Many-Task Applications in the Integrated Plasma Simulator

Samantha S. Foley*, Wael R. Elwasif*, David E. Bernholdt*, Aniruddha G. Shet*‡, Randall Bramley†
*Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA
†Dept. of Computer Science, Indiana University, Bloomington, IN, USA
‡Current address: Intel Research Laboratory, Bangalore, India
Email: {foleyss, elwasifwr, bernholdtde, shetag}@ornl.gov, bramley@cs.indiana.edu

***Abstract*—This paper discusses the Integrated Plasma Simulator (IPS), a framework for coupled multiphysics simulation of fusion plasmas, in the context of many-task computing. The IPS supports multiple levels of parallelism: individual computational tasks can be parallel, components can launch multiple tasks concurrently, tasks from multiple components can be executed concurrently within a simulation, and multiple simulations can be run simultaneously. Each level of parallelism is constructed on top of the many-task computing capabilities implemented in the IPS, the foundation for the parallelism presented at the multiple simulation level. We show that a modest number of simultaneous simulations, with appropriately sized resource allocations, can provide a better trade-off between resource utilization and overall execution time than if they are run as separate jobs. This approach is highly beneficial for situations in which individual simulation tasks may differ significantly in parallel scalability, as is the case in many scientific communities where coupled simulations rely substantially on legacy code.**

***Keywords*-coupled multiphysics simulation, many-task computing**

I. Introduction

High-fidelity simulation of physical phenomena is the core of what is traditionally considered to be high-performance computing (HPC). Such applications are typified by programs in which the parallel processes interact though a high-performance communication infrastructure such as MPI [1], in the "single-program multiple-data" (SPMD) style to model a single phenomenon of a physical system.

Lately, scientific interest and computing capability have been pushing the high-performance computational science community towards coupled multiphysics simulations, which permit even more faithful and more detailed modeling of many complex physical systems. Such simulations vary widely in their computational characteristics. Where the algorithmic coupling is strong [2], multiphysics simulations are often implemented as a single code in the SPMD style, similar to the simpler simulations above. Where the algorithmic coupling is weaker, it is often possible and desirable for there to be a greater separation between parts of the software handling different the physical phenomena. Generally speaking, such simulations are amenable to a "multiple-program multiple-data" (MPMD) implementation style, in which modules representing different physics can in principle be run concurrently, within the constraints imposed by the data dependencies (couplings) between them. Thus, individual multiphysics simulations that execute several tasks per time step and run for hundreds or thousands of time steps, can be viewed as many-task problems [3], albeit with significant constraints due to dependencies.

Finally, the pursuit of new scientific understanding with any simulation code typically involves many runs with different inputs for exploration, parameter sweep studies, uncertainty quantification, and many other purposes. At this level, there are generally few, if any, direct dependencies between different runs. To the extent that such runs can be enumerated *a priori*, they might be combined and treated as an even larger many-task computing (MTC) work flow, with the advantage that tasks from different simulations are unlikely to have dependencies.

Taken together, it becomes clear that many computational science problems can offer multiple levels of parallelism if the infrastructure is available to take advantage of it. In this paper, we consider how concepts from many-task computing can be brought together with traditional coupled multiphysics simulations to allow users to complete their simulations more quickly and at lower cost.

In this paper, we present the Integrated Plasma Simulator (IPS), an application framework for integrated fusion modeling, from an MTC perspective. We will describe the IPS as an infrastructure for coupled multiphysics applications and how that model enables its use for many-task style computing as well (Section II). We will then consider several scenarios for use of the multiple simultaneous simulation execution capability of the framework and explore how they enable more efficient use of computational resources and better time-to-solution, thus facilitating computational research (Section III). After placing this work in the broader context of computational science and many-task computing (Section IV), we draw some conclusions about our work (Section V).

II. The Integrated Plasma Simulator

The field of plasma physics has historically consisted of coupled simulations of relatively low-fidelity physical models, and HPC simulations of high-fidelity models of specific physical phenomena. Interest in high-fidelity "integrated modeling" has grown with the recognition that important phenomena observed in experimental fusion reactors involve coupled

978-1-4244-9705-8/10/$26.00

physics, and that computational capabilities are becoming available to make simulation of such phenomena tractable. This interest is heightened by ITER [4], an international collaboration to develop a tokamak fusion reactor to take the field's experimental capabilities to the next level.

The Center for Simulation of Radio Frequency Wave Interactions with Magnetohydrodynamics (SWIM) [5] is a project funded by the U.S. Department of Energy to study specific multiphysics fusion energy phenomena. It is one of three projects [6], [7] funded to explore fusion simulation as prototypes of a future Fusion Simulation Project, which will target comprehensive "whole device" integrated modeling. The computational goal of SWIM is to develop a framework to enable coupled multiphysics simulations to gain a better understanding of the behavior of and how to control plasmas for fusion energy production.

The SWIM project offers an interesting context for coupled multiphysics simulation; we anticipate that similar circumstances will be common in many areas as exploration of coupled multiphysics increases. Most of the physics applications that make up the coupled simulations of interest to SWIM have been in use and development for decades, and continue to be developed and used for other projects besides SWIM. Therefore, it is very important that the applications be changed as little as possible for SWIM. These physics applications also vary significantly in their parallel scalability, including codes which are strictly sequential, modestly scalable (tens of processors), and highly scalable (hundreds or thousands of processors). The starting point for the project has been to assume that loose physics coupling and weak algorithmic coupling [2] suffice for most problems of interest, with selective improvements as necessary.

To address the project's requirements, we developed the Integrated Plasma Simulator (IPS) [8]–[10], a lightweight component framework based on the concepts of the Common Component Architecture (CCA) [11], implemented in Python. IPS components satisfy a simple interface with `init()`, `step()`, and `finalize()` as the primary methods. IPS components are unmodified physics application executables wrapped with Python. The wrapper, along with small "helper" executables, adapt an application's native inputs and outputs to the interface expected by the IPS. Components generally exchange small amounts of data (typically a few megabytes) via a "plasma state" file [12], a set of special purpose netCDF files containing plasma specific data and a Fortran library to access them. The IPS framework provides a small set of services to components, such as data (file) management, resource and task management, configuration information, and an event service (Figure 1).

IPS simulations are controlled by "driver" components written in Python. This approach was chosen to provide users with a familiar procedural way of expressing the simulation work flow with complete flexibility. IPS simulations are typically time-stepped, with multiple components run at each time step in sequences dictated by their data dependencies. Work flows may include timed events, such as power changes in the RF

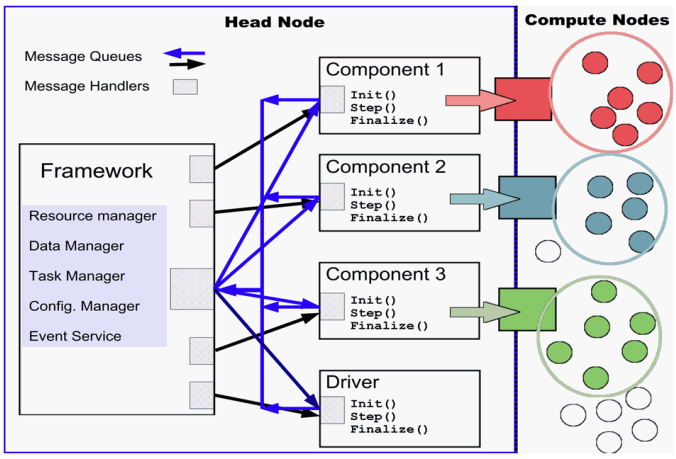

Fig. 1. An illustration of the relationships between the framework, components, services, and tasks in the Integrated Plasma Simulator (IPS).

subsystem, heating via beams of neutral particles, or fuel pellet injection. As the simulation progresses, the plasma may develop anomalies or instabilities which are not adequately represented by the primary components and must be treated specially.

The IPS is designed to execute in self-contained fashion as a batch job or (where permitted) interactively. The core of the IPS framework runs as a single Python process on a head node, service node, or in the compute partition, depending on the system architecture and the capabilities or limitations of each class of nodes (Figure 1). Components are spawned as separate processes, which in turn invoke the underlying physics codes. These typically parallel computational tasks are launched using appropriate mechanisms for the host system, such as `mpiexec` or `aprun` (Cray). The current version of the framework supports "concurrent multitasking" or MPMD-style execution through non-blocking versions of both component method invocations and launching of underlying physics tasks [9]. The IPS's resource manager (RM) tracks the utilization of the pool of nodes given to the job from the batch system so it can process resource requests from the task manager (TM) using a simple first-come first-served (FCFS) algorithm with a first-fit backfill policy [13], [14]. The task manager maintains a list of pending tasks made up of the task launch requests as they are submitted to the TM. Requests are processed and allocated resources in order until there are insufficient resources available for any pending task. Smaller tasks may run before larger ones that appear earlier in the list, however each release of resources triggers the traversal of the list from the beginning, ensuring that large tasks will be run as soon as enough resources become available. Data dependencies are implicitly defined in the driver component such that task launch requests will not be made until other dependencies are satisfied.

Additionally, multiple simulations may execute simultaneously in a single IPS invocation simply by specifying multiple input files on the command line. In this case, separate drivers

are instantiated for each simulation. From the viewpoint of the IPS task and resource managers, there is simply a larger set of runnable tasks to deal with. As before, as long as there are sufficient resources to run the largest parallel task in any of the simulations, the simulations will naturally interleave, and all will eventually finish, though there is no attempt at fairness among the individual simulations or tasks.

The IPS provides a rich environment in which a user may compose multiphysics applications with multiple levels of parallelism: individual computational tasks can be parallel, components can launch multiple computational tasks concurrently (allowing, for example, a physics code which evaluates one plasma flux surface at a time to be turned into a component which treats multiple surfaces concurrently), tasks from multiple components of a simulation can be executed concurrently, and multiple simulations can be run simultaneously. Each level of parallelism is constructed on top of the many-task computing capabilities implemented in the IPS RM and TM, the foundation for the parallelism presented at the multiple simulation level.

The original motivation for introducing MPMD capabilities into the framework had to do with resource utilization. In cases where the parallelism of simulation components varies significantly, computational resources may be left idle when "smaller" tasks are running. The ability to run multiple tasks concurrently whenever data dependencies permit provides opportunities for other tasks to "fill in the gaps". The approach of running multiple simultaneous simulations in the IPS offers users a simple but effective way to take advantage of this feature without having to worry about the unfamiliar (to most physicist users) complexities of writing MPMD drivers for individual simulations that properly account for all dependencies.

In the following section, we examine in more detail the multiple simulation capabilities of the Integrated Plasma Simulator, and explore how they can be used to increase the efficiency of resource utilization and reduce the overall time to solution for integrated fusion simulation.

III. Many-Task Execution with the IPS

As mentioned above, the primary goal for the introduction of many-task capabilities into the IPS was to improve resource utilization. The minimum number of processors required to carry out a given simulation is based on the parallelism of the largest task launched by any participating component. In a batch environment, jobs are allocated (and charged for) a fixed pool of resources for the duration of the job, so when tasks with a smaller degree of parallelism are executing, the remaining resources in the pool sit idle. While some degree of inefficiency may be acceptable as a trade-off for the new scientific capabilities provided, it is obviously preferable to do better. One obvious approach is to increase the scalability of the tasks launched by the bottleneck components. Unfortunately, this approach is often infeasible for scientific, technical, or even financial reasons. By running multiple simulations simultaneously, there are more tasks available to utilize otherwise idle nodes. In such a scenario, if a single task dominates the pool so that there are not enough nodes to run other tasks, they will wait for the large task to terminate and release its resources. In this way, the multiple simulations will naturally interleave their execution, making more effective use of the resource pool. The actual interleaving depends on the details of each simulation involved (the parallel sizes and execution times of each task), the size of the resource pool, and the exact timing (due to variations in task execution times, for example).

A. SWIM Simulation Scenarios

Our goal in this section is to illustrate the effectiveness of this approach, and to try to derive some general rules for how to arrange multiple simulation runs to maximize the overall resource utilization and minimize the average time to solution for each simulation. We have chosen two simulation scenarios, representative of scientifically-relevant studies currently underway in the SWIM project to provide concrete examples. Because these represent only two points in the vast space of simulations that *could* be run with the IPS, we also study how a number of variations of these scenarios affect the results. While this is by no means exhaustive, it allows us to extract some basic rules of thumb for the effective use of the IPS simultaneous simulations capability.

We have chosen two simulation scenarios based on the observed time and resource utilization of real scientific workloads in use to investigate plasmas in the planned ITER tokamak. The two workloads employ four different physics codes:

- TSC [15], [16] is a serial code that evaluates plasma equilibria and transport phenomena;
- NUBEAM [17], [18] evaluates heating of the plasma by injection of beams of neutral particles using a Monte Carlo algorithm, making it highly scalable with a minimum of approximately 500 particles per process;
- TORIC [19] and AORSA [20] evaluate the effects of radio-frequency waves on plasmas using different approaches, and with differing degrees of scalability (AORSA being higher fidelity and more scalable than TORIC).

Table I shows the execution characteristics for the two scenarios for a single physics time step. The number of time steps in a simulation can vary significantly depending on the specifics of the scientific study, but for example, the TNT scenario is being used to simulate an 1800s ITER discharge with 1s time steps. In this work, we uniformly use 100 time steps to adequately sample the interleaving of up to six simultaneous simulations. Each configuration is run ten times and average results are presented.

In order to get a sense of the many-task behavior of other simulations in the IPS, we vary the execution time and number of processors used by the NUBEAM task in both simulation scenarios. (Note that in the TNT case, NUBEAM is the task with the most parallelism, while in the ANT case it occupies the middle spot.) Since NUBEAM is a particle-based Monte Carlo code, this is analogous to changing the number of particles and process count in various ways, though we do not

TABLE I
SWIM SIMULATION SCENARIOS TNT AND ANT USED IN THIS WORK. TASKS ARE LISTED IN ORDER OF EXECUTION. TASK TIMINGS AND STANDARD DEVIATIONS ARE BASED ON ANALYSIS OF REAL SIMULATION RUNS ON A LARGE SHARED-MEMORY SYSTEM (STIX) AT THE PRINCETON PLASMA PHYSICS LABORATORY [21] FOR THE TNT SCENARIO, AND ON THE CRAY XT4 (FRANKLIN) AT THE NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER (NERSC) [22] FOR THE ANT SCENARIO.

Task	Processes	Time (s)
Scenario TNT		
TORIC	4	97 ± 2
NUBEAM	16	115 ± 15
TSC	1	130 ± 40
Scenario ANT		
AORSA	1024	1020 ± 5
NUBEAM	512	1020 ± 300
TSC	1	130 ± 40

claim that such variations are scientifically relevant—we are using them merely to sample the broader space of simulations possible in the IPS. In this paper, we examine only the topmost level of parallelism described in Section II, so within each simulation, the tasks are run strictly in sequence (shown in Table I).

B. Resource Usage Simulator

A Resource Usage Simulator (RUS) for the IPS was developed to explore and evaluate the behavior of interleaving many-task simulations quickly and without undue cost. The RUS consists of a resource manager and a MPMD execution simulator which presents tasks from each of the simultaneous simulations in the appropriate order and timing. The resource manager uses the same algorithm and policies as the IPS RM, but presents an interface appropriate to the RUS.

The RUS allows task run times to vary from one invocation to the next randomly, sampled from a Gaussian distribution based on standard deviations supplied with the task timings. We have run the experiments with and without such variations, and observe no substantive difference in the results, but for realism we present the results *with* timing variations.

Another feature of both the IPS and the RUS is the ability to handle resource allocation in multicore environments realistically. Typically, batch-managed HPC systems use *nodes* as their smallest unit of resource allocation, where each node comprises one or more processor cores per node (`ppn`). Computational tasks may utilize these cores as the wish, however only one task can run on each node. For example, a serial task like TSC will be assigned an entire node, regardless of the `ppn`, leading to inefficiencies imposed by the system policies. In this work, the Cray XT4 with nodes containing a single quad-core processor and the tasks that utilize one MPI process per *core* are modeled. The RUS and IPS RMs can handle any granularity of resource allocation through appropriate configuration parameters, however this capability is not the subject of this work.

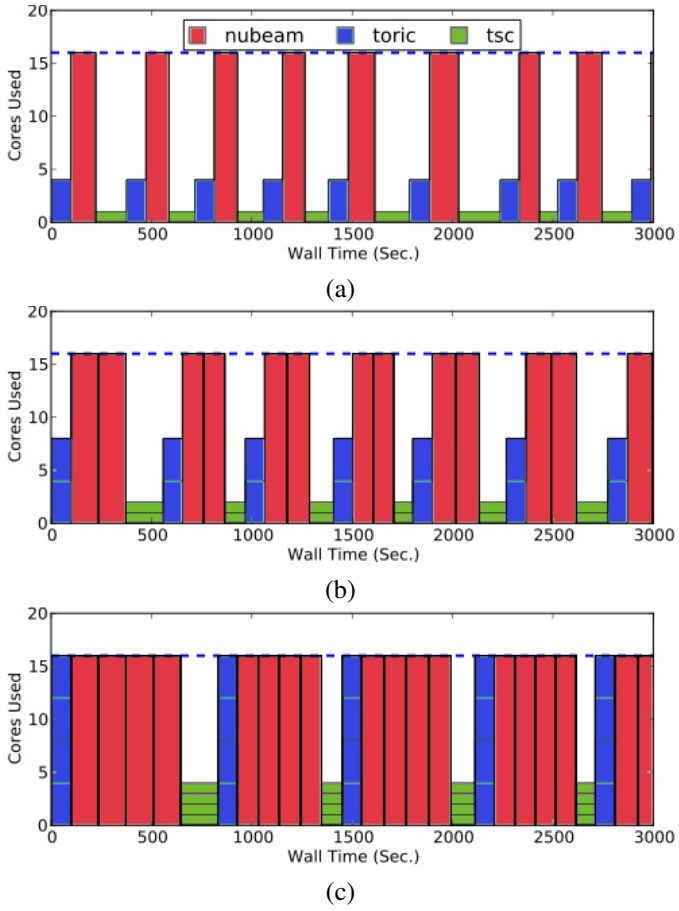

Fig. 2. Number of cores in use as the simulation progresses for the TNT scenario run in a 16-core pool for one (a), two (b), and four (c) simultaneous simulations.

C. Illustrating Many-Task Execution in the IPS

We begin with a simple illustration of how multiple simulations interleave to understand the impact on overall resource utilization and the average time to solution for a simulation. Figure 2 shows the resource utilization over time for the first 3000s of the TNT scenario running on 16 cores with different numbers of simulations running simultaneously.

In the single simulation case (Figure 2a), as each task runs in succession, the majority of cores are idle and the overall resource utilization (the integral of processes used productively over time compared to the total allocation) is only 43%. If instead two simulations run concurrently in the same 16 core resource pool, Figure 2b depicts how tasks from the different simulations run concurrently, leading to a much better resource efficiency of 64%. Finally, running four simulations in the same size pool, Figure 2c shows an efficiency of 86%.

It is worth noting how many simulation steps are completed in the 3000s window shown in these graphs. A single simulation completes eight steps and starts a ninth. In the two simulation case, each completes six steps, or a total of 12 simulation steps. When four simulations are running, each completes four steps in the same period, but the total is 16 simulation steps. In other words, because the multi-

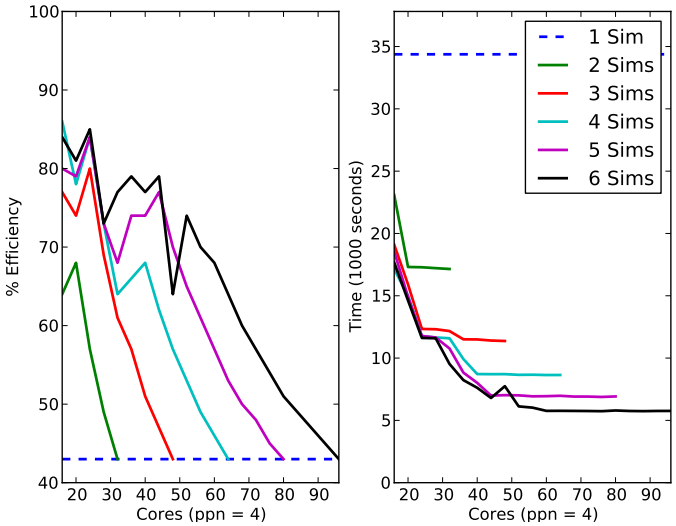

Fig. 3. Resource efficiency (left) and average time per simulation (right) for many-task execution of the TNT scenario.

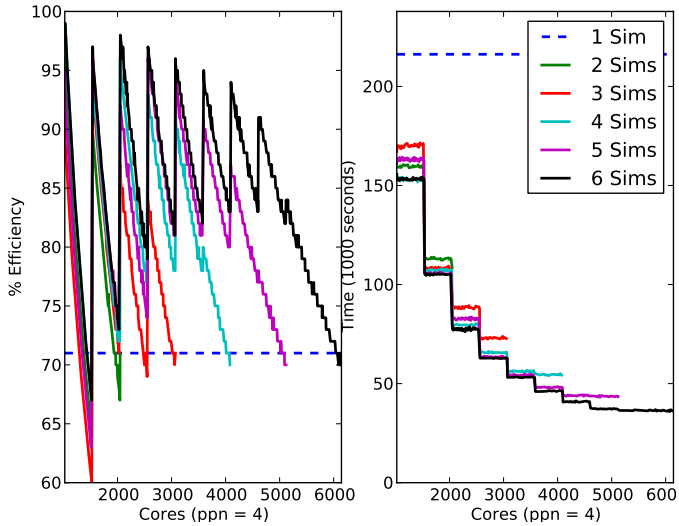

Fig. 4. Resource efficiency (left) and average time per simulation (right) for many-task execution of the ANT scenario.

simulation runs use resources more efficiency, the physics simulations are completed sooner than if they had each been run in succession as single-simulation runs. In this way, multi-simulation execution also provides better "average time to solution" on a per simulation basis.

While these examples have focused on a fixed pool of 16 cores, clearly the size of the resource pool can also be varied, which may make it easier (or harder) for tasks from different simulations to interleave effectively. Thus the user has the opportunity to significantly increase resource utilization efficiency, and tune the average time to solution using the RUS to find the appropriate "sweet spots".

D. Behavior of Many-Task Execution of the TNT and ANT Scenarios in the IPS

To better understand how these factors interact, the RUS is used to systematically examine the consequences of varying the size of the resource pool and the number of simultaneous simulations on the resource utilization efficiency and the average time to solution. We chose to limit the number of simultaneous simulations to a maximum of six because in many phases of a scientific investigation in a project like SWIM, the investigator can only formulate and manage a handful of different jobs at once. As we will see, even small numbers of simulations allow very high resource utilizations, though it may be possible to do even better with more tasks.

Figure 3 depicts the resource utilization efficiency (left) and the average time to solution (right) for the TNT scenario as we vary the size of the resource pool. The first feature to note is that interleaved execution for all cases (2–6 simulations) provides both better efficiency and lower time to solution (per simulation) than the baseline single simulation for all resource pools (up to the maximum useful pool size for each case).

Efficiencies vary non-monotonically with the size of the pool because fixed sizes and durations of the tasks do not always lead to better interleaving as resources are added. The same effect can be observed as plateaus in the time to solution. Although the behavior for different numbers of simulations varies, many of the peaks (local maxima) in the efficiency curves fall at the same core counts. In this case, we observe that the four simulation case starts with 86% efficiency at 16 cores. Curves for 3–6 simulations also show peaks of 80% or greater efficiency at 24 cores, which is sufficient to run one NUBEAM, one TORIC, and one TSC instance simultaneously. Further peaks in the 4–6 simulation curves, variously occurring at 36, 40, 44 and 52 cores correspond to the ability to run two NUBEAMs and several instances of TORIC and TSC simultaneously, or three NUBEAMS and one TORIC or TSC. The two simulation curve, while significantly more efficient than a single simulation, does not perform nearly as well as 3–6 simulations, with a single peak of 68% efficiency at 20 cores (one NUBEAM and one TORIC instance).

The ANT scenario shows generally similar characteristics (Figure 4), though there is less differentiation of the curves for different numbers of simulations. The single simulation run gives a resource efficiency of 71%. The peaks are generally in excess of 90% efficiency, and some are up to 99%. Peaks for different numbers of simulations generally coincide at the same numbers of cores. The peaks occur at core counts such as 1028, 1032, 1540, 2052, 2564, and 3076, corresponding to multiples of the sizes of the three tasks (note that TSC is allocated an entire four-core node). The 1540 core peak, for example, corresponds to one AORSA, one NUBEAM, and one TSC instance running simultaneously.

E. Exploring Variations in Run Time and Parallelism

In order to begin to get a sense of the behavior we might expect to see in other simulation scenarios beyond the two we have focused on so far, we consider the effect of systematically varying the run time and parallelism of various tasks in the TNT and ANT scenarios. The efficiency and time to solution curves produced by these experiments are generally similar

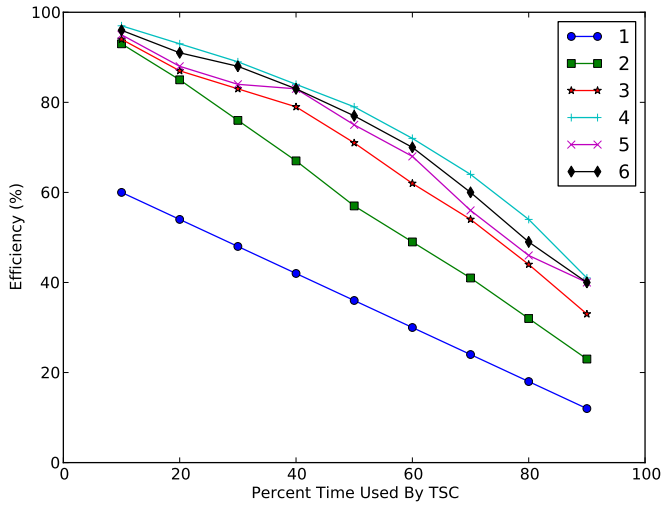

Fig. 5. Peak resource utilization efficiency as the execution time of TSC is varied in the TNT scenario.

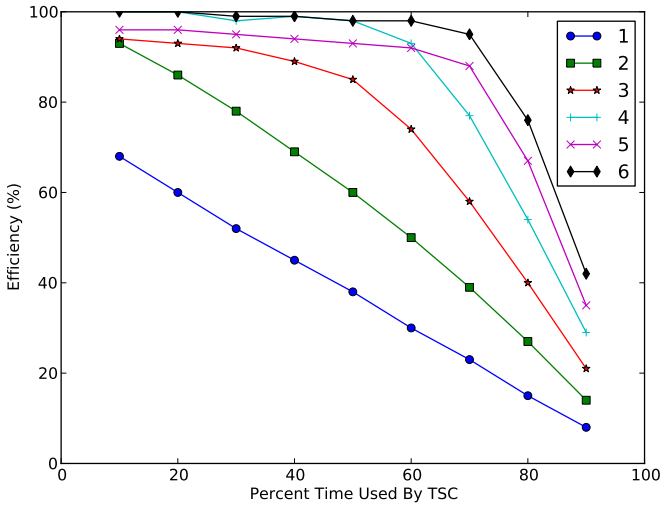

Fig. 6. Peak resource utilization efficiency as the execution time of TSC is varied in the ANT scenario.

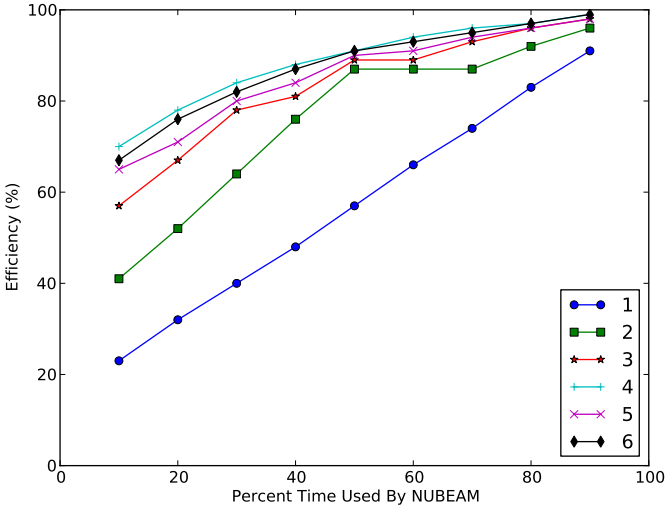

Fig. 7. Peak resource utilization efficiency as the execution time of NUBEAM is varied in the TNT scenario.

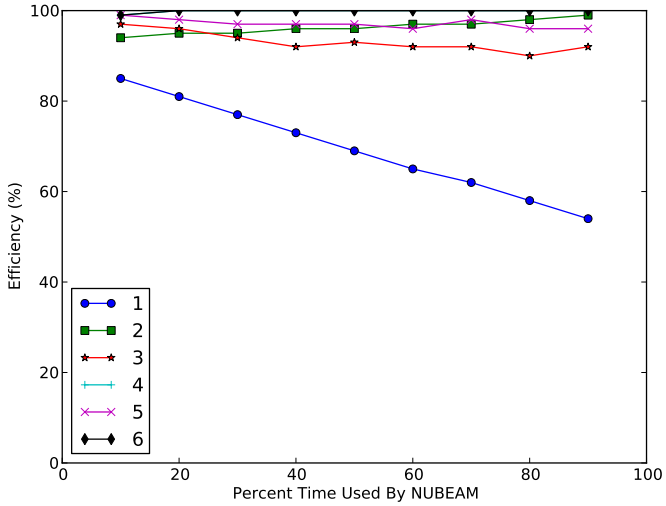

Fig. 8. Peak resource utilization efficiency as the execution time of NUBEAM is varied in the ANT scenario.

to those we have presented in the previous section, except of course that the specific locations and heights of the peaks will be different. Since we know it is always possible to identify specific configurations that will provide high efficiency (or low time to solution) for a given case, our focus this section is on how the variations in the simulation scenarios effect the peak efficiencies attainable with different numbers of simultaneous simulations.

Our first experiment is to vary the run time of TSC, the smallest task in both scenarios. Keeping the absolute time for the other two tasks in each scenario fixed, we vary the timing for TSC to range from 10% of the total time for a time step to 90%. Figures 5 and 6 show the peak achievable efficiency as a function of the time spent in TSC based on the TNT and ANT scenarios, respectively. Not surprisingly, as the task with the least parallelism increasingly dominates the overall time required for each step, the harder it is to effectively interleave the simulations, and the achievable efficiency drops. In the ANT case, running more simulations allows the peak efficiency to remain high longer, but the drop becomes more precipitous. This is because the ANT case has a much larger difference in parallelism between TSC and the other tasks.

A similar experiment varying the execution time of NUBEAM instead has a different impact on the two scenarios. In the TNT case, NUBEAM is the task with the most parallelism. As it increasingly dominates the simulation time step, Figure 7 shows that the peak achievable efficiency rises. For the ANT case, however, NUBEAM is the middle-sized task, and Figure 8 shows very different behavior. While the single simulation efficiency drops steadily as NUBEAM's fraction of the time step increases, the multiple simulation cases are able to maintain high efficiencies throughout. Because NUBEAM happens to be allocated precisely half the number of cores as AORSA, pairs of NUBEAM tasks easily interleave with

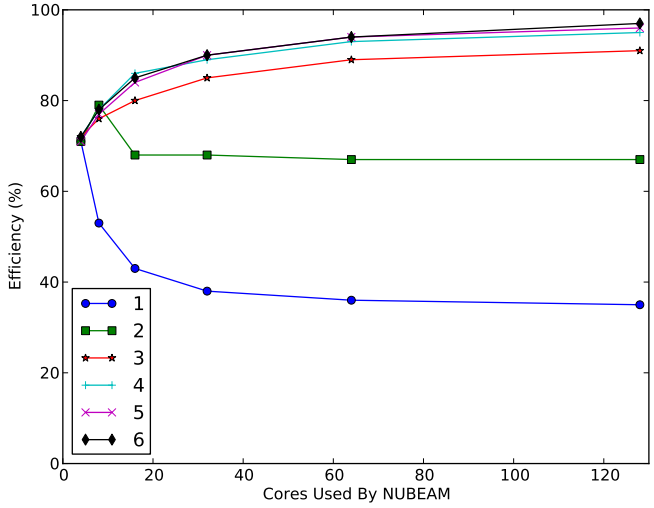

Fig. 9. Peak resource utilization efficiency as the parallelism of NUBEAM is varied (weak scaling) in the TNT scenario.

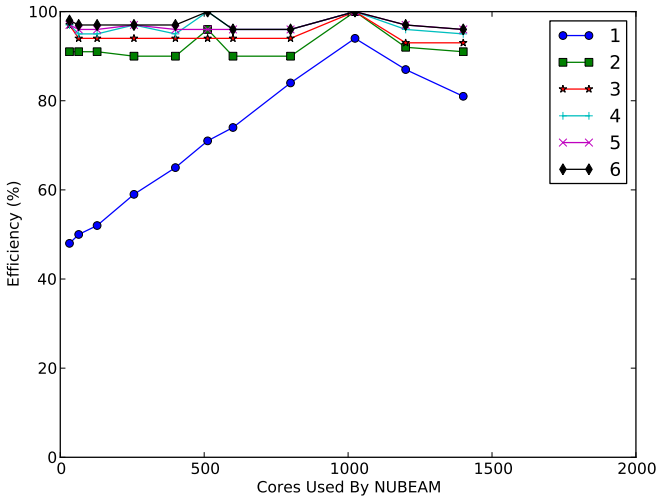

Fig. 10. Peak resource utilization efficiency as the parallelism of NUBEAM is varied (weak scaling) in the ANT scenario.

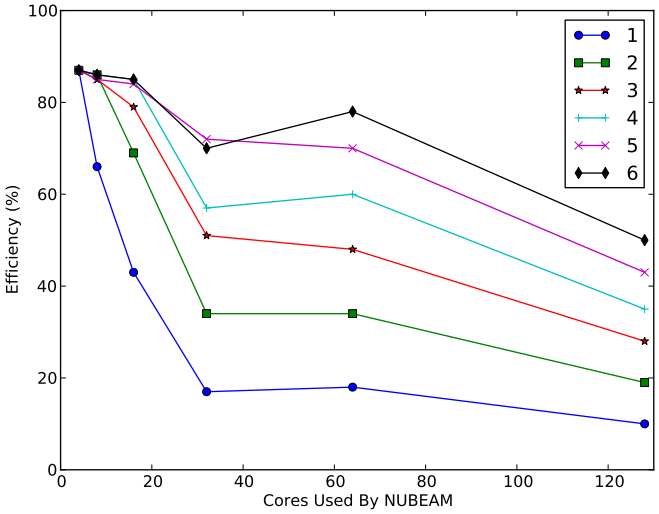

Fig. 11. Peak resource utilization efficiency as the parallelism and execution time of NUBEAM are varied (strong scaling) in the TNT scenario.

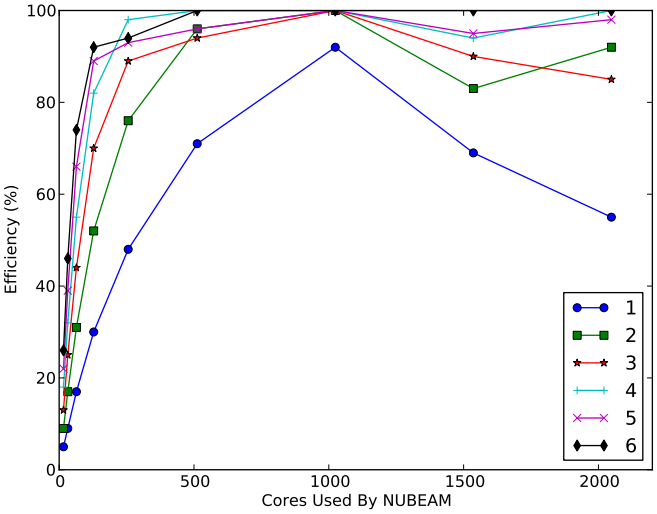

Fig. 12. Peak resource utilization efficiency as the parallelism and execution time of NUBEAM are varied (strong scaling) in the ANT scenario.

AORSA tasks. In fact, near the right side of the graph, we can see how odd numbers of simultaneous simulations have somewhat lower efficiency than the even numbers as the pairing of NUBEAM tasks becomes increasingly important to successful interleaving.

Another direction to explore involves variation in the parallelism of selected tasks within the simulation scenario. Since NUBEAM is a Monte Carlo code, and therefore readily scalable, we consider how such changes would affect achievable efficiencies. We can imagine both weak and strong scaling scenarios. In the strong scaling scenario, we keep the work (number of particles in NUBEAM) constant, so that increasing the number of parallel processes correspondingly decreases the execution time. In the weak scaling scenario, work is added (more particles) to keep the execution time constant as the number of processes is increased.

In the TNT scenario, weak scaling of NUBEAM decreases the efficiency of the single simulation case as NUBEAM becomes larger (Figure 9). Interleaving simulation execution moves from no increase in efficiency when each task runs on a single node, to significantly better resource efficiency, over 90% for 32 or more cores. However, three or more simulations yields significantly better efficiencies than the single simulation and even two simulation case, due to the similar running times of the three tasks. In the ANT scenario, the two large tasks, AORSA and NUBEAM are conveniently matched in execution time and differ by a factor of two in parallelism. In the weak scaling case, the execution time remains matched with AORSA's, while the number of cores consumed varies. In Figure 10 we see that the single-simulation efficiency peaks when NUBEAM uses the same number of cores as AORSA (1024). For two or more simultaneous simulations, we see peaks at 512 and 1024 cores, but interestingly, the efficiencies are uniformly high ($> 90\%$) throughout the range of sizes

studied for NUBEAM. Looking at the detailed data, we see that the peak efficiencies shown on the graph are occurring at core counts that allow one or more NUBEAM instances to run together with one (or more) AORSA instance, just as we observed for the basic ANT and TNT scenarios, above.

Strong scaling of NUBEAM in the TNT scenario also leads to decreased efficiency for the single-simulation case, however the multi-simulation curves generally decrease over time, as NUBEAM's runtime becomes so small, yet its parallelism is so large (Figure 11). Interestingly, the efficiency of the single and multiple simulation cases are the same when all tasks use one node, but do not get 100% efficiency. This is because TSC is unable to use its three other cores. For the strong scaling variation of the ANT scenario, the single-simulation curve in Figure 12, we see that peak efficiency rises rapidly, passing through the baseline ANT scenario (512 cores, 1020s), and achieving an efficiency of 92% when NUBEAM is the same size as AORSA (1024 cores), even though the run time is half as much (520s). The multiple simulation curves show similar behavior, but with better peak efficiencies. Beyond 1024 cores, the single-simulation curve starts to decline, along with the multiple simulation curves. At this point, NUBEAM is dominating in resources, but takes significantly less time than AORSA, which matches the behavior of the strong scaling of NUBEAM in the TNT case.

For both scenarios, it appears that having tasks of comparable size (parallelism) is more beneficial than having comparable execution times. However, scenarios with larger tasks can tolerate execution time mismatches better than those with smaller tasks. We also note that even numbers of simultaneous simulations tend to perform better than odd numbers as the size of NUBEAM increases, similar to what we observed when varying the execution time of NUBEAM with fixed parallelism (Figure 8).

F. Discussion

The results clearly show that running multiple independent simulations simultaneously within a single IPS invocation is an effective way to boost resource utilization efficiency for simulation scenarios that may have wide disparities in execution time and/or parallelism among the tasks, as can easily happen in multiphysics simulations.

Moreover, high levels of efficiency may be achieved with a small number of simultaneous simulations. Typically, interleaving three or four simulations is sufficient to obtain the best efficiency, and even two simulations are often close. We consider this an important finding for the practical use of this approach. While there are cases where it is possible for a researcher to formulate large numbers of simulations at once, such as a parameter sweep study, or a sensitivity analysis, scientific investigations often progress in an incremental fashion, in which the researcher can only formulate a small number of simulations at a time, the results of which must be analyzed and understood before proceeding to the next step.

It is generally true in the scenarios we have studied that running more simulations simultaneously gives rise to higher efficiencies. This might be considered a logical consequence of the fact that more simulations provide more tasks to help utilize idle cores. However it is not hard to find specific counter-examples to this rule of thumb. For example, in Figure 3 at 16 cores, four simulations are more efficient than either five or six, and at 48 cores, five simulations are more efficient than six. This appears to be result of specific details of the simulation scenario allowing certain combinations of tasks to fit together particularly well. On the other hand, the differences in efficiency at such points is generally not large, and we reiterate the point that nearly all multi-simulation configurations give better efficiency than a single simulation.

If the user is more concerned about the time to solution for a set of simulations than their resource utilization efficiency, it is useful to note that the same interleaving phenomena that allows multiple simulations to use resources more efficiently also allows those simulations to complete in timeframes that may be significantly shorter than if they were run individually, as illustrated in the time to solution graphs in Figures 3 and 4.

Another important observation from these results is that high levels of efficiency can be achieved with relatively modest core counts. In the experiments we have conducted, resources sufficient for one instance of the largest task together with one or more instances of the second largest task generally suffice for several simulations to interleave effectively. Adding more cores in order to run multiple instances of the largest task concurrently typically also yields local maxima in the efficiency curves, but in many cases not as good as for smaller resource allocations.

As small or sequential tasks come to dominate the execution time for a step, it is harder to effectively interleave their execution with shorter-running tasks, and efficiencies suffer regardless of the number of simulations. However if the other tasks are significantly more parallel, as in the ANT scenario (512 and 1024 cores), multiple simulations can act as a buffer, keeping peak efficiencies higher, before a more precipitous decline.

Finally, parallel scaling studies suggest that it is more beneficial to have tasks with similar size requirements than to have similar execution times. This is potentially useful information for the construction of efficient simulation scenarios when the user has a choice of trading off time and parallelism for some of the tasks. Since this is a very large problem space, it is a good target for further study to refine these guidelines.

IV. Related Work

In the short time since the term "many-task computing" was coined [23], it has been applied to a fairly wide range of computational approaches. The majority of papers in this area have focused on pools of related but (mostly) independent tasks which are easily generated, often numbering in the many thousands, and in some cases millions or more. These tasks are executed in HPC [23], [24] or distributed environments [25]–[27] with the help of separate middleware to manage workflow, scheduling and resource management, and other needs [24],

[28]–[30]. Our work, on the other hand, involves tasks with significant dependencies (time-stepped simulations) and the execution of modest numbers of simulations concurrently, all carried out within a single invocation of a component framework, running within a single batch job on an HPC system. The task-based parallel linear algebra framework by Song et al. [31] is more closely related to our work in the sense of many-task problems with significant dependencies. A number of other many-task papers involve individual tasks which are (potentially) substantial and parallel in their own right, such as computational fluid dynamics [32], bioinformatics [25] and ocean acoustics [27].

Nimrod/K [33] and DAKOTA [34] are examples of external drivers which can be used to manage parameter sweep, optimization, sensitivity analysis, and uncertainty quantification studies. Such tools could be used with the IPS, but based on the results shown here, would be more efficient if the capabilities were integrated into the IPS so that multiple simulations could interleave in the same resource pool. We are investigating the potential to integrate DAKOTA with the IPS, for this purpose.

The SWIM IPS is one of the growing number of coupled multiphysics frameworks being developed across the computational science community. Other efforts in plasma physics include the Japanese TASK framework [35] and the European Integrated Tokamak Modeling (ITM) [36] framework, as well as the US-based Center for Plasma Edge Simulation (CPES) [7], [37] and the Framework Application for Core-Edge Transport Simulations (FACETS) [6], [38]. Broadening to other domains, there are too many examples to cite. The current state of the art in coupled multiphysics is that each framework typically utilizes a different software architecture, with different approaches to coupling the components and different execution models. To our knowledge, the IPS's ability to carry out multiple simulations simultaneously in a single framework invocation is unique.

V. Conclusions

We have presented the SWIM project's Integrated Plasma Simulator (IPS), a framework for coupled multiphysics simulations of fusion plasmas, which supports multiple levels of parallelism. This work focuses on the ability to run multiple independent simulations simultaneously in a single invocation of the framework, which provides a many-task computing capability to IPS users as a small extension to support for MPMD parallelism used in other contexts. We believe that this kind of integration is central to the ability to effectively carry out many-task computing for complex multiphysics simulations comprising multiple tasks with different parallel scalabilities.

For simulation scenarios relevant to the SWIM project, we have shown how this approach provides an effective tool to obtain much higher efficiency using the allocated computing resources than is possible running one simulation at a time. Generally, small numbers of simultaneous simulations running in resource allocations that are only modestly larger than the minimum required to run a single simulation suffice to provide these improvements.

This experience with many-task computing in the IPS gives rise to a number of different opportunities for future work. First, we have sampled but a tiny subset of the possible of simulation scenarios that can be carried out in the IPS environment. We would like to expand this exploration, emphasizing new simulation scenarios that SWIM (or other projects that might use the IPS) develop.

Second, the Resource Usage Simulator (RUS) has been a simple but useful exploratory tool in this work, but it could be extended in several ways. With the addition of an optimization driver, it could be turned into a tool that would allow users to quickly provide the optimal configuration for specific user-supplied simulation scenarios. Also, in real simulations, it is not uncommon for the execution time of components to vary systematically as the simulation progresses, which cannot currently be modeled in RUS.

Finally, as we mentioned earlier, the IPS currently uses a simple FCFS scheduling algorithm with a first-fit backfill policy. There may be other scheduling algorithms and policies that would provide better interleaving of multiple simulations in challenging situations.

Acknowledgments

We would like to thank Steve Jardin for the suggestion that led to this work.

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 Eduction 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.

References

[1] Message Passing Interface Forum, "MPI: A message-passing interface standard version 2.2," http://mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf, September 4 2009.

[2] R. Hooper, M. Hopkins, R. Pawlowski, B. Carnes, and H. K. Moffat, "Final report on LDRD project: Coupling strategies for multi-physics applications," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Sandia Report SAND2007-7146, August, 26 2008.

[3] I. Raicu, I. Foster, and Y. Zhao, "Many-Task Computing for Grids and Supercomputers," in *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08)*. co-located with IEEE/ACM Supercomputing 2008, Nov 2008.

[4] "ITER: The way to new energy," http://www.iter.org.

[5] "Center for Simulation of RF Wave Interactions with Magnetohydrodynamics," http://cswim.org/.

[6] "Framework for Core-Edge Transport Simulations," http://facetsproject.org.

[7] "Center for Plasma Edge Simulation," http://www.cims.nyu.edu/cpes/.

[8] W. Elwasif, D. Bernholdt, A. Shet, S. Foley, R. Bramley, D. Batchelor, and L. Berry, "The Design and Implementation of the SWIM Integrated Plasma Simulator," in *18th Euromicro Int'l. Conf. on Parallel, Distributed and Network-based Processing (PDP)*, Pisa, Italy, 17–19 February 2010.

[9] S. S. Foley, W. R. Elwasif, A. G. Shet, D. E. Bernholdt, and R. Bramley, "Incorporating Concurrent Component Execution in Loosely Coupled Integrated Fusion Plasma Simulation," in *Component-Based High-Performance Computing (CBHPC)*, Karlsruhe, Germany, 16–17 October 2008.

[10] W. R. Elwasif, D. E. Bernholdt, L. A. Berry, and D. B. Batchelor, "Component Framework for Coupled Integrated Fusion Plasma Simulation," in *CompFrame '07: Proc. of the 2007 symp. on Component and framework technology in high-perf. and scientific computing*, Montreal, Canada, 21–22 October 2007.

[11] B. A. Allan, R. Armstrong, D. E. Bernholdt, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and S. Zhou, "A Component Architecture for High-Performance Scientific Computing," *Intl. J. High-Perf. Computing Appl.*, vol. 20, no. 2, pp. 163–202, Summer 2006. [Online]. Available: http://hpc.sagepub.com/cgi/reprint/20/2/163

[12] D. McCune, "Plasma State," http://w3.pppl.gov/ntcc/PlasmaState/.

[13] D. Lifka, "The ANL/IBM SP scheduling system," in *Job Scheduling Strategies for Parallel Processing.*, ser. IPPS'95 Workshop Proceedings. Berlin: Springer-Verlag, 1995, pp. 295–303.

[14] K. Aida, "Effect of Job Size Characteristics on Job Scheduling Performance," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000, vol. 1911, pp. 1–17. [Online]. Available: http://dx.doi.org/10.1007/3-540-39997-6_1

[15] S. C. Jardin, N. Pomphrey, and J. Delucia, "Dynamic modeling of transport and positional control of tokamaks," *J. Computat. Phys.*, vol. 66, no. 2, pp. 481–507, 2 October 1986.

[16] S. Jardin, M. Bell, and N. Pomphrey, "TSC simulation of Ohmic discharges in TFTR," *Nucl. Fusion*, vol. 33, no. 3, pp. 371–382, March 1993.

[17] R. J. Goldston, D. C. McCune, H. H. Towner, S. L. Davis, R. J. Hawryluk, and G. L. Schmidt, "New Techniques for Calculating Heat and Particle Source Rates Due to Neutral Beam Injection in Axisymmetric Tokamaks," *J. Computat. Phys.*, vol. 43, no. 1, pp. 61–78, 1981.

[18] A. Pankin, D. McCune, R. Andre, G. Bateman, and A. Kritz, "The tokamak Monte Carlo fast ion module NUBEAM in the National Transport Code Collaboration library," *Computer Phys. Comm.*, vol. 159, no. 3, pp. 157–184, 1 June 2004.

[19] J. C. Wright, P. T. Bonoli, M. Brambilla, F. Meo, E. D'Azevedo, D. B. Batchelor, E. F. Jaeger, L. A. Berry, C. K. Phillips, and A. Pletzer, "Full Wave Simulations of Fast Wave Mode Conversion and Lower Hybrid Wave Propagation in Tokamaks," *Phys. Plasmas*, vol. 11, pp. 2473–2479, 2004.

[20] E. F. Jaeger, L. A. Berry, E. D'Azevedo, D. B. Batchelor, M. D. Carter, K. F. White, and H. Weitzner, "Advances in Full-Wave Modeling of Radio Frequency Heated, Multidimensional Plasmas," *Physics of Plasmas*, vol. 9, no. 5, pp. 1873–1881, 2002. [Online]. Available: http://link.aip.org/link/?PHP/9/1873/1

[21] "PPPL Cluster Hardware," http://beowulf.pppl.gov/hw.html.

[22] "National Energy Research Scientific Computing Center," http://www.nersc.gov.

[23] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, and B. Clifford, "Toward loosely coupled programming on petascale systems," in *SC '08: Proc. of the 2008 ACM/IEEE conf. on Supercomputing*, 2008.

[24] P. Marshall, M. Woitaszek, H. M. Tufo, R. Knight, D. McDonald, and J. Goodrich, "Ensemble Dispatching on an IBM Blue Gene/L for a Bioinformatics Knowledge Environment," in *MTAGS '09: Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, Nov 2009.

[25] X. Qiu, J. Ekanayake, S. Beason, T. Gunarathne, G. Fox, R. Barga, and D. Gannon, "Cloud Technologies for Bioinformatics Applications," in *MTAGS '09: Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, Nov 2009.

[26] R. Costa, F. Brasileiro, G. L. Filho, and D. M. Sousa, "OddCI: on-demand distributed computing infrastructure," in *MTAGS '09: Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, 2009.

[27] C. Evangelinos, P. F. Lermusiaux, J. Xu, P. J. Haley, and C. N. Hill, "Many Task Computing for Multidisciplinary Ocean Sciences: Real-Time Uncertainty Prediction and Data Assimilation," in *MTAGS '09: Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, Nov 2009.

[28] I. Raicu, I. Foster, M. Wilde, Z. Zhang, K. Iskra, P. Beckman, Y. Zhao, A. Szalay, A. Choudhary, P. Little, C. Moretti, A. Chaudhary, and D. Thain, "Middleware support for many-task computing," *Cluster Computing*, vol. 13, no. 3, pp. 291–314, 2010.

[29] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[30] L. Hui, Y. Huashan, and L. Xiaoming, "A Lightweight Execution Framework for Massive Independent Tasks," in *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08)*, ser. co-located with IEEE/ACM Supercomputing 2008, Nov 2008.

[31] F. Song, A. YarKhan, and J. Dongarra, "Dynamic task scheduling for linear algebra algorithms on distributed-memory multicore systems," in *SC '09: Proc. of the 2009 ACM/IEEE conf. on Supercomputing*, 2009.

[32] E. Ogasawara, D. de Oliveira, F. Chirigati, C. E. Barbosa, R. Elias, V. Braganholo, A. Coutinho, and M. Mattoso, "Exploring Many Task Computing in Scientific Workflows," in *MTAGS '09: Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, Nov 2009.

[33] A. I. Abramson D, Enticott C, "Nimrod/K: Towards Massively Parallel Dynamic Grid Workflows," in *SC '08: Proc. of the 2008 ACM/IEEE conf. on Supercomputing*, Austin, Texas, November 2008.

[34] B. Adams, W. Bohnhoff, K. Dalbey, J. Eddy, M. Eldred, D. Gay, K. Haskell, P. Hough, and L. Swiler, "DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 5.0 User's Manual," Sandia National Laboratories, Tech. Rep. SAND2010-2183, December 2009.

[35] "TASK Code Home Page," http://bpsi.nucleng.kyoto-u.ac.jp/task/.

[36] "EFDA Task Force on Integrated Tokamak Modelling," http://www.efda-taskforce-itm.org.

[37] J. Cummings, J. F. Lofstead, K. Schwan, A. Sim, A. Shoshani, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and R. Barreto, "EFFIS: an End-to-end Framework for Fusion Integrated Simulation," in *18th Euromicro Int'l. Conf. on Parallel, Distributed and Network-based Processing (PDP)*, Pisa, Italy, 17–19 February 2010.

[38] J. R. Cary, A. Hakim, M. Miah, S. Kruger, A. Pletzer, S. Shasharina, S. Vadlamani, A. Pankin, R. Cohen, T. Epperly, T. Rognlien, R. Groebner, S. Balay, L. McInnes, and H. Zhang, "FACETS – a Framework for Parallel Coupling of Fusion Components," in *18th Euromicro Int'l. Conf. on Parallel, Distributed and Network-based Processing (PDP)*, Pisa, Italy, 17–19 February 2010.