Workflows for Performance Evaluation and Tuning

TR-08-03

Jeffrey L. Tilson
Mark S.C. Reed
Robert J. Fowler

May 5, 2008

Workflows for Performance Evaluation and Tuning

Jeffrey L. Tilson [1], Mark S.C. Reed [2], Robert J. Fowler[3]

Renaissance Computing Institute, University of North Carolina at Chapel Hill, U.S.A.

{[1]jtilson, [2]markreed, [3]rjf }@renci.org

***Abstract*— We report our experiences with using high-throughput techniques to run large sets of performance experiments on collections of grid accessible parallel computer systems for the purpose of deploying optimally compiled and configured scientific applications. In these environments, the set of variable parameters (compiler, link, and runtime flags; application and library options; partition size) can be very large, so running the performance ensembles is labor intensive, tedious, and prone to errors. Automating this process improves productivity, reduces barriers to deploying and maintaining multi-platform codes, and facilitates the tracking of application and system performance over time. We describe the design and implementation of our system for running performance ensembles and we use two case studies as the basis for evaluating the long term potential for this approach. The architecture of a prototype benchmarking system is presented along with results on the efficacy of the workflow approach.**

I. Introduction

There are a wide variety of High Performance Computing (HPC) system architectures deployed today, from commodity clusters to special purpose machines. In this environment, there are many ways of building, configuring, and running applications on each system. In this complex environment, manually running extensive performance experiments is infeasible and attempts to automate the process using an *ad hoc* methods based on custom scripts fall short. We have therefore begun to use a workflow framework to manage our benchmarking and performance tuning experiments. We report here on our approach and our experiences.

Many large-scale scientific applications have now outlasted several generations of high-end system architectures. These are now used across the full spectrum of currently active hardware and software environments. These codes can have many, possibly conflicting, configuration options with choices of solver methods, external libraries, and alternative algorithms. These choices potentially result in significant differences in performance for each target system.

Traditional benchmarking methods are not sufficient to handle the increasing complexity of rapidly evolving HPC systems as evidenced by the growth in chip complexity. Chips have gone from single-CPU systems to multi-processors, then multi-core, and now heterogeneous many-core systems that include external accelerator units. This complexity is exacerbated by the use of deep memory hierarchies and the mix of distributed and shared memory components resulting in performance receding further behind the memory wall. Yet, even these are not the final complications as growing hardware complexity is mirrored by the rapid growth in software choices including compilers, compiler options, communications libraries and other run-time configuration options. Finally, the rankings of these choices depend on the complexity and size of inputs as well as on system partition size. This proliferation of options is likely to increase in the future.

Thus, the overarching question becomes; given one or more large-scale HPC systems, what combination of these options should one select in the deployment of an application? Ideally, the analyst would test all the possible choices under conditions that mimic expected usage. Unfortunately, the number of combinations of these possible choices generally precludes manually performing such an analysis. Even taking a small-subset of likely important options can grow to a large number of trials. Thus the ability of the analyst to exhaustively, or even reasonably, optimize the deployment of the application or to state with confidence that the optimal choices have been made is difficult.

A related concept addressable by this work is the ongoing maintenance of such codes through periodic performance assurance testing especially across multiple platforms. This is a vital and often even more time-consuming process than the initial deployment.

In this paper, we report our explorations into techniques for improving the productivity of researchers responsible for the deployment and maintenance of multi-platform codes within a grid-services environment. In particular, we describe methods for facilitating performance studies in an environment that involves many platforms and configuration alternatives. We evaluate the relative strengths and weaknesses of our experimental prototypes.

II. Performance Campaigns

A performance campaign is defined in this work as a set of experiments across (possibly) multiple platforms. Each experiment is defined by a set of configuration options and test inputs. A campaign step runs an experiment by building a configuration on a system and then executing it on a specific input using a specified environment. For example, a parallel scaling study of an application on 64, 128, and 256 nodes, on three different clusters, and for two sets of compile or link parameters would require 18 separate campaign steps. Thus a campaign is such a set of campaign steps. This High ThroughPut (HTP) workflow automates the execution of the campaign. A highly simplified specification for a campaign step is shown in Table 1.

A. Base Cyberinfrastructure

Fig. 1 shows the structure of our performance workflow environment. The outer Workflow Layer enacts the performance workflow and serves as the user interface. Campaign specifications are entered at this layer by choosing

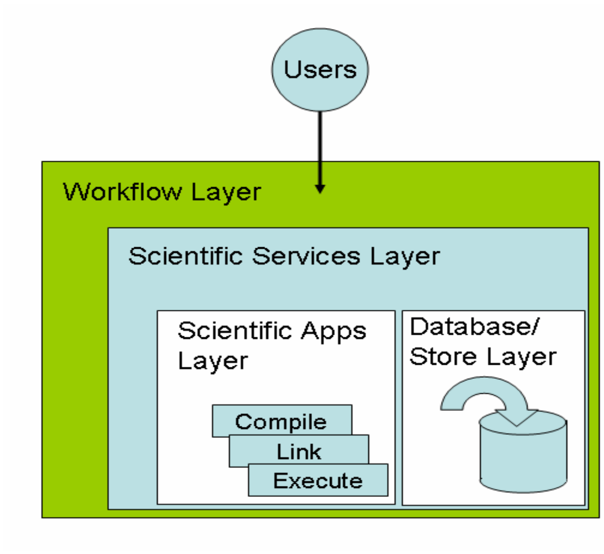


Fig. 1. Depiction of the layered structure of our performance campaign environment

a particular workflow and the configuration option ranges for the campaign. The specified workflow assembles the campaign from steps by enumerating combinations of configuration options. We use a standard workflow framework, so it is easy to create new workflows by adapting and extending existing ones. The case studies used here are adaptations of a *base* workflow to applications in Chemistry and in Storm Surge modelling. The Scientific Services Layer encapsulates the web and grid services that initiate the compilation, linking and running steps on (potentially) remote target systems. The Scientific Apps Layer consists of a set of scripts (Perl, Python, bash, etc.) installed on the *remote* machine to connect the workflow services to the native build/run processes of the scientific application under study.

TABLE 1

REPRESENTATIVE NAME-VALUE PAIRS FOR A CAMPAIGN STEP

CC	/opt/mpi/openmpi/intel/bin/ mpicc
FC	/opt/mpi/openmpi/intel/bin/ mpif77
FFLAGS	-O3 –xT –i8 –pg
NODES	8
CORESPERNODE	4
CAMPAIGNNAME	GamessNetworkTest
LAUNCHER	/opt/mpi/openmpi/gnu/bin/ mpirun --mca btl mvapi,self -verbose
TRACKNAME	Standard
CODE	Gamess

Select outputs from all parts of each campaign step are committed to persistent storage by sending them to a relational database provided by the Database Layer.

B. *Workflow Technologies*

Our system uses the Taverna Workbench [1] to define and instantiate workflows. We chose Taverna for this work because of the large number of included services and our substantial experience building Taverna-based workflows for biological applications.[2],[3] Though Taverna doesn't directly support grid services, the Scientific Services Layer provides mechanisms for using the standard Taverna system within a computational grid [4],[5] system.

The Taverna Workbench integrates many software tools, including web services, and provides a desktop authoring environment and enactment engine based on a centralized scheduling model. In particular, Taverna uses a centralized control approach to enact non-directed acyclic graphs.[6] This is a powerful approach permitting extensive use of implicit iteration and basic failure recovery modes.

C. *Service technologies*

The service technologies used to wrap, grid-enable, and invoke the scientific applications are implemented with the Generic Service Toolkit (GST).[7] GST is a Java-based system that enables a user to write a service that will invoke applications on a remote machine. GST provides a simple web-services (wsdl) interface compatible with Taverna; uses grid-services to manage data (file) movement between the *remote* computer and the application computer; and launches the application. GST uses the Java CoG interface [8] to Globus GT4.[9]

Using GST to create services provides two benefits. The first is the ability to exploit grid-connected resources without requiring the workflow user to deal explicitly with the grid. Second, as a web service, it fits readily into the Taverna framework. Specifically, the workflow accesses the GST service using web services; the GST service in turn invokes a grid service that transfers data files to a remote grid-hosting computer using gridFTP and launches jobs via GRAM.

As an example we describe some specifics of the Chemistry workflow services (See Case Studies: GAMESS). Three services are required for this workflow: Compile, Link, and Execute. *In practice*, each campaign step is stored to a uniquely named campaign step file. Thus, each service is designed to read a campaign step specification file and to act on it.

1. The Compile Service decides which compiler and options to use in this step. It passes this information to the target system.
2. The Linker Service deals with the choice of which libraries to use at link time. Typical options include choices of math, and tracing libraries.
3. The Execute Service handles runtime choices such as specifying the launcher to use, the number of cores and nodes, and other command line options These may include selection of dynamically linked libraries such as the communication library to employ.

These three services can be combined in different ways depending on target application. For example, the Compile and Link Services are combined in the Chemistry case study.

Most of these services are implemented as a script on the target system that accepts information from its associated service and passes it to the application's mechanisms for building and executing. Build mechanisms have become extremely complex, and this approach reduces the need to modify the changes to the native build or run procedures of the application.

The process for "wrapping" one of these scripts is basically the same for all three services, thus we will illustrate this process by focusing on the Compile Service. We begin by selecting the server onto which the service will be run. Fig. 2 depicts the hardware environment. In this figure, the GST Services server is a large-memory Linux machine that runs all the GST services. It is a grid-hosting environment running GT4 and Java 1.5. First the GST software is installed onto the system. The associated bridging script (compall_workflow) interfaces with the GAMESS [10] native compile procedure (a slightly modified version of `compall`) and is installed on the remote system.

We then create three GST description files. GST names these the HostDescription, ApplicationDescription, and ServiceMap files, respectively. These files together completely specify the service. The ServiceMap file is an application-specific configuration file which specifies the interface between the service and the application (e.g., compall_workflow). In the Compile Service this includes a single input port referring to the campaign step description file. The Compile Service takes this campaign step and delivers it via gridFTP to the remote machine. After delivery, the remote script is invoked to read the "step" and pass the information to the native `compall` script. The ServiceMap also describes an output port that receives stdout and stderr streams returned to the workflow from the compilation process.

The ApplicationDescription file specifies the location of the application (compall_workflow) on the target system. The HostDescription file specifies how to launch jobs to the target system. For the GAMESS workflow, Compile and Link scripts are launched as "fork" jobs to the remote parallel

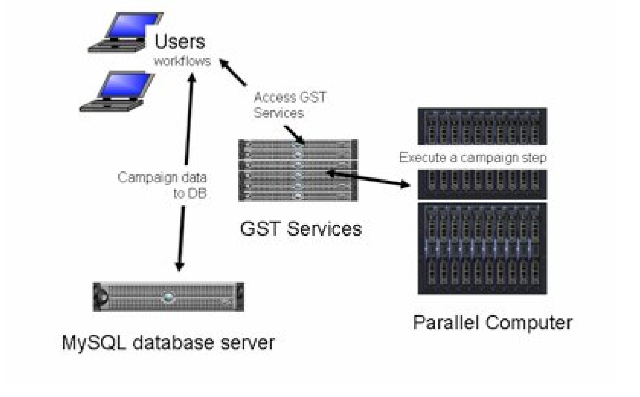


Fig. 2. Hardware environment used by the performance workflows. Large-scale computations were performed on both a large Linux cluster computer and a large IBM Blue Gene/L (BGL)

system's head-node via the GT4 jobmanager-fork launcher. For example, on one of our systems the Execute Service launches the GAMESS application as an MPI [11] job.

The workflow coordinates with the GST service to define and create scratch directories and unique filenames for all input data. To reduce the number of redundant compilation/linking steps when the same executable can be used for multiple experiments, *e.g.*, a scaling experiment using different partition sizes, the workflow reuses compilation and linking configurations when it can.

D. *Storage of results*

The final step of the workflow is the archiving of results from the Compile, Link, and Execute steps into a relational database. Our experiments use MySQL. A simple schema was sufficient for these experiments, see Table 2. We are contemplating extensions and perhaps the adoption of a system such as PerfTrack.[12]

III. Case Studies

To evaluate our strategy for automating performance campaigns, we undertook two case studies. These studies were intended to understand how much of a workflow can be reused between applications, to exercise a representative database schema, and to identify simplifications to the workflow system that facilitate the kinds of optimizations an analyst might ask of an application. The performance workflow system is intended to help answer questions

TABLE 2

DATATYPES STORED FOR PERFORMANCE WORKFLOWS

MySQL name	Description	Example
CampaignName	Name	GamessNetworkingTest
CampaignDate	Launch date of the campaign	Feb 20 2007
CampaignUser	Workflow user	Jtilson
Target	CPU type	X86_64
Machine	Machine name	Kittyhawk
Code	application	Gamess
ExtraDescription	User selection	Compare IB to gigE under hard scaling
Clang	C compiler	icc
Flang	F compiler	ifort
Libs	Additional libs	-lmass
Nodes	Number of nodes	8
CoresPerNode	Number cores per node	4
TotalCores	Cores for run	32
RunCase	Input file	Exam01
Times	walltimes	List of times
DiskUsage	disk space used	List of sizes
ActualCores	ActualTotal cores used by the app.	Not always the same as TotalCores
mpiTimes	mpiTimes	List of times
ioTimes	I/O Times	List of times

typically asked by users, developers, or management such as:

1. How well does the application scale for a fixed problem size? (hard scaling)
2. How well does the application scale for growing problem sizes? (weak scaling)
3. How well does the application scale on alternative interconnection networks and communication libraries?
4. What is the impact of choosing different libraries?
5. How many cores per node can be used at each scale?
6. How much overhead results from turning on profiling or message tracing?
7. How do compiler optimization choices impact scaling and performance?
8. What is the impact of using highly tuned cache optimizations?

Note that the expense of running experiments in the conventional way means that compiler, communications, and library combinations are rarely tested beyond the "obvious" or "historically good" choices.

The first case study addresses a problem in computational chemistry using the highly optimized GAMESS application. The second case study looks at a complicated system of several different applications that is important for large scale storm surge modelling.

It is not the purpose of this paper to discuss either specific performance results or compare them to published results. Rather, we focus on evaluating the impact of using workflow techniques on the process for performing large-scale and HTP performance measurement. Thus the specific results should be considered as suggestive, unless otherwise indicated.

For each case study we give representative results. We discuss observed difficulties in the adaptation of the workflow to the scientific application, and what database schema limitations, if any, were encountered. In the Conclusions we present plans for future deployments.

A. Case Study One: GAMESS

1) Workflow and Campaign description

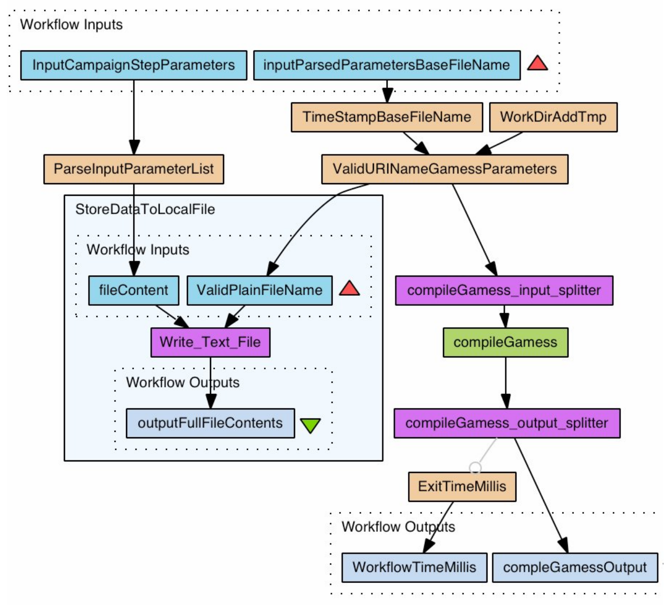
The first case study involves the General Atomic and Molecular Electronic Structure System [10], GAMESS, a general purpose *ab initio* quantum chemistry package with a long history of performance and parallel scalability upgrades.

Fig. 3 is a typical workflow graphic from Taverna which shows major workflow steps. The MySQL sub-workflow is in Fig. 4. In this case study, the Compile and Link Services are combined. For the purposes of illustration some shim- and I/O-related information has been removed.

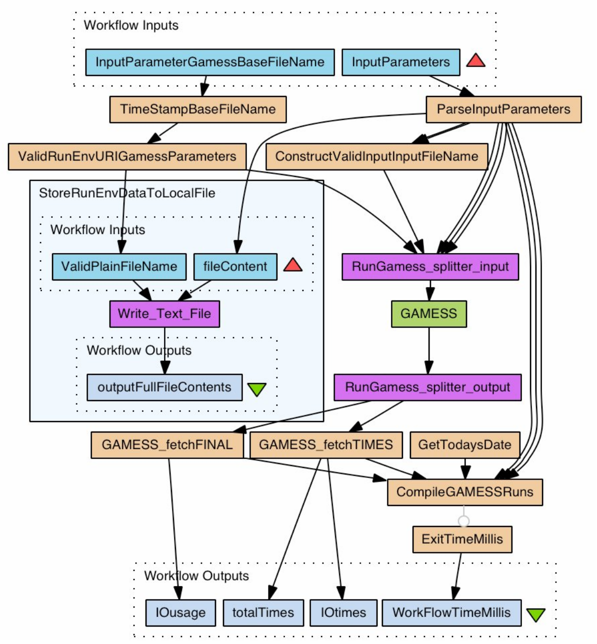
Each box in these Taverna images represents either a processor or an I/O port. The light brown boxes are Taverna local processors (shims, Java beanshell code) that run within the workflow on the driving system. These shims couple the workflow datatypes to the data requirements of a specific application. The "Gamess" and "CompileGamess" processors (colored Green) are the main steps of the workflows.

These representative performance campaigns are intended to answer two questions: How does GAMESS (hard) scale when using an Infiniband (IB) versus a Gigabit Ethernet (gigE) interconnection network? For a fixed number of processors, how do high levels of compiler optimizations compare?

In the following discussion, some GAMESS-specific



A



B

Fig. 3. Depictions of the Compile and Link Services as part of a combined "build" sub-workflow (A) and the Execution sub-workflow(B) for the GAMESS case study. The green colored Gamess and compileGamess boxes refer to the actual GST service calls to the Execute and Compile+Link steps, respectively. The remaining steps manage remote directory names, filenames, GAMESS output parsing and workflow timing.

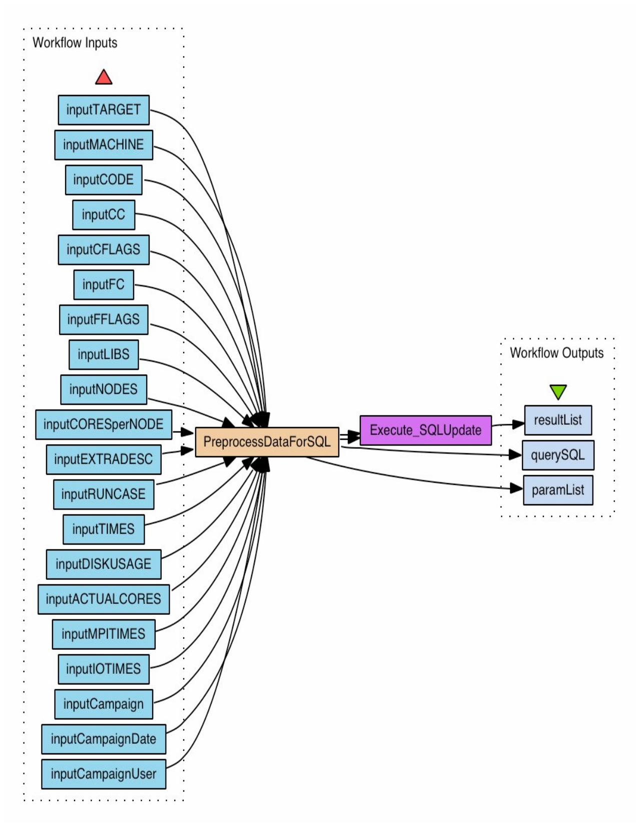


Fig. 4. Depiction of the sub-workflow used to populate the performance database (MySQL) for the GAMESS case study. The vertically aligned input ports are combined in the PreprocessDataForSQL shim and passed to the update service.

nomenclature is used. For the first campaign, we specify a single input (molecule) file for the GAMESS calculation. We choose the "standard.inp" input file supplied with the distribution. This input file performs a DFT(B3LYP) calculation using a 6-311G(d,p) basis set on the molecular system; O_2SiCF_2. Each campaign step was specified to compile the GAMESS using version 9.1 of the Intel compilers for Linux. Interprocessor communications was managed using openMPI.[13] GAMESS was built using the "DDI over MPI" configuration. The compiler options were set to "`-O3 -xT -i8`" and basic linear algebra functionality was supplied by ATLAS [14], version 3.7.19. The parameters that were varied in the campaign include the number of nodes (8,16,32,64), and the choice of interconnection network. The number of cores per node is fixed at 4. Runtime selection of the network fabric and protocol was simplified by our concomitant selection of openMPI. The generality of our services, however, permits other approaches such as re-linking with alternative implementations of MPI. Specifically, as part of the campaign, the "launcher" (See Table 2) parameters values were:

- `mpirun --mca btl mvapi,self (IB)`
- `mpirun --mca btl tcp,self --mca btl_tcp_if_include eth0 (gigE)`

This creates 8 campaign steps. The final energy for each step was recorded into the database and the other outputs are stored in the database for correctness checking.

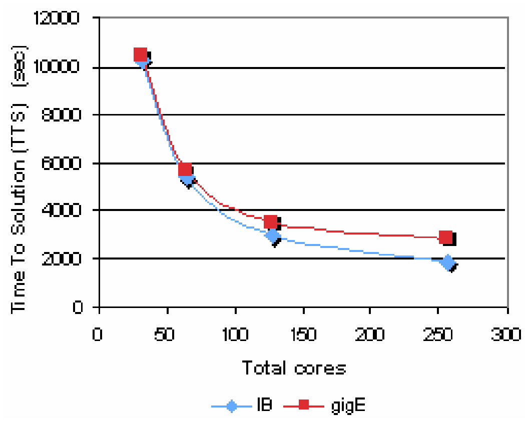


Fig. 5 GAMESS networking campaign. Total Time To Solution (TTS) as a function of the number of applied cluster cores. A comparison between using an Ethernet (gigE) versus an Infiniband (IB) interconnect.

2) *GAMESS Results*

The scaling results are shown in Fig. 5. Not surprisingly, GAMESS scales less well using gigE than when using Infiniband (IB). Assuming ideal behaviour at 32 cores, the IB test indicates a parallel efficiency of 69.7% on 256 cores. The analogous gigE value becomes 45.8%. The fact that IB is a faster network and that GAMESS scales better on IB is not the point of these results. Rather, the HTP approach allows the researcher to look at this data and then ask follow-on questions such as "How does scaling change with multiple kinds of compiler options?","How does scaling vary with/without using ATLAS?","How does scaling vary with different compilers?" Manually recording performance information is tedious, so automatically and consistently archiving results in a database for subsequent data mining is a large boost for productivity.

B. *Case Study Two: The North Carolina Floodplain Mapping Project (NCFMP).*

The NCFMP partnership[1] is developing a state-of-the-art system for the simulation of storm surge levels along the North Carolina coastal region waters for input into floodplain analysis systems as required by the Federal Emergency Management Agency (FEMA). The computationally expensive piece of this analysis is being executed on an IBM Blue Gene/L (BGL) machine, but can also effectively use large-scale clusters that have high performance interconnects. In this work we report BGL results.

Each simulation of the storm surge for a particular storm scenario is performed as a pipelined composition (dataflow) of wind, wave, and surge models. Storm scenarios involve synthetically-generated hurricanes, category types 3 through 5

[1] This is a multi-organizational effort involving the Renaissance Computing Institute (RENCI), the UNC-Chapel Hill Institute of Marine Sciences, the US Army Corps of Engineers,, Applied Research Associates, Coastal Risk Technologies, Dewberry, and the North Carolina Division of Emergency Management.

for both land-falling and bypassing storms, as well as extra-tropical storms. The four models that comprise the NCFMP system include a Hurricane Boundary Layer (HBL) model [15] for wind and pressure, a basin-scale wave model for offshore waves, WaveWatch3 (WW3) [16],[17], a near shore wave model, SWAN [18],[19], and a state-of-the-art coastal circulation model ADCIRC [20] to compute the coastal inundation. The model components are coupled together through file exchange and the entire simulation process is scripted.

1) *Computational Requirements*

For each storm track (scenario), there is an HBL wind run, followed by a WW3 run and then four SWAN runs; two runs on coarse grids run concurrently, followed by two runs on fine grids which may also run concurrently. These are followed by two ADCIRC runs, one without forcing from the SWAN output and one with the forcing to help quantify the effect of waves on storm surge. By far, the most computationally expensive steps are the SWAN and ADCIRC runs. Our testing framework focuses on each of these components individually.

The computational workload for the entire project is substantial. The first phase of the study required over 1.4 million processor hours on the BGL system. The next phase of analysis will be about one order of magnitude larger. Efficient execution is vital.

2) *Re-Purposing the Base Workflow*

We adapted the base performance workflow to run the storm tracks for the NCFMP system on the BGL. This allows us to reuse much of the workflow infrastructure, such as the generically defined inputs and outputs, the construction of the campaign steps and the sub-workflows that produce the campaign step files and that store results into the relational database. The NCFMP workflow is shown in Fig. 6. This is similar to the workflow shown in Fig. 3(B). The primary difference is the green box at the heart of the workflow, here labelled NCFMP. This is where the NCFMP web service is invoked, rather than the GAMESS web service. Thus, we are required to write a new web service. This is not difficult and is done by creating GST description files, as described in the Service Technologies section. This wraps an application and uses Globus to launch jobs to the BGL. As in the GAMESS studies, GST wrapping requires specification of input and outputs to the application and the writing of a short script to run on the target system. This script is invoked by the web service. It performs the campaign step compilation; submits the application to the batch queue; and passes parsed results back to the workflow through the web service. The script interfaces with the production infrastructure developed for NCFMP. The various compile and linking options are passed cleanly into the respective source trees for SWAN and ADCIRC without modifying the original code. Thus, the workflow is minimally intrusive and the abstractions for the generic design proved successful.

3) *Performance Results*

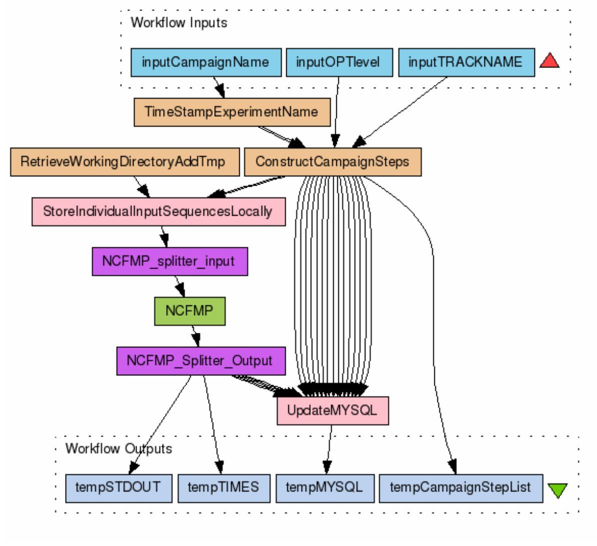


Fig. 6 Depiction of the primary NCFML sub-workflow. The storage sub-workflow is that same as that in Fig. 4 and is not pictured here. The green colored NCFMP box refers to the GST service call.

Several performance campaigns were enacted for both SWAN and ADCIRC to quantify their performance and scaling attributes, as well as to illustrate the use of HTP techniques to answer some of the questions posed at the beginning of Case Studies. These campaigns involved running a short test storm track that modelled 5.5 hours around landfall of the 1996 hurricane Fran that struck the North Carolina coast. Each individual campaign step was a particular combination of compiler options, link options, processor counts, and distribution across nodes for either SWAN or ADCIRC.

3.1) *SWAN Results*

MASS libraries: The use of the Mathematical Acceleration Subsystem (MASS) library routines are thought to improve performance [21] on the BGL system for frequently used math intrinsic functions such as `sqrt` or `exp`. We performed a campaign to determine whether using MASS would improve overall code performance and if so, by how much. In combination with other choices of compiler options, these experiments produced many runs that differ only in selection of the MASS library. Side-by-side comparisons using this set show that, indeed, using MASS did always improve performance. For SWAN on 100 and 150 processors the improvement varied from 4-18% with one exception. For SWAN on 16 processors, where the time spent on MPI and I/O is proportionally less important (and are unaffected by the use of MASS libraries) we find that the results split into two categories. Either there was a no statistically significant benefit, 1-3% for codes with optimization levels of `O5` (and one with `O3 qhot`) or there was a significant benefit (11-16%) for all other combinations. This contradicts the Blue Gene Compiler Guide [21] which indicates that the MASS libraries will be automatically enabled for optimization levels `O5`, `O4`, and `O3 qhot`. Our findings agree with this for the `O5` and `O3 qhot` settings but we consistently observed a significant performance boost using `O4 with the MASS libraries over O4 without the explicit specification. This was observed` on both inner grids and for all of the processor counts. This may be related to the use of the `sqrt` function in the SWAN code. Without automation, it is doubtful that we could have done the full set of runs.

Cache Optimizations: The IBM blrts compiler family allows the selection of the `qcache` option with very specific cache optimizations that can be tuned to the PowerPC architecture on the BGL. As with the MASS campaign, we compared results for many compiler and processor configurations with and without this option specified. Overall, no systematic benefit of using this optimization was found with generally little or no effect observed.

Compiler Optimizations: For the time consuming inner grid runs of SWAN, we examined combinations of the base optimizations `O3`, `qnoautoconfig O4`, `qnoautoconfig O5` with the additional optimizations of `qnounwind`,

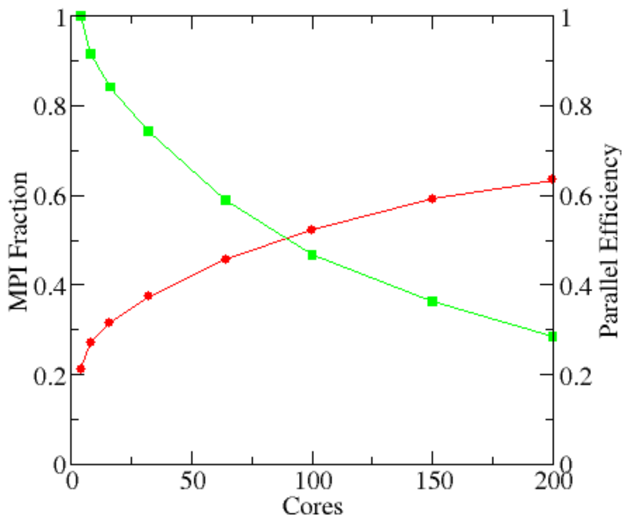


Fig. 7. SWAN (hard) scaling on the RENCI IBM BG/L system. Red circles indicate the fraction of total run time spent in MPI library calls. Green squares are the parallel efficiency relative to the 4 processor run.

`qunroll=yes`, `qarch=400`, `qarch=400d qtune=440`, `qbgl qhot` (used `qipa` for `O3`) on 16 and 128 processors. The `qarch=440` flag was the only one that consistently showed significant performance improvements albeit with one exception. The `qarch` flag only with the `O4` optimization only on one of the inner grids inexplicably did much worse then any other setting. Over all cases the `“-O5 –qnoautoconfig –qarch=440”` was the fastest setting for both grids and processor counts.

Scaling: Fig. 7 shows the results of a hard scaling study for the SWAN code using the inner (finer) grid. As can be seen, the application does not scale to large numbers of processors and this is reflected in both the efficiency plot (scaled to the 4 core results) and the fraction of run time spent in MPI calls, as reported by mpiP [22]. These are the results for the short, hurricane Fran test track but they are representative of the longer runs as the tracks will get longer in time but will run on the same grid. The plots show that MPI time grows steadily out to 200 cores and that the efficiency of the code is less than 50% at 100 cores. The conclusion from this study is that it is more productive to run several storm tracks concurrently on smaller system partitions, than to use large partitions to speed up individual runs.

3.2) ADCIRC Results

MASS Libraries: For ADCIRC, running on 128 processors and using various optimization flags combined with `O3`, `O4`, and `O5`, we saw an additional 3-7% improvement when using the MASS libraries versus the identical run without MASS. As noted in the SWAN results section, this is somewhat at odds with the Blue Gene compiler documentation.

Compiler Optimizations: We performed several campaigns to evaluate compiler flags. These generated nearly 90 ADCIRC runs on 128 processors. The variation in time from fastest to slowest for the 1996 Fran test track was 40%. We concluded that a minimum level of `O3` was required to give good performance. The `O4` and `O5` flags yielded further improvements. For the blrts compiler, these levels augment `O3` with some specific additional optimizations. To clarify the effects, we examined optimization levels of `O3` with the MASS libraries and then with selected additional optimizations. We summarize our results as follows. The `qarch=440` was better than `qarch=440d` across various optimization levels. The code was not sensitive to some

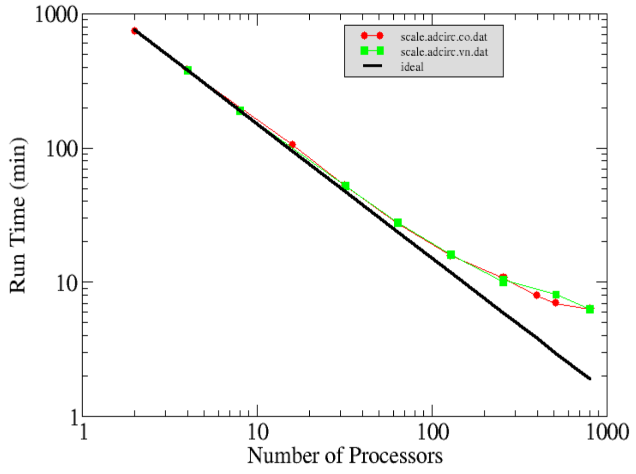


Fig. 8 ADCIRC hard scaling results from the RENCI IBM BGL system. Compared are using virtual node (vn) versus coprocessor (co) modes. Ideal scaling shown as a solid black line.

particular combination of compiler options; rather we observed a clustering of campaign steps that showed similar performance (within 3% of the best time). The `qhot` flag optimization seemed to improve performance with `qhot=vector` or `qhot=simd` being the best, followed closely by `qhot=level=1`. Curiously, further increasing

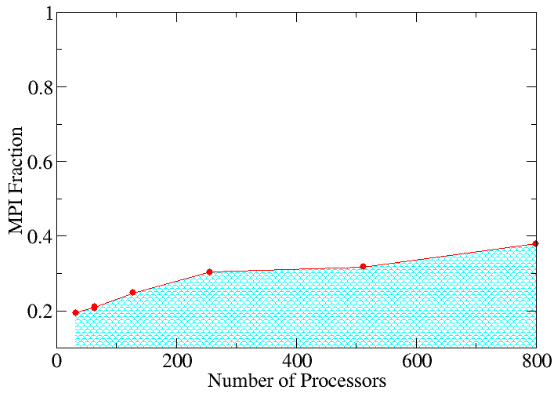


Fig. 9 ADCIRC: Fraction of MPI time on the BGL

this, using `qhot=level=2` degraded the performance for this code. It was hard to see any consistent benefit when using `qbgl`, `qessl`, `qnounwind` and using `qunroll=yes` only resulted in slower runs. Using the inter-procedural analysis, `qipa`, with various levels also failed to provide any boost in performance. To summarize, using the MASS libraries with "`-O3 -qnoautoconfig -qarch=440 -qtune=440 -qhot=vector`" seems to be a good combination. Without automation, this search would have been too labor-intensive to have performed.

Scaling: Fig. 8 shows ADCIRC (hard) scaling on the BGL system. The plot shows results for running the code in both available modes, namely coprocessor mode (co) and virtual node (vn) mode. For this application, no benefit from running in coprocessor mode was observed. ADCIRC exhibits good scaling up to around 100 processors and beneficial scaling to 500 or so processors with level `O3` optimization. Fig. 9 shows the percentage of MPI time spent running in vn mode as reported from profiling with the MPI trace library [23] available on the BGL system. In contrast to SWAN, the ratio of MPI time to computation grows modestly in runs up to 800 processors. This is reflected in the nice scaling exhibited in Fig. 8.

IV. Evaluation

Adaptation of the base workflow and services to GAMESS was not difficult. The workflow itself required additional shims to interface GAMESS output files to the workflow and to fetch relevant information (times, energies) for the database. Two GST services that were written for performing the Compile + Link, and Execute steps were constructed from *base* services. Thus, once access to the remote machines is granted, deployment of the service is fairly straightforward. It was necessary to have login access to the remote machines to debug services, and understand developmental issues that arise while adapting the base workflows.

A few problems occurred when adapting the system to GAMESS. The first was an insufficient schema for the MySQL database. This observation will likely be true for several new applications during our prototyping period of work. The second problem pertains to the inability to disconnect/reconnect to the workflow in Taverna. A newer version of Taverna (1.7) supplies techniques for doing this. Other issues include the occasional failure to update the database. For the GAMESS "standard.inp" file, the run times are sufficiently long that for even a modest sized campaign (64 or fewer steps, several days runtime), the chances of a system failure somewhere in the grid environment is high. A failure results in complete loss of all data for the campaign. So for now, it is important for campaigns to be quite focused by varying only a few options at a time. In the future, we will incorporate methods for robustly restarting workflows and campaigns.[24]

A major benefit of automating performance experiments as described in this work is that it makes regression testing more economical. For example, a new and improved grid (a two dimensional irregular finite element mesh) will be available soon for ADCIRC. This grid is nominally 25% larger than the current grid and will resolve more features along the North Carolina coast. Repeating our extensive suite of past performance experiments for the new configuration will be doable with minimal commitment of labor. Using the framework described in this paper that is simply a matter of reloading a campaign and automatically rerunning it. The results will be automatically archived into the database for easy extraction and comparison with previous experiments. Similarly the SWAN grids, a coarse and a fine regular rectangular grid for each of the northern and southern North Carolina shorelines, are being evaluated and may change as well.

V. Conclusions

Performance experiments are labor intensive and the use of clusters and grids with varying system and node architectures has only increased to combinatorial complexity of running extensive suites of experiments. Faced with this challenge, we automated our performance evaluation work through the use of a grid workflow framework.

Once an application has been "wrapped" to connect its build and execution procedures to the framework, it is easy to design performance campaigns to do extensive and exhaustive studies. Because campaigns can be rerun, it is much easier to study statistical variation; to perform regression studies when hardware or software, either application or system, has changed; and it is relatively easy to perform these studies across a range of diverse systems.

Working with new applications does require the manual creation of wrapper scripts. While this requires a modest amount of work by an expert, in the end it saves enormous amounts of tedious effort running experiments.

Thus far, our campaigns have been structured as exhaustive searches in small sub-spaces of the available parameters. In our applications we needed a full combinatorial search, so this was not an issue for us. In the future we are planning to move towards more efficient optimization methods as are being explored under the term “autotuning”.[25]-[27] These searches will require loops and other data-dependent control flow in the specification and execution of the campaign. We are investigating frameworks that can support these capabilities.

Acknowledgements

We gratefully acknowledge support from the NSF: Collaborative Research: The NSF Cyberinfrastructure Evaluation Center SCI-0510267.

We also wish to thank the Renaissance Computing Institute for providing access to large-scale computing platforms and Bradley Viviano for essential cluster, IBM BGL, and Globus GT4 support

We wish to thank Brian Blanton and Howard Lander for fruitful discussions on the NCFMP project.

References

[1] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M.R. Pocock, A. Wipat, and P. Li, “Taverna: a tool for the composition and enactment of bioinformatics workflows”, *Bioinformatics*, 20, 3045–3054, 2004.

[2] J.L. Tilson, G. Rendon, M.-F. Ger, E. Jakobsson, “MotifNetwork: A Grid-enabled Workflow for High-throughput Domain Analysis of Biological Sequences: Implications for annotation and study of phylogeny, protein interactions, and intraspecies variation” in *7th IEEE International Conference on Bioinformatics and Bioengineering* (BIBE’07), 2007, pp. 620-627.

[3] J.L. Tilson, A. Blatecky, G. Rendon, M.-F. Ger, E. Jakobsson, “MotifNetwork: Genome-Wide Domain Analysis using Grid-enabled Workflows” in *7th IEEE International Conference on Bioinformatics and Bioengineering* (BIBE’07), 2007, pp.872-879.

[4] I. Foster, C. Kesselman, S. Tuecke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations” *International J. Supercomputer Applications*, 15(3), 2001.

[5] I. Foster, C. Kesselman, Chapter 2 of *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufman, 1999.

[6] J. Yu, R. Buyya, “A Taxonomy of Workflow Management Systems for Grid Computing,” Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005

[7] G. Kandaswamy, L. Fang, Y. Huang, S. Shirasuna, S. Marru, and D. Gannon “Building Web Services for Scientific Grid Applications,” *IBM Journal of Research and Development*, 50(2/3), pp. 249-260, 2006.

[8] G.von Laszewski, I. Foster, and J. Gawor, “CoG kits: a bridge between commodity distributed computing and high-performance grids” in *Proceedings of the ACM 2000 Conference on Java Grande* (San Francisco, California, United States, June 03 - 04). JAVA '00, 2000. ACM Press, New York, NY, pp. 97-106.

[9] I. Foster, “Globus Toolkit Version 4: Software for Service-Oriented Systems,” in *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS 3779, 2006, pp 2-13. http://www.globus.org

[10] M.S. Gordon, M.W. Schmidt, "Advances in electronic structure theory: GAMESS a decade later," in *Theory and Applications of Computational Chemistry: the first forty years* C.E. Dykstra, G. Frenking, K.S. Kim, G.E. Scuseria, Eds., Elsevier, Amsterdam, pp. 1167-1189., 2005.

[11] W. Gropp, E. Lusk, A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 2nd ed., MIT Press, 1994

[12] R.L. Knapp, K. Mohror, A. Amauba, K.L. Karavanic, A. Neben, T. Conerly, and J. May, "PerfTrack: Scalable Application Performance Diagnosis for Linux Clusters," in *8th LCI International Conference on High-Performance Clustered Computing*, 2007, May 15-17, South Lake Tahoe, California, USA.

[13] R.L. Graham, G.M. Shipman, B.W. Barrett, R.H. Castain, G. Bosilca, A. Lumsdaine, “Open MPI: A High-Performance, Heterogeneous MPI,” in *Proceedings, Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, 2006, Barcelona, Spain.

[14] R.C.Whaley and A.Petitet, “Minimizing development and maintenance costs in supporting persistently optimized (BLAS),” *Software: Practice and Experience*, 35, 101-121, 2005.

[15] P.J. Vickery, P.F. Skerjl, and L.A. Twisdale, “Simulation of hurricane risk in the U.S. using empirical track model,” *J.Struct. Engr*., pp. 1222-1237, 2000.

[16] H.L. Tolman, *User manual and system documentation of WAVEWATCH-III*, Version 2.22. Technical Note, U.S. Department of Commerce, NOAA, NWS, NCEP, Washington, DC., 2002.

[17] H.L. Tolman, D.V. Chalikov, “Source terms in a third-generation wind-wave model,” *J. Phys. Oceanogr*., 26, pp. 2497-2518, 1996.

[18] N. Booij, R.C. Ris and L.H. Holthuijsen, “A third-generation wave model for coastal regions, Part I, Model description and validation,” *J. of Geophysical Research*, C4, 104, pp. 7649-7666, 1999.

[19] W.E. Rogers, J.M. Kaihatu, H.A.H. Petit, N. Booij, and L.H. Holthuijsen, “Diffusion reduction in a arbitrary scale third generation wind wave model,” *Ocean Engng*., 29, pp. 1357-1390, 2002.

[20] R.A. Luettich, Jr., J.J. Westerink, and N.W. Scheffner, “ADCIRC: an advanced three-dimensional circulation model for shelves coasts and estuaries,” U.S. Army Engineers Waterways Experiment Station, Vicksburg, MS, Report 1: Theory and Methodology of ADCIRC-2DDI and ADCIRC-3DL, Dredging Research Program Technical Report DRP-92-6, p. 137, 1992.

[21] *Using the XL Compilers for Blue Gene*, IBM, 1st ed.,SC10-4310-00, 2006.

[22] J.S.Vetter and M.O.McCracken, "Statistical Scalability Analysis of Communication Operations in Distributed Applications," *Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming* (PPOPP), 2001.

[23] G. Mullen-Schultz, *Blue Gene/L: Performance Analysis Tools*, IBM Red Books, SG24-7278-00, July 2006.

[24] G. Kandaswamy, A. Mandel, and D.A. Reed, “Fault Tolerance and Recovery of Scientif Workflows on Computational Grids,”in Proc. IEEE International Symposium on Cluster Computing and the Grid” workshop on Resiliency in High-Performance Computing in conjunction with CCGrid’08, 2008. to appear http://www.renci.org/~anirban/papers/PaperResilience08-Anirban.pdf

[25] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms," Supercomputing (SC), 2007.

[26] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. Clint Whaley, K. Yelick, “Self-Adapting Linear Algebra Algorithms and Software,”in Proc. IEEE, Special Issue on Program Generation, Optimization, and Adaptation, February 2005, 93(2).

[27] CScADS Workshop on Automatic Tuning for Petascale Systems, K. Yellick and K. Cooper (organizers), http://cscads.rice.edu-/workshops/july2007/autotune-workshop-07