Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 00 (2011) 1–12
Procedia Computer Science

Toward Malleable Model Coupling

Dai-Hee Kim[a], J. Walter Larson[b,c,d], Kenneth Chiu[a]

[a]*Department of Computer Science, SUNY Binghamton, P.O. Box 60000, Binghamton, NY 13902, USA*
Email: {dkim17,kchiu}@cs.binghamton.edu
[b]*Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439, USA*
Email: larson@mcs.anl.gov
[c]*Computation Institute, University of Chicago, 5735 S. Ellis Ave., Chicago, IL 60637, USA*
[d]*School of Computer Science, The Australian National University, Canberra, ACT 0200, AUSTRALIA*

Abstract

Model coupling is a well-known method employed to simulate complex multiphysics and multiscale phenomena. Approaches have concentrated on coupling parallel models involving static data distribution among processes without the consideration of top-level dynamic load balancing. Malleability, the ability to change during execution the number of processes in an application, allows applications to configure themselves to better utilize available system resources. To date, however, malleability has been applied primarily to monolithic applications. We have extended the Model Coupling Toolkit (MCT) to support processing element malleability for coupled models, resulting in the Malleable Model Coupling Toolkit (MMCT). MMCT consists of a load balance manager (LBM) implementing a practical dynamic load-balancing algorithm and a malleable model registry that allows management of dynamically evolving MPI communicators. MMCT requires only standard MPI-2, sockets, and MCT. We benchmark MMCT using a synthetic, simplified coupled model application similar to the Community Climate System Model. Preliminary performance data demonstrate the efficacy of the LBM and a low ($\approx 3\%$) monitoring overhead.

Keywords: MPI, Dynamic Load Balance, Model Coupling, Multiphysics Modeling, Multiscale Modeling

1. Introduction

Current scientific challenges often involve complex systems spanning multiple disciplines (e.g., a climate system model) or spatiotemporal scales (e.g., nested-domain numerical weather prediction)—*multiphysics* and *multiscale* problems, respectively. A defining characteristic of these systems is the existence of data dependencies, or *couplings*, between their constituent subsystem models, or *constituents*. Thus, multiphysics and multiscale models are more generally called *coupled models* or *coupled systems*. Coupled models abound among grand challenge applications in computational science. Advancements in microprocessor technology and the advent of parallel computing have enabled coupled model development. The march toward exascale computing will make coupled models increasingly common and pose new challenges for model coupling software. A critical requirement for utilizing up to million-way parallelism will be the design and implementation of highly effective and efficient load balance mechanisms, including support for dynamic load balance both within and between constituents. An online automatic rebalancing scheme will be critical as multiphysics and multiscale models grow in complexity from relatively few to large numbers of constituents.

The challenge inherent in building coupled models is called the *coupling problem* [1]. Moreover, coupled systems frequently employ parallel computing, in particular distributed-memory parallelism using the Message Passing Interface (MPI) [2] standard, giving rise to the *parallel coupling problem* (PCP) [1]. Each constituent in a parallel coupled model resides on a set of processors—or *processing elements* (PEs)—called a *PE cohort*, or simply *cohort*. Parallel coupled models evolve their state by solving each constituent's equations of evolution on its respective cohort, with data dependencies on other constituents satisfied at a minimum by parallel $M \times N$ data transfers [3], though other data transformation operations (e.g., intermesh interpolation) are likely required.

The main performance concern of scientists who use coupled models is *throughput*, which is the amount of simulation accomplished (in terms of the model's timestep) divided by the wall clock time required to perform the simulation; for example, throughput in a coupled climate model is measured in model years per wall-clock day. Achieving high throughput requires not only that individual constituents scale well but also that the resources allocated to each constituent be harmonized to minimize lost cycles while constituents await data during the coupling process. A key challenge in building efficient parallel coupled models is the allocation of resources, specifically PEs, to constituents, and the reallocation of resources should runtime load characteristics change dramatically during a simulation. Load balance and resource allocation in parallel coupled models strongly affect model throughput. The present work's focus is solely on this aspect of the PCP. Typically, parallel coupled models employ static load balance, with resource allocation to constituent subsystems determined offline through trial and error. Automatic dynamic load balance, both within and between constituents, is thus highly desirable. Load balance within a constituent is amenable to existing load-balancing approaches. Interconstituent load balance, however, requires further work, in particular the capability for a coupled model to resize constituents' PE cohorts to better use system resources; we call this property *cohort malleability*, or simply *malleability* [4]. We define a parallel coupled model with malleable constituent cohorts to be a *malleable coupled model* (MCM).

Malleability imposes new requirements on coupled model infrastructure: (1) load monitoring; (2) evaluation of current load balance; (3) algorithms for computing more optimal resource distributions; (4) checkpointing or redistribution of constituents' states; (5) reallocation of constituent PE pools; and (6) instantiation and initialization of constituents on the resized PE pools, including handshaking of interconstituent $M \times N$ communication schedules. One can also view these requirements as the sequence of operations performed by an MCM as it rebalances itself. Our primary focus is a generic, portable solution to requirements (1–3) and (5–6). We view checkpointing or redistribution as highly application-specific. A general mechanism for checkpointing parallel codes is a problem beyond the scope of our work.

Malleable iterative parallel applications have been investigated and shown to be beneficial. The ReSHAPE [5] framework adopts malleability for efficient job scheduling by resizing the processor allocation of malleable applications based on the utilization of distributed resources. Maghraoui et al. [6, 7] introduced malleable iterative MPI applications that can change their process configuration dynamically by working with the PCM/IOS runtime environment. PCM/IOS provides mechanisms for reconfiguration, most notably a profiling module that monitors the status of applications to guide and inform resource reallocation. SRS [8] also provides methods to allow a parallel application to reconfigure itself through its stop and restart processes. These approaches, however, are not immediately applicable to malleable model coupling because they are targeted to monolithic parallel applications, as opposed to frameworks/toolkits for model coupling. Ko et al. [9] describe a coupled multiphysics simulation that can optimize the allocation of constituents on the given set of PEs. The load-balancing algorithm described in [9] is acceptable only for a coupled model consisting of two constituents. Moreover, each model communicates by exchanging files, an approach that is less general than MPI or socket-based communication. The CSCAPES project [10] has investigated load balancing and allocation issues but primarily has used a priori knowledge about the application and has examined mainly load balance within a constituent. As coupled models become increasingly complex, mechanisms that gather online performance information and utilize it to perform interconstituent dynamic load balance become increasingly attractive.

To address these issues, we extended the Model Coupling Toolkit (MCT) [11, 12] to enable it to support malleable MPI communicators and a malleable MPI_COMM_World, thus supporting MCMs. MCT is generic MPI-based coupling middleware that is widely used by the climate, weather, and ocean modeling communities. We enhanced MCT by adding load-monitoring and optimization schemes, a central load balance manager, and a mechanism for resizing constituents' cohorts. The resulting system—*malleable MCT* (MMCT)—extends MCT's programming model to allow construction of parallel coupled models whose constituent PE cohort configuration is automatically reconfigurable at runtime using decisions made by its load balance manager. We then investigated various dynamic load-balancing techniques to show the feasibility of the approach and the effectiveness of the modifications to MCT. We devised general optimization and expansion algorithms loosely based on the gradient descent method [13]. The optimization and expansion algorithms are used to optimize the configuration for N constituents and to allocate more available resources on a homogeneous cluster to improve utilization of distributed resources and coupled model throughput.

We briefly discuss MCT, describe our load-balancing mechanism, and give an overview of MMCT in Sections 2, 3, and 4, respectively. In Section 5 we describe a testbed coupled climate model simulator that employs MMCT to implement malleable model coupling, and we present preliminary performance results. We conclude with a discussion of the implications of our results and an outline of future work necessary to deploy MMCT in production.

2. Model Coupling Toolkit

MCT is open-source software [14] that provides an API consisting of datatypes and functions supporting the description of constituents and their input/output data, parallel data transfer and redistribution, and commonly encountered

data transformation operations. Supported data transformation operations include conservative intergrid interpolation implemented as a parallel linear transformation, time integration (averaging) of flux (state) variables, and merging of outputs from multiple constituents for input to another constituent. Collectively, MCT's API is widely applicable to data coupling between legacy MPI-based parallel models. MCT implements a *peer communication* approach: couplings are viewed as large-scale $M \times N$ messaging between models, extended by other data transformation operations. A complete discussion of MCT's design philosophy and object model is given in [11]. A detailed description of MCT's parallel data transfer and regridding facilities is given in [12]. MCT has been used in numerous applications, most notably as coupling middleware for versions 3 and 4 of the Community Climate System Model (CCSM) [15, 16], used by an international community to study climate change, climate variability, climate sensitivity, and paleoclimates.

Below we describe aspects of MCT pertinent to its extension to support malleability. The MCT programming model is largely unchanged by the implementation of cohort malleability, and modifications of an existing parallel coupled model to make it an MCM should be straightforward.

The MCTWorld class is a lightweight constituent registry, encapsulating the number of constituents in the coupled model, the allocation of processors to constituents, and their individual communicators. MCT was developed by using the MPI-1 standard's [2] communicator creation and manipulation scheme. The MCTWorld class encapsulates a global communicator that is typically a copy of MPI_COMM_WORLD. The singleton MCTWorld can be created by using either a single master call by the root PE on the global communicator or a distributed call by each PE on each constituent's cohort, with individual constituent communicators previously created by application of a communicator splitting mechanism such as MPI_COMM_SPLIT() to the global communicator. MCT stores the communicator ID and list of member MPI processes for each cohort in the MCT_World. MCT builds communications schedulers for $M \times N$ transfers through process ID lookup using the MCT_World registry. This is conceptually attractive because it allows constituents to exchange domain decomposition descriptions defined using local process IDs on their respective communicators, with the MCT_World the interlocutor to create interconstituent communications schedules. In the current, publicly released version of MCT, the MCT_World class is considered a singleton, instantiated in the global variable ThisMCTWorld, and static over the life of an application at runtime.

3. Load-Balancing Algorithm

We first define terms for timestepping and resource allocation in coupled models. The time evolution of a coupled model occurs as each constituent solves its respective equations of evolution on its domain, utilizing (supplying) coupling input (output) from (to) other constituents. The provision (uptake) of coupling data occurs during *coupling events* [1]. The timing of coupling events may not be known in advance, but rather as a consequence of runtime behavior. Or, the coupling events may be known a priori—*scheduled coupling* [1]. For some scheduled coupling situations, the full schedule of interconstituent coupling time intervals may be mutually commensurate, resulting in a *coupling cycle* [1]. The ratio of timescales corresponding to the constituent's timestep versus the interval between coupling events can be used to define the *tightness* of the couplings [1]. For example, in the standard configuration of CCSM, the constituents are the atmosphere, ocean, sea-ice, and land models and an additional entity called the *coupler*; all data traffic between the constituents in CCSM is directed via the coupler. In a standard configuration of CCSM, the model timesteps are $\{\Delta t_{\rm atm}, \Delta t_{\rm ocn}, \Delta t_{\rm ice}, \Delta t_{\rm land}, \Delta t_{\rm cpl}\} = \{20, 20, 60, 60, 60\}$ minutes. CCSM's couplings are scheduled, with the interval between the atmosphere, land, and sea-ice models' interactions with the coupler occuring hourly, while the ocean communicates with the coupler once per model-day; thus CCSM has a repeating coupling cycle ΔT equal to one model-day. Note that the atmosphere, land, and sea-ice models are more tightly coupled to the system than is the ocean model.

PE cohorts in coupled models can be defined in a wide variety of configurations, or *process compositions* [1]. The two most basic are *sequential* and *parallel* compositions [17]. A sequential composition allocates all constituents to the same PE cohort, with the coupled system running as an event loop executing each constituent in turn. A parallel composition allocates each constituent its own PE cohort, and these cohorts are disjoint.

Consider a coupled model with N constituents. In the load balance analysis that follows, we assume that each constituent has its own constant timestep $\Delta t_i, i \in \{1, \ldots, N\}$; coupling events fall into a coupling cycle $\Delta T \geq \Delta t_i \forall i$; and the constituents are configured in a parallel composition. These assumptions, though highly restrictive, are present in a wide variety of coupled climate, weather, and ocean applications.

Our load-balancing schemes analyze constituent profiling data gathered over a *load-sampling interval* (LSI); in the analyses presented here, the LSI corresponds to the coupling cycle time ΔT. The *global iteration time* τ_G is the wall clock time required by the coupled model to evolve its state over a period ΔT. The *constituent iteration time* τ_i is the wall clock time required for the constituent to evolve its state over the period ΔT; this time is decomposed as $\tau_i = \tau_i^{\rm comp} + \tau_i^{\rm coup}$, the sum of the *constituent computation* and *constituent coupling* times, respectively. In other words, $\tau_i^{\rm comp}$ and $\tau_i^{\rm coup}$ are the wall clock times during which the constituent computes its internal state (including comcomitant communciations within

```
initialize
prev_iter_time = ∞; prev_comp_sum = ∞;
prev_donor = -1; prev_recipient = -1;
end
update
prev_iter_time = cur_iter_time;
prev_comp_sum = cur_comp_sum;
prev_donor = donor; prev_recipient = recipient;
end
initialize;
repeat
  cur_iter_time = τ_G;
  cur_comp_sum = sum of τ_i^comp of all models;
  {Condition 1.}
  if cur_iter_time ≤ prev_iter_time and cur_comp_sum ≤ prev_comp_sum
  then
    Mark [prev_recipient, prev_donor] pair as a non-viable reallocation
    direction;
    Choose [donor, recipient] pair using SEL1 or SEL2;
    num_procs = 1; try_count = 1;
    update;
    {Condition 2.}
  else if (cur_iter_time ≤ prev_iter_time or cur_comp_sum ≤
  prev_comp_sum) and try_count < try_count_limit then
    {Try again.}
donor, recipient = [prev_donor, prev_recipient];
    num_procs = 1; try_count++;
    update;
  else {Condition 3}
    Mark [prev_donor, prev_recipient] pair as a non-viable reallocation
    direction;
    {Reverse direction to undo}
donor, recipient = [prev_recipient, prev_donor];
    num_procs = try_count;
    initialize;
  end if
  if [donor, recipient] pair is viable then
    reallocate([donor, recipient], num_procs);
  end if
until [donor, recipient] pair is viable
```

Figure 1: Optimization algorithm.

```
{ n_models is the number of models. models_comp is an array of model
references in decreasing τ_i^comp order. models_slope is an array of model
references sorted in decreasing derivative δτ_i^comp order.}
{First search.}
for i = n_models − 1 to 1 do
  {Current smallest slope.}
  donor = models_slope[i];
  {Pick greatest comp time as recipient.}
  for j = 0 to j = n_models − 1 do
    if models_comp[j].comp_time > donor.comp_time and mod-
    els_comp[j] not donor then
      recipient = models_comp[j];
    end if
  end for
  if [donor, recipient] pair is viable reallocation direction then
    return [donor, recipient];
  end if
end for
{Second search, if slowest model can no longer benefit from more pro-
cessors.}
for i = 0 to n_models − 1 do
  donor = models_comp[i];
  for j = 0 to n_models − 1 do
    if models_slope[j].comp_time < donor.comp_time and mod-
    els_slope[j] not donor then
      recipient = models_slope[j];
    end if
  end for
  if [donor, recipient] pair is viable reallocation direction then
    return [donor, recipient];
  end if
end for
{No viable direction}
return [-1, -1];
```

Figure 2: SEL2 selection algorithm.

its cohort) and awaits (supplies) data from (to) other constituents, respectively. Our objective is to formulate analyses of $\{\tau_1, \ldots, \tau_N; \tau_1^{\text{comp}}, \ldots, \tau_N^{\text{comp}}\}$ that guide constituent cohort reallocation to decrease τ_G.

We assume that τ_i^{comp} for a constituent follows a simple curve with respect to the number of PEs N_i allocated to it. Initially, the curve slopes downward, indicating that the performance increases. At some point, we assume performance saturates, and the curve starts to slope upward, indicating that communication and other overheads are starting to dominate τ_i^{comp}. The iteration time of the complete coupled application, τ_G, is a complex function of its PE allocation to its constituents. If the models are fully concurrent and never wait on each other except at the end of an iteration, the τ_G will be the same as that of the slowest constituent (i.e., the one with the maximum value of τ_i^{comp}). In more realistic cases, however, the situation is more complex.

We assume that two types of allocation situations may occur. The first is where the current PE allocation across constituents is performaing poorly and needs to be rebalanced. We consider this an *optimization* phase. The other is where a new PE is to be donated to the application's global PE pool, and thus a recipient constituent needs to be chosen. We call this an *expansion* phase. The two phases may be interleaved, of course. *Contraction*, or rescinding a PE from the global pool, would also be a useful operation but is not considered in this paper; it will be examined in future work.

Optimization. The algorithm for optimization is described in Figure 1. We assume that each pair of constituents, a *donor* and a *recipient*, represents a direction for possible reallocation (i.e., taking a PE away from the donor and giving it to the recipient). The algorithm reallocates in a direction as long as performance is improved. The directions that are tried are selected by the algorithms SEL1 and SEL2 given in Figure 3 and 2, respectively. When it is detected that a direction works, the opposite direction is marked false, to indicate that it should not be considered. Once a direction fails to improve performance, the last reallocation is undone. The algorithm tries all possible directions for reallocation at least once.

```
{ n_models is the number of models. models_comp is an array of model
references sorted in decreasing τ_i^comp order.}
{First search.}
for i = n_models − 1 to 1 do
  {Current fastest.}
  donor = models_comp[i];
  for j = 0 to i − 1 do
    {Current slowest.}
    recipient = models_comp[j];
  end for
  if [donor, recipient] pair is viable direction then
    return [donor, recipient];
  end if
end for
{Second search, if slow model can no longer benefit from more proces-
sors.}
for i = 0 to n_models − 1 do
  {Current slowest.}
  donor = models_comp[i];
  for j = i + 1 to n_models − 1 do
    {Next slowest.}
    recipient = models_comp[j];
  end for
  if [donor, recipient] pair is viable reallocation direction then
    return [donor, recipient];
  end if
end for
{No viable direction.}
return [-1, -1];
```

Figure 3: SEL1 selection algorithm.

```
call EMCTWorld_init(comms, nprocs, lbm_info)
call EMCT_LB_init(current_iter, end_iter)
if(Get_emctstatus() == EMCT_BEGIN) then
!Initialize model
endif
do i = current_iter, end_iter
  call EMCT_LB_profile(Routs, comms)
  emct_status = Get_emctstatus()
  if(emct_status == EMCT_SHRINK) then
  !Redistribute data
    call EMCT_LB_resizeset(emct_status, comms)
  elseif(emct_status == EMCT_EXPAND) then
    call EMCT_LB_resizeset(emct_status, comms, i,
                           end_iter)
    !Redistribute data
  elseif(emct_ststus == EMCT_PRESERVE) then
    call EMCT_LB_resizeset(emct_status, comms)
  endif
  if(emct_status /= EMCT_RUN)
    !Recompute communication schedule with
    !connected models
  endif
  !Do own job for simulation
enddo
```

Figure 4: Fortran skeleton code for an MMCT-based MCM.

The performance of a constituent may be subject to factors other than the number of processors, such as OS jitter, communication bottlenecks caused by other jobs, or computational changes in any particular iteration. We detect such an anomaly by using both τ_G and the sum $\sum_{i=1}^{N} \tau_i^{\text{comp}}$. If a reallocation direction reduces one of these quantities but not the other, we consider this an ambiguous situation and continue to reallocate in this direction until we can unambiguously determine that this direction negatively affects performance.

To choose the donor and recipient constituents, we tried two different algorithms. The SEL1 algorithm uses only a constituent's computation time τ_i^{comp} to make its decisions. The SEL2 algorithm uses a constituent's τ_i^{comp} and the ratio $\delta\tau_i^{\text{comp}} = \tau_i^{\text{comp}}/N_i$. We call the term $\delta\tau_i^{\text{comp}}$ the *derivative*, which is under the assumption of perfect intraconstituent load balance, a finite-difference approximation of $d\tau^{\text{comp}_i}/dN_i$; the uncertainty in the derivative $\delta\tau_i^{\text{comp}}$ is proportional to load imbalance within the ith constituent. For both SEL1 and SEL2, the guiding heuristic is that we should try to speed the slowest constituent (i.e., the one with the greatest τ_i^{comp}).

SEL1 proceeds by iterating through the constituents in increasing order of their τ_i^{comp}, choosing each in turn as a candidate donor. For each candidate donor, the slowest constituent is chosen as a recipient. This pair is used as a candidate direction, and the first viable direction is used. If no viable directions can be found, we assume that the slowest constituent has reached the point of negative returns. SEL1 then tries to donate from the slower constituents. Details are given in Figure 3.

We noted that SEL1 did not always work well, as discussed in Section 5. To address this issue, we modified the algorithm to also consider the derivative. SEL2 proceeds by iterating through the constituents in increasing order of their derivative $\delta\tau_i^{\text{comp}}$, choosing each in turn as a candidate donor. For each candidate donor, the slowest constituent (one with the greatest τ_i^{comp}) is chosen as a recipient. This pair is used as a candidate direction, and the first viable direction is used. If no viable directions can be found this way, we assume that the slowest constituent has reached the point of negative returns. We then try to donate from the slower constituents instead. Details are given in Figure 2.

Expansion. The expansion algorithm uses the same algorithm as the optimization algorithm execpt that no donor constituent is chosen. Instead, the processor is a free processor obtained externally. The recipient can be chosen by two means. The first is based on τ_i^{comp}; the second is based on $\delta\tau_i^{\text{comp}}$.

4. Malleable MCT

We extended MCT for use in developing MCMs by adding a communicator management system that employs dynamic process management, a global constituent load diagnosis and balance calculation system that employs the algo-

rithms described in Section 3, and a central load balance manager that coordinates constituent cohort reapportionment. The current implementation of MMCT is a research prototype but is fully capable of supporting malleable parallel composition of models such as the testbed application described and evaluated in Section 5. Two new modules were added to MCT to implement these features. EMCTWorld implements dynamic process management to enable constituent cohort malleability. EMCT_LB implements interconstituent dynamic load balancing. The current MCT programming model is unchanged aside from these additions; in fact, legacy MCT applications can use the nonmalleability features in MMCT without modification. No modifications to the MCTWorld source code were required. For reference, we provide a skeleton Fortran code (Figure 4) to illustrate the usage of MMCT's methods in an MMCT-based MCM.

EMCTWorld represents a profound change in how MCT's constituent registry is initialized and functions. In MCT, constituents' individual communicators are created by splitting the global communicator. This top-down approach, however, is not appropriate for MCMs because any initial PE that belongs to MPI_COMM_WORLD cannot be removed at runtime; hence MCTWorld is considered a singleton class in MCT. In MMCT we employed a bottom-up approach that leverages MPI2's [2] dynamic process management facility to create and manipulate communicators. MPI-2 allows one to spawn new PEs that can be merged into the global communicator. In MMCT the global communicator is constructed by combining constituents' communicators via invocations of MPI_Comm_Connect() and MPI_Comm_Accept(). These operations reside in the EMCTWorld_init() method. Under MMCT, an MCM with N constituents is initialized with a global communicator containing N PEs, one per constituent; these PEs are each constituent's *head nodes* and are the only PEs guaranteed to remain with the constituent throughout the full run of the MCM. PEs are added incrementally to each constituent's communicator to establish the initial configuration of $(P_1, \ldots, P_N)$ PE's. EMCTWorld_init() also registers the constituents' and global communicators in the MCTWorld module, allowing the rest of MCT to function as is.

The LoadBalance module encapsulates system constituent load information and implements the initialization, load assessment and rebalancing, and PE reallocation mechanisms. EMCT_LB_init() initializes the LoadBalance module by referring to communicators registered in the MCTWorld object. ECMT_LB_profile() is the constituent load balance analysis system and is called periodically in the global time integration loop of an MCM. This function is typically invoked at the beginning of the time loop of the malleable constituent to measure the constituent's computation and coupling times, τ_i^{comp} and τ_i^{coup} (e.g., execution times for routines such as MCT_Send(), MCT_Recv(), and MCT_Wait()), respectively. This function also serves as a synchronization point in the MCM, which is vital to ensuring constituent cohort reallocation proceeds correctly. EMCT_LB_resizeset() performs constituent cohort resizing. Each constituent PE cohort undergoes one of three actions—*expand*, *shrink*, or *preserve*—that result in respective increased, decreased, or unchanged size.

Each PE in an MMCT-based system has a *status* that evolves as the system runs. Five possible status values exist—BEGIN, SHRINK, EXPAND, PRESERVE, and RUN. The process status can be queried at any time by calling the MMCT function Get_emctstatus(). BEGIN indicates the malleable coupled model is to be initialized. SHRINK, EXPAND, and PRESERVE indicate constituent cohort reallocation must occur, with appropriate action required to resize allocated memory for data and/or data redistribution. RUN indicates no cohort reapportionment is necessary; that is, the system is in between load diagnosis or rebalancing events. Note that when the PE status is not RUN, constituents will likely require re-handshaking of interconstituent parallel data transfer communication schedules.

The centralized load balance manager (LBM) runs on a physical processor distinct from the MPI global communicator. It communicates with constituents' head nodes using out-of-band, socket-based communications; thus the communications burden on the LBM scales as the number of an MCM's constituents rather than the total number of MPI PEs, lowering the likelihood the LBM will be a bottleneck. Socket-based communications were chosen to separate concerns between the LBM and issues of the MPI-based malleable simulation environment, such as communicator management.

The LBM is initialized with the machine information of the cluster on which the MCM will run and sends a list of machines and/or free PEs to the MCM's constituents' head nodes. The LBM receives profiling data reports from each constituent's head node; these reports are generated by constituents' calls to ECMT_LB_profile(). The LBM analyzes constituents' profiling data using one of its load balance algorithms, computing a refined resource allocation for the MCM. The LBM sends reconfiguration decisions to each constituent's head node, with orders on whether the constituent must SHRINK, EXPAND, or PRESERVE its cohort; for SHRINK or EXPAND decisions, a list of machines/PEs to be respectively eliminated or added to the cohort is included.

The runtime architecture for a simple MMCT-based MCM is depicted in Figure 6. Constituents' argument lists must be extended to include the following arguments: NPEi, the initial number of PEs assigned to the ith constituent; LBM_HOST, the hostname of machine where LBM will run; and LBM_PORT, the port number constituents' head nodes will use to communicate with the LBM. The MCM application is invoked by first initiating execution of the LBM, followed by execution of the MPI-based coupled model using mpiexec:

```
mpiexec -np 1 ./constituent1 $NPE1 $LBM_HOST $LBM_PORT  : ...
```

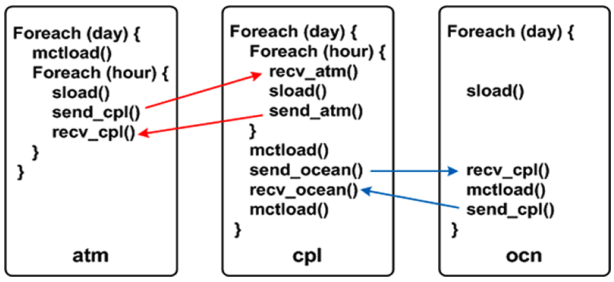

Figure 5: Climate benchmark application.

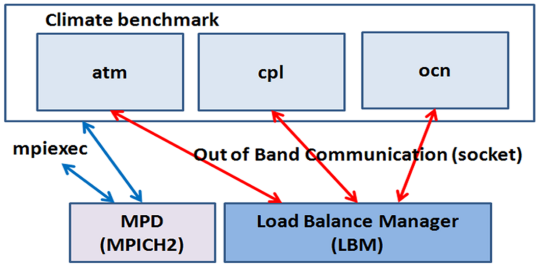

Figure 6: Runtime architecture for the Climate benchmarkwith MPD and LBM.

```
... : -np 1 ./constituentN $NPEN $LBM_HOST $LBM_PORT
```

Runtime overhead for the LBM was measured as part of the testbed experiments described in Section 5. The LBM, out-of-band LBM/MCM, communications, constituent profiling, and optimization algorithms occupied approximately 3% of the total wall-clock runtime—well below the performance gains realized by enabling interconstituent dynamic load balance. This initial result augurs well for MMCT's viability in implementing model coupling malleability.

The prototype MMCT provides a scientist-friendly programming model for implementing constituent cohort malleability in MCMs. Creating a fully functional MCM still requires the model developers to change the number of PEs in the middle of a run, with accompanying reinstantiation and reinitialization of constituents and re-handshaking of interconstituent $M \times N$ connections. These tasks can be handled by explicitly coding the necessary state checkpointing and data redistribution mechanisms (which is what we did for our testbed application in Section 5); leveraging any system-level checkpoint/restart functionality that may already be present to provide fault tolerance; or, if the application supports it, using its own restart file mechanisms. For CCSM, we expect the easiest, most portable checkpointing strategy may be to leverage its restart file mechanism (for a description and example, consult Chapter 5 of [18]).

5. Results

To validate our algorithms and evaluate the performance of MMCT, we have built a malleable coupled model testbed application that mimics a coupled climate model running in parallel composition. The testbed consists of three constituents: atmosphere (atm), ocean (ocn), and coupler (cpl) that computes interfacial fluxes exchanged between atm and ocn. The atm grid is identical to that of CCSM's atmosphere's T42 spectral solver, with 64 latitudes and 128 longitudes. The ocn grid corresponds to CCSM's ocean grid, with 384 latitudes and 320 longitudes. The domain decompositions of these two-dimensional grids are described by using MCT GlobalSegMaps. Parallel data transfer schedules are stored in MCT Routers. The domain decompositions for this test case are purely one-dimensional by latitude. The cpl uses both atmosphere and ocean grids and performs intergrid interpolation of field and flux data. The amount of data is not large, but our goal in this work was to focus on the feasibility of the implementing a malleable MCT and using it with a variety of possible load balancing algorithms.Larger data sets would result in greater data redistribution costs, which would affect the scheduling and trade-offs of reallocation decisions, but not our primary contributions.

Figure 5 shows the simplified structure of the climate testbed and the communication pattern between atmosphere model, ocean model, and coupler. Each model invokes four routines denoted as sload(), mctload(), send(), and recv() inside the time evolution loop. The sload() functions in atm, cpl, and ocn use the sleep() function to simulate the amount of time taken by each model to integrate its equations of evolution during intervals between coupling events. The amount of sleep time is determined from previously measured CCSM performance data [19] on a number of PE cohort sizes. The function mctload() performs several MCT data transformation operations, including intergrid interpolation, implemented as sparse matrix-vector multiplication, and time averaging/accumulation of state/flux data. Taken together, these loads provide a plausible picture of constituent load balance in a coupled climate model. Interconstituent communications occur between atm and cpl and between ocn and cpl. These parallel data transfers are implemented by using MCT, encapsulated in interconstituent communication via send() and receive(). These $M \times N$ transfers comprise numerous point-to-point nonblocking MPI messages, but the overall transfers employ MPI_Waitall(); this means the overall parallel data transfer constitutes blocking communication. The intercommunication pattern (arrows in Figure 5) allows all models to run concurrently since the ocean model can perform its own work while the atmosphere model is doing computation and communication with the coupler in a nested time loop representing 24 hours.

We benchmarked the optimization and expansion algorithms and the performance of MMCT LBM on Argonne National Laboratory's Fusion cluster. Fusion contains 320 compute nodes, each with dual 2.53 GHz Xeon quad core processors and 36 GB or 96 GB memory. The nodes are connected by InfiniBand QDR.

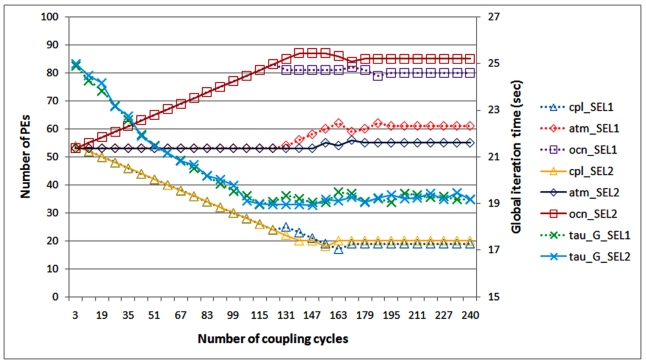

Figure 7: Optimization with SEL1 and SEL2 for INIT1 case: SEL1 found (19, 61, 80) with τ_G = 19.3s was found at the 195th coupling cycle, and SEL2 found (20, 55, 85) with τ_G = 19.3s at the 187th coupling cycle.

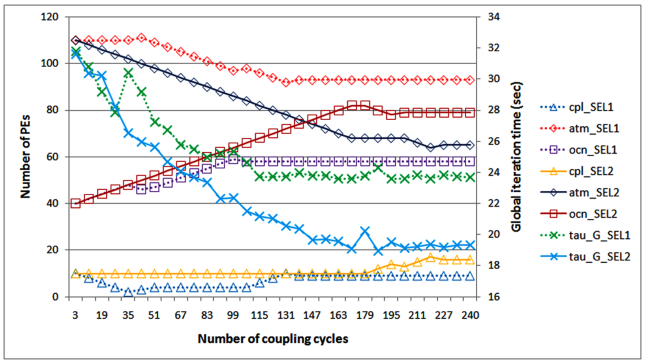

Figure 8: Optimization with SEL1 and SEL2 for the INIT2 case: SEL1 found (9, 93, 58) τ_G = 23.7s at the 139th coupling cycle, and SEL2 found (16, 65, 79) τ_G = 19.4s at the 227th coupling cycle.

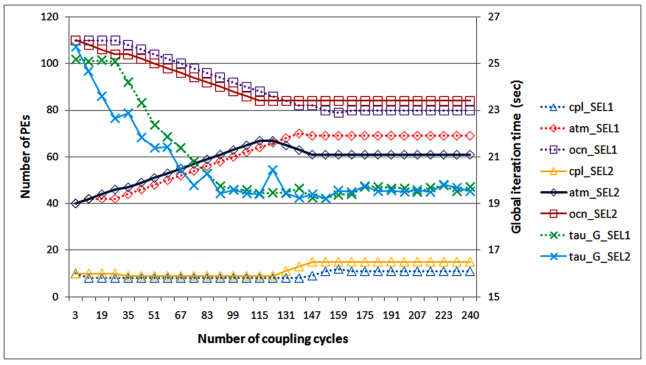

Figure 9: Optimization with SEL1 and SEL2 on initial configuration for the INIT3 case: SEL1 found (11, 69, 80) with τ_G = 19.6s at the 167th coupling cycle, and SEL2 found (15, 61, 84) τ_G = 19.6s at the 155th coupling cycle.

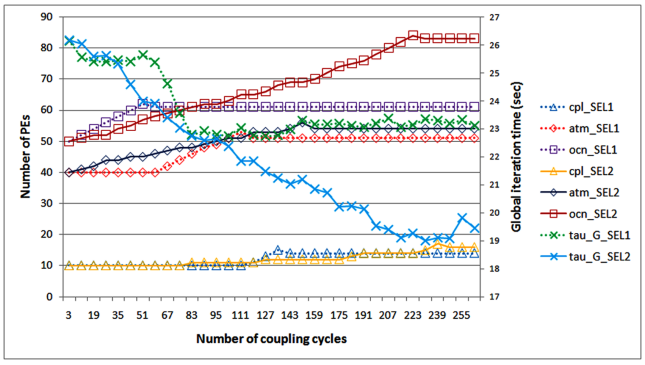

Figure 10: Expansion algorithm with SEL1 and SEL2: SEL1 found (14, 51, 61) with τ_G = 23.1s at the 143rd coupling cycle, and SEL2 found (16, 54, 83) with τ_G = 19.5s at the 247th coupling cycle.

The optimization algorithm was tested by running the climate model simulator benchmark with three initial PE allocations. For all these allocations the initial global processor pool has 160 PEs, a number drawn from previous benchmarking data [19]. Through trial and error, CCSM developers had discovered that a PE configuration $(N_{\text{cpl}}, N_{\text{atm}}, N_{\text{ocn}}) = (16, 64, 80)$ worked well for the model for its standard resolution and configuration; in the discussion that follows, we refer to this as the "ideal" configuration.

For reference purposes, we ran the benchmark with the ideal constituent PE cohort configuration, with the LBM active but no PE redistribution performed. This setup allowed us to measure for our simulator the "ideal" value of τ_G = 19.5s, analogous to that found by hand tuning CCSM. We will use this ideal value to estimate PE cohort reallocation overhead.

We performed three experiments with *try_count_limit* = 2, shown in Figure 1, and differing initial allocations. In the discussion that follows we describe model configurations as the triple $(N_{\text{cpl}}, N_{\text{atm}}, N_{\text{ocn}})$. The first initial allocation, INIT1 = (54, 53, 53), was as uniform an apportionment of PEs between constituents as possible; this allocation overprovisions cpl with PEs. The second initial allocation, INIT2 = (10, 110, 40) overprovisions the atm constituent. The third initial allocation, INIT3 = (10, 40, 110), overprovisions the slowest constituent (ocn).

Table 1 summarizes throughput improvements realized by performing cohort reallocation employing the SEL1 and SEL2 criteria. For the INIT1 case, values of τ_G obtained by the SEL1 and SEL2 algorithms were almost the same as the ideal τ_G. Both SEL1 and SEL2 improved the throughput by 22% from the initial allocation. Moreover, the final allocations obtained by SEL1 and SEL2 were similar to the ideal allocation; the reason is that all constituents had

Table 1: τ_G (seconds)

Alg., Init. alloc.	Initial τ_G	SEL1	SEL2
Opt., INIT1	24.9	19.4	19.5
Opt., INIT2	31.7	21.8	19.5
Opt., INIT3	25.9	20	19.9
Exp.	26.3	22.8	21.8

Table 2: Reconfiguration and $M \times N$ Re-handshaking timings (sec).

Operation	Remove PE	Create PE (intra-cohort)	Create PE (extra-cohort)
Reconfig	0.05	0.1	3.2
Reroute	0.44	0.45	0.52

sufficient PEs so that reconfiguration for any donor/recipient pair resulted in smooth changes to τ_G, allowing both SEL1 and SEL2 to find near-ideal allocations. For the INIT2 and INIT3 cases, both SEL1 and SEL2 found good cohort PE allocations but were not always able to match the ideal value of τ_G. Instabilities in our benchmark's global iteration time is the likely cause. For example, underprovisioning cpl often makes the τ_G noisy because it adversely impacts interconstituent communications times, notably the hourly interactions with atm.

For the INIT2 case, SEL2 performed superior load balancing to SEL1 because SEL2 can avoid choosing constituents with a large derivatives as either donor or recipient. Moreover, the fluctuation caused by a constituent that is allocated too few processors can be reduced as its PE cohort size is increased. Additionally, SEL2 always tries to reallocate PEs in a way that reduces noise. For the INIT2 case, in 83% of the runs (5 times out of 6) SEL2 found an allocation that resulted in a $\tau_G < 20$ s. The other run was very close, with $\tau_G = 20.6$ s. By contrast, SEL1 found a near-ideal configuration in only 33% (2 times out of 6) of the runs. As a result, SEL2 reduced τ_G by 38.4% on average for this case. For the INIT3 case, both SEL1 and SEL2 found a near-ideal allocation ($\tau_G < 20$s) 83% of the time (5 times out of 6), with the remaining trials for SEL1 and SEL2 producing values of τ_G of 22.25 s and 21.75 s, respectively. On average, SEL1 and SEL2 reduced τ_G by about 23% from its initial value.

Figures 7, 8, and 9 display the time evolution of constituent PE cohort configurations determined by the SEL1 and SEL2 algorithms for the INIT1, INIT2, and INIT3 cases, respectively. Superimposed on these plots are curves for the time evolution of τ_G (the scale for τ_G is on right vertical axis in each plot). The time unit for all of these plots is the LBI, which corresponds to the coupling cycle (one model-day). SEL2 reduced τ_G smoothly. In Figure 7, SEL2 found an optimal allocation of (20, 55, 85) with $\tau_G = 19.3$ s, slightly better than the ideal value. This implies the allocation for constituents could differ from their ideal allocations depending on the degree of concurrency of the coupled model. Figure 8 shows the time evolution of SEL1 and SEL2 for the INIT2 case. Here, SEL1 chose cpl as the donor, handing over PEs to ocn at the fourth coupling cycle even though cpl was starved for resources; cpl ceased donating PEs to ocn at the 35th iteration and took back two PEs from ocn to undo previous reconfiguration, an action caused by large increments in τ_G. By this point, however, cpl had already lost 6 of its initially allocated 10 PEs, making it so badly underallocated that it made τ_G unstable, disrupting the optimization process. By contrast, SEL2 chose atm as the donor because it realized that atm had sufficient processors and its iteration time would not be greatly changed even if it lost a processor; this decision stabilized τ_G until optimization was completed.

Figure 9 demonstrates that both SEL1 and SEL2 determined good, though slightly different allocations for the INIT3 case. SEL1 encounters a similar problem here as with INIT2, but less dramatic. SEL1 chose cpl as the donor for atm early in the run. But, cpl did not lose as many processors as in the INIT2 case because the benefit generated by adding processors to atm was often less pronounced than the disadvantage resulting from removing PEs from cpl. Moreover, cpl did not donate any PEs to ocn because ocn was initially overallocated. Hence, cpl maintained 8 PEs, which is enough to reduce—though not eliminate—the fluctuations in τ_G caused by underprovisioning cpl.

To evaluate the expansion algorithm, we ran the climate benchmark on an initial configuration of (10,40,50) PEs with *try_count_limit* = 2. We chose this initial allocation as it represents a 38% across-the-board cut in constituents' resource allocations in the ideal configuration of (16,64,80). Table 1 shows τ_G when run with the expansion algorithm using the SEL1 and SEL2 criteria. Both variations reduce τ_G, but the performance was worse than we expected. Figure 10 shows the time evolution of the PE cohorts for the expansion algorithm with SEL1 and SEL2; time evolution curves for corresponding τ_G values are also included (refer to right vertical axis for scale). The expansion algorithm with SEL1 found the ideal allocation for one of the six trials because it just kept adding processors to ocn, which was the slowest model. This caused the communication overhead imposed on other constituents to be large and unstable, since the other constituents maintained their PE cohort sizes as the ocn cohort was expanded. Eventually, ocn could not be a recipient once the communication or noise was larger than the decreased time by adding a processor to ocn. At the 51st coupling cycle, τ_G increased while ocn gained processors, meaning that ocn could not be a recipient during the rest of the expansion process. When recipients were selected with SEL2, the algorithm determined good allocations 50% of the time (3 times out of 6) and on average improved τ_G by 13%. For the three trials where this approach failed, we found that after adding processors to atm, whose derivative was biggest at the beginning of iteration, τ_G slightly increased (by less than 0.05 s). We believe this noise in τ_G originates in unstable communication costs and overhead; future work will seek to address these fluctuations by appropriate filtering or weighting.

Our performance results demonstrate MMCT can support coupling malleability and achieve faster runtimes. An important question is: At what cost? That is, what is the overhead associated with redistributing a constituent's state data from the old cohort to the new one? Table 2 summarizes wall clock time measured from ocn with under the expansion algorithm, including reconfiguration (the cost of adding or removing a PE) and rerouting (re-handshaking the $M \times N$ communications schedules). In our test case, the model grids are still rather coarse, and data redistribution costs were low compared to τ_G. The overhead for rerouting is nearly equal to that of adding or removing a PE if the operation is performed on a machine where the current cohort resides. The wall clock time to create a processor on an extra-

cohort machine, however, is more than 3 seconds. We believe that this overhead is caused by MPI-2 mechanisms to spawn processes. Consequently, when reconfiguration occurs for removing processors or creating a processor on an intra-cohort machine, it would increase the ideal global iteration time by 3% approximately; if it creates a processor on an extra-cohort machine, it increases the ideal global iteration time by 19%. We believe these overheads can be reduced by weighing the frequency of reallocation with the possible benefits. This will be an area of future study.

6. Conclusion and Future Work

Coupled systems are an important feature on the scientific computing landscape. Achieving the scalability required for these systems to use exascale platforms will prove a tremendous challenge. We have identified top-level load balance to be a crucial problem. We have shown the feasibility of malleable coupling by developing a simple yet effective research prototype that is an extension of the widely used Model Coupling Toolkit. This coupling malleability mechanism, MMCT, extends the MCT API to runtime reallocation of PE resources in a wide class of parallel coupled models. Benchmark results for the simple load-balancing strategies currently included in MMCT demonstrate their effectiveness; the schemes reported here provide significant improvements in model throughput with acceptably low overhead. Both of these performance characteristics will be studied further, with an eye to their optimization.

Future research and development will focus on creating a production environment. Further investigation of the current load balance schemes in our LBM is warranted, along with investigations into a wider variety of load-balancing strategies. Such research will become critical once we move beyond the current assumption of a purely parallel composition to nested compositions such as those used in CCSM4. MMCT's LBM requires further design to make it open to inclusion of third-party load diagnosis and management software. Additionally, MMCT's architecture must be expanded to offer interoperability with current and emerging parallel application checkpointing mechanisms.

Acknowledgments

We thank the Laboratory Research Computing Resource Center at Argonne National Laboratory for access to their Fusion cluster. Argonne National Laboratory is supported by the U.S. Department of Energy, under Contract DE-AC02-06CH11357. Work at SUNY Binghamton is supported by the National Science Foundation.

References

[1] J. W. Larson, Ten organising principles for coupling in multiphysics and multiscale models, ANZIAM Journal 48 (2009) C1090–C1111.
[2] The Message Passing Interface (MPI) standard, `http://www-unix.mcs.anl.gov/mpi/`.
[3] F. Bertrand, R. Bramley, D. E. Bernholdt, J. A. Kohl, A. Sussman, J. W. Larson, K. Damevski, Data redistribution and remote method invocation for coupled components, J. Parallel Distrib. Comput. 66 (7) (2006) 931–946.
[4] D. G. Feitelson, L. Rudolph, Towards convergence in job schedulers for parallel supercomputers, in: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, Springer-Verlag, 1996, pp. 1–26.
[5] R. Sudarsan, C. Ribbens, ReSHAPE: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment, in: Parallel Processing, 2007, ICPP2007., IEEE, 2007.
[6] K. E. Maghraoui, B. K. Szymanski, C. Varela, An architecture for reconfigurable iterative mpi applications in dynamic environments, in: Proceedings of the Sixth International Conference on Parallel Processing and Applied Mathematics (PPAM2005), number 3911 in LNCS, Springer Verlag, 2005, pp. 258–271.
[7] K. El Maghraoui, T. J. Desell, B. K. Szymanski, C. A. Varela, Dynamic malleability in iterative MPI applications, in: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, CCGRID '07, IEEE, 2007, pp. 591–598.
[8] S. S. Vadhiyar, J. J. Dongarra, SRS - a framework for developing malleable and migratable parallel applications for distributed systems, Parallel Processing Letters. 13 (2) (2003) 291–312.
[9] S.-H. Ko, N. Kim, J. Kim, A. Thota, S. Jha, Efficient runtime environment for coupled multi-physics simulations: Dynamic resource allocation and load-balancing, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 349–358.
[10] Institute for Combinatorial Scientific Computing and Petascale Simulations, `http://www.cscapes.org/`.
[11] J. Larson, R. Jacob, E. Ong, The Model Coupling Toolkit: A new Fortran90 toolkit for building multi-physics parallel coupled models, Int. J. High Perf. Comp. App. 19 (3) (2005) 277–292. doi:10.1177/1094342005056115.
[12] R. Jacob, J. Larson, E. Ong, M×N communication and parallel interpolation in CCSM3 using the Model Coupling Tookit, Int. J. High Perf. Comp. App. 19 (3) (2005) 293–308. doi:10.1177/1094342005056116.
[13] J. C. Meza, Steepest descent, in: Wiley Interdisciplinay Revies: Computational Statistics, Vol. 2, 2010, pp. 719–722.
[14] Model Coupling Toolkit Web site, `http://mcs.anl.gov/mct/`.
[15] Community Climate System Model Web Site, `http://www.cesm.ucar.edu/models/ccsm4.0/`.
[16] A. P. Craig, B. Kaufmann, R. Jacob, T. Bettge, J. Larson, E. Ong, C. Ding, H. He, CPL6: The new extensible high-performance parallel coupler for the Community Climate System Model, Int. J. High Perf. Comp. App. 19 (3) (2005) 309–327. doi:10.1177/1094342005056117.
[17] I. Foster, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison Wesley, Reading, Massachusetts, 1995.
[18] Community Climate System Model Version 4.0 Users' Guide, `http://www.cesm.ucar.edu/models/ccsm4.0/ccsm_doc/book1.html/`.
[19] J. W. Larson, R. L. Jacob, E. T. Ong, A. Craig, B. Kauffman, T. Bettge, Y. Yoshida, J. Ueno, H. Komatsu, S.Ichikawa, C. Chen, P. Worley, Benchmarking a parallel coupled model, poster presented at Supercomputing '03.

Government License