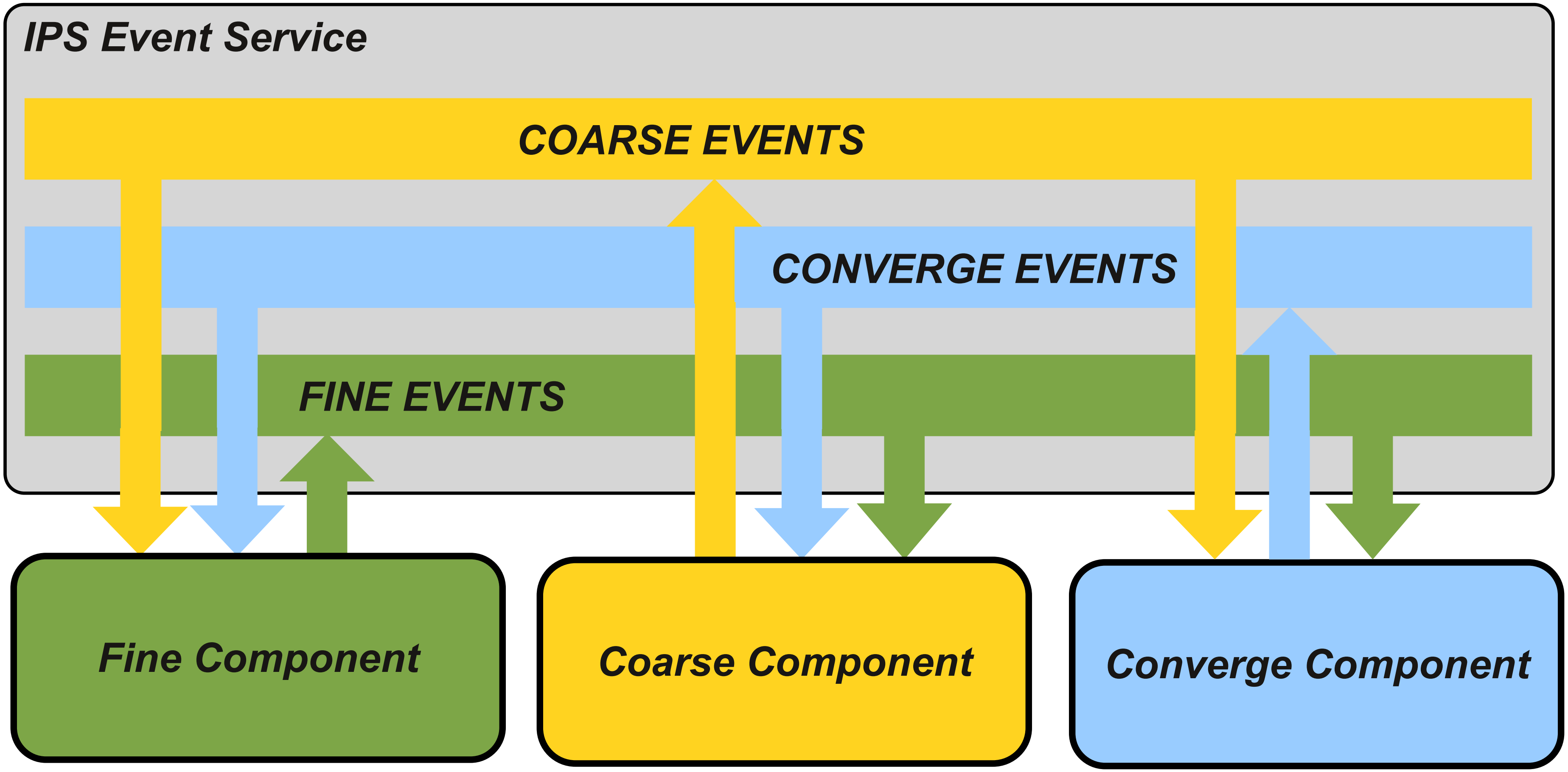
Dependency-Driven Parareal

- Three "***server***" components, **Fine**, **Coarse**, and **Converge**
- Distributed flow control:
  - Simulation logic spread across the three components
- Components initiate tasks independently
  - As as soon as ***all their dependencies are satisfied***
  - Each component encodes dependencies for its class of tasks
- Components manage their own task wait queues (FIFO)

Synchronization-reducing algorithms
- Break Fork-Join model

Jack Dongara *– Critical Issues at Peta & Exascale for Algorithm and Software Design*

Dependency Propagation Using the IPS Event Service

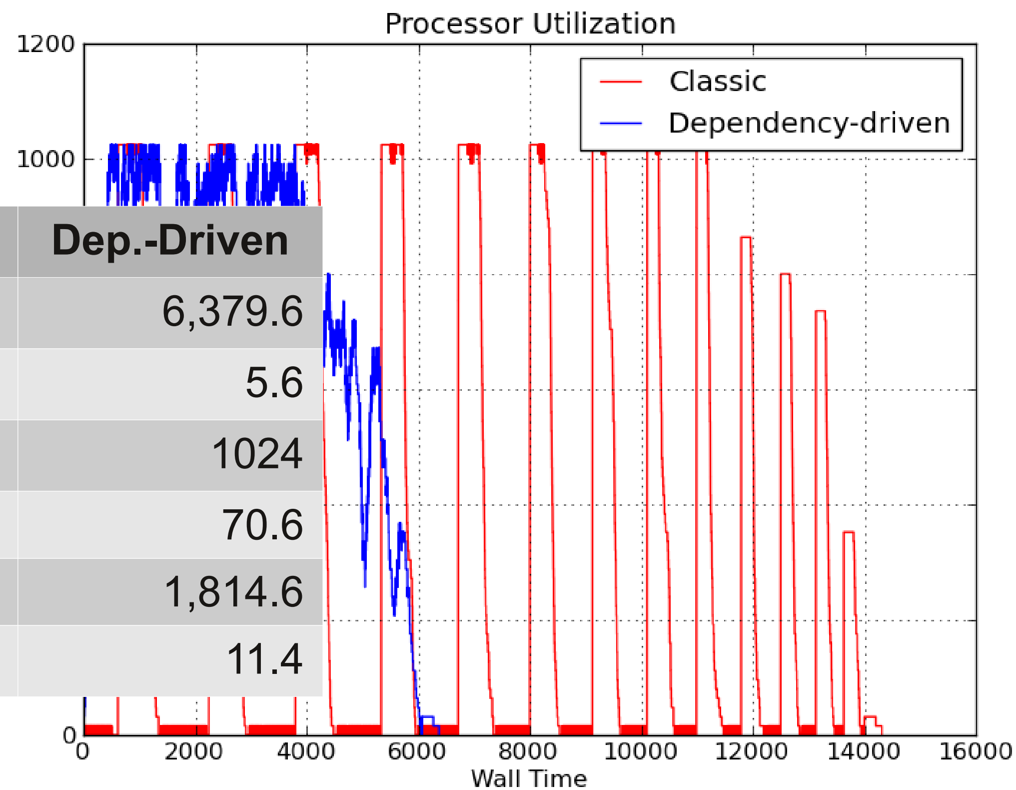


Dependency-Driven Parareal: Results

- Overlap the execution of multiple iterations
  - Up to 5 in this case
- Perform the same work done by the classic Parareal
  - Change only ***WHEN*** a task is executed
- Effective utilization: **70.6%**

Coarse Task Started
Fine Task Started
Coarse Task Ended
Fine Task Ended

Dependency-Driven Parareal: Results



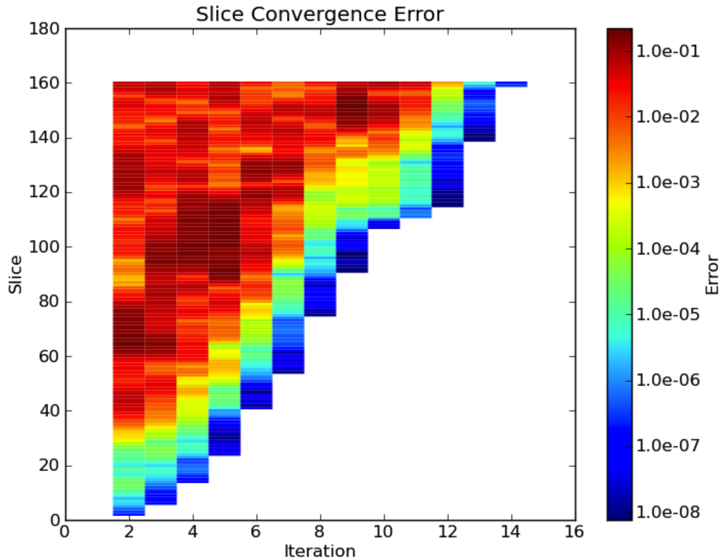
	Serial*	Classic	Dep.-Driven
Run Time (S)	35,704	14,330.8	6,379.6
Speed Up	1	2.5	5.6
Cores Used	16	1024	1024
Utilization %	100	31.6	70.6
Cost (Cpu H)	158.7	4,076.3	1,814.6
Relative Cost	1	25.7	11.4

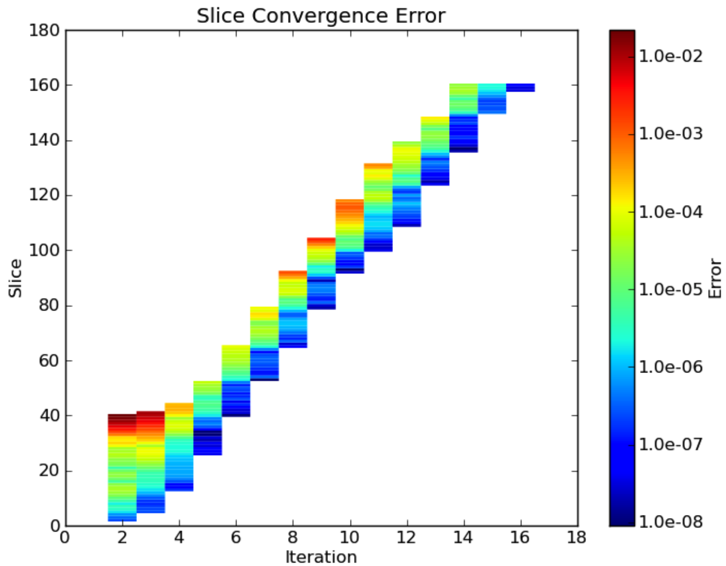
* *Estimated*

Dependency-Driven Parareal uses 44.5 % wallclock time of the classic version

Moving Window Parareal

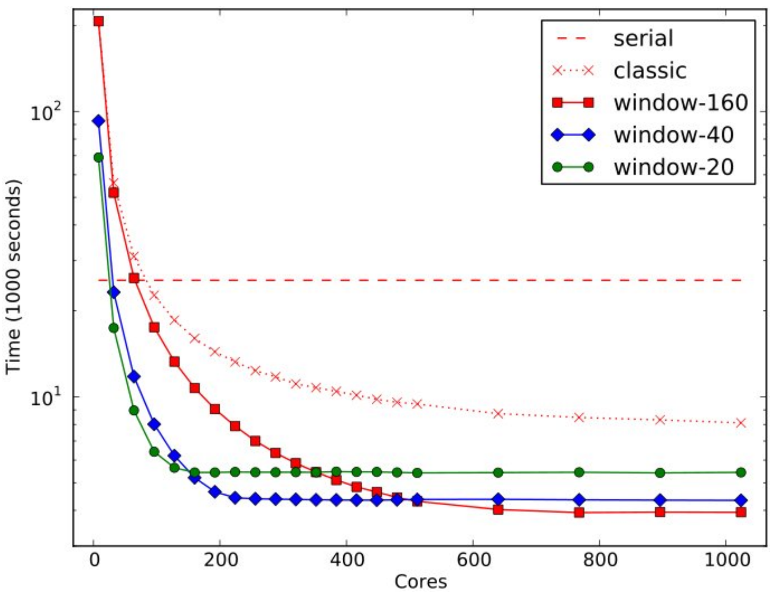
- Heat map shows convergence error per slice, per iteration (logarithmic color scale)
- Error in upper-left corner suggests limited reach of the coarse solver
  - Quality of coarse solution deteriorates as we move away from last converged result
- Optimize resource utilization by ***NOT*** executing those tasks
  - Implement a "***moving window***" version
  - Start with **n < N** slices
  - Add more, as slices converge
- Less work, less resources
  - But may need more iterations
- Introduces trade-off between window size, time to solution, and required resources



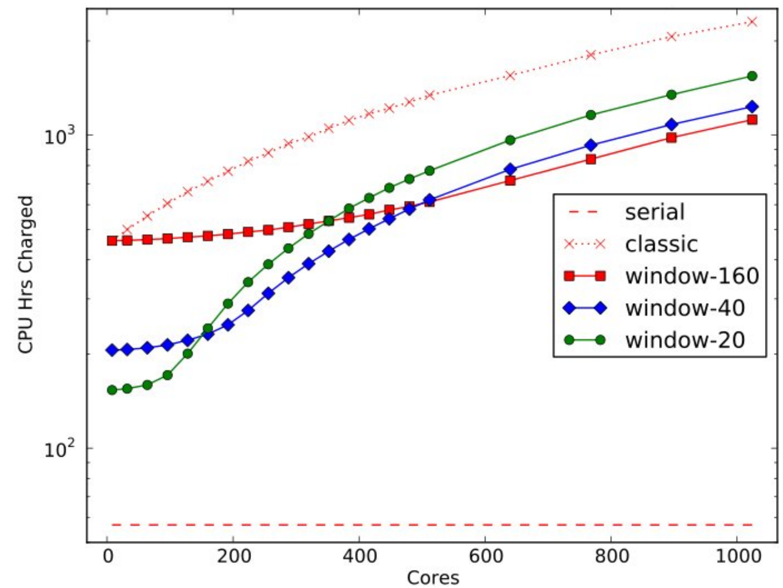


Exploring the Moving Window Parareal Trade-offs

Simulation Time



Cost



- Simulations used to explore the trade-off between resource utilization, window size, and time to solution for BETA
- Best choice depends on what's important (time, or cost ?)
- Moving window parareal allows users to choose the configuration that best meets their priorities

In Summary

- Parareal algorithm re-cast as a ***many-task problem*** executed within the IPS Framework
- ***Dependency-driven parareal*** improves resource utilization and reduces simulation time
- ***Moving window parareal*** avoids the performance of un-productive work and reduces over-all resource requirements
- Many-task implementation using IPS-parareal is:
  - Flexible – easily experiment with different coarse solvers
  - Retargettable – adapting to new problems takes less than a day's work
  - Currently being used to explore 1D MHD problems (***JOREK1D***), gyrokinetic (***GENE***) and fluid electrostatic turbulence (***TRB***)
  - Starting work on 3D MHD (***PIXIE3D***)
- *Dependency-Driven formulation should allow the use of slower (and better) coarse solvers, probably leading to faster convergence*
  - Subject of future study

Questions ?

For more info on the IPS, join us for the **PYHPC** paper

“***The Integrated Plasma Simulator: A Flexible Python Framework for Coupled Multiphysics Simulations***”,

Friday, 11:00 AM, TCC 102.

This work has been supported by the U.S. Department of Energy, Office of Science, Offices of Advanced Scientific Computing Research (ASCR) and Fusion Energy Sciences (FES). It has also been supported by by the ORNL Postmasters and Postdoctoral Research Participation Programs and the ORNL Higher Education Research Experiences Program which are sponsored by ORNL and administered jointly by ORNL and by the Oak Ridge Institute for Science and Education (ORISE). ORNL is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. ORISE is managed by Oak Ridge Associated Universities for the U.S. Department of Energy under Contract No. DE-AC05-00OR22750. The authors are grateful for grants of supercomputing resources at the University of Alaska, Arctic Region Supercomputing Center (ARSC) in Fairbanks.