2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Malleable Model Coupling with Prediction

Daihee Kim
State University of New York at Binghamton
dkim17@cs.binghamton.edu

J. Walter Larson
Argonne National Laboratory
larson@mcs.anl.gov
University of Chicago
The Australian National University

Kenneth Chiu
State University of New York at Binghamton
kchiu@cs.binghamton.edu

***Abstract*—Achieving ultrascalability in coupled multiphysics and multiscale models requires dynamic load balancing both within and between their constituent subsystems. Interconstituent dynamic load balance requires runtime resizing—or *malleability*—of subsystem processing element (PE) cohorts. We enhance the Malleable Model Coupling Toolkit's Load Balance Manager (LBM) to incorporate prediction of a coupled system's constituent computation times and coupled model global iteration time. The prediction system employs piecewise linear and cubic spline interpolation of timing measurements to guide constituent cohort resizing. Performance studies of the new LBM using a simplified coupled model testbed similar to a coupled climate model show dramatic improvement (≈77%) in the LBM's convergence rate.**

***Keywords*-MPI, Dynamic Load Balance, Model Coupling, Multiphysics Modeling, Multiscale Modeling**

I. Introduction

Simulation of complex multiphysics and multiscale—or, more generally, *coupled*—systems in science and engineering often relies on compound applications that incorporate multiple interacting subsystem models, or *constituents*. Coupled systems appear in numerous interdisciplinary research areas. The interactions between subsystems amount to interconstituent data dependencies; in distributed-memory parallel implementations interconstituent data transfer is parallel and frequently called $M \times N$ *transfer* [1], [2]. More generally, the transport and transformation (e.g., intermesh interpolation) of one subsystem's output into another's input are called *coupling* and, on multiprocessors, *parallel coupling* [3].

A key challenge in building scalable, parallel coupled systems is runtime resource allocation, that is, the mapping of constituents to collections of processing elements (PEs), or *cohorts*. In a coupled system using parallel composition—that is, where constituent PE cohorts are non-overlapping—scalability and computational intensity of the constituents inform sizing of their respective cohorts. Load balancing a parallel coupled model is complicated by its interconstituent data traffic and the wall-clock time it consumes. Resources (in our case PEs) must be allocated to constituents harmoniously to reduce the wait time caused by load imbalance and data-dependency-driven interconstituent synchronization. Resource allocation to a coupled model's constituents is generally performed through trial and error, consuming much time and effort. Frequently these resource allocations are static, even though the runtime load per constituent may vary with time. Furthermore, load-balance configurations are platform-dependent. Thus, developers of a parallel coupled model could benefit from infrastructure that supports automatic, runtime interconstituent load balance, both for implementing dynamic load balance at runtime and for determining static load-balance sweet spots. This need will become more pronounced with trends toward more complex coupled models (with increasing number of constituents) and exascale computer hardware (with increasing numbers of PEs).

We call the ability to reallocate constituent PE cohorts *malleability* [4]. We call a coupled model that allows dynamic interconstituent load balance a *malleable coupled model* (MCM). In previous work [5], we introduced the Malleable Model Coupling Toolkit (MMCT), an extension of the Model Coupling Tooklit (MCT; [6], [7]) that supports dynamic load balance in parallel coupled models. MMCT consists of MCT, a load balance manager (LBM), and a dynamic process and communicator management system. The present work focuses on improvements to the LBM. The simple and practical LBM originally developed for MMCT converges too slowly to support production use. Our objective is to improve the LBM's convergence properties and final load-balance solution.

We describe related work in Section II and malleable model coupling and MMCT in Section III. The improved, prognostic load-balancing algorithm is presented and its performance in a simplified coupled-model testbed is reported in Sections IV and V, respectively. In Section VI we summarize our conclusions and outline future work.

II. Related Work

Some research has been done to apply malleability to parallel iterative applications for dynamic load-balancing. SRS [8] allows a parallel application to reconfigure PE cohorts by stopping and restarting its execution. PCM/IOS [9], [10] enables profiling of parallel applications and triggering of reconfigurations to support malleable, iterative MPI applications. The ReSHAPE [11] framework changes PE allocations of malleable parallel applications during job scheduling for system resource utilization. Utera et al. [12] also dealt with malleability for efficient job scheduling. None of these approaches, however, are immediately applicable to malleable model coupling because they concentrate on applying malleability to monolithic parallel applications. Although a constituent's throughput is improved, the performance of the coupled model cannot be improved if other constituents perform poorly.

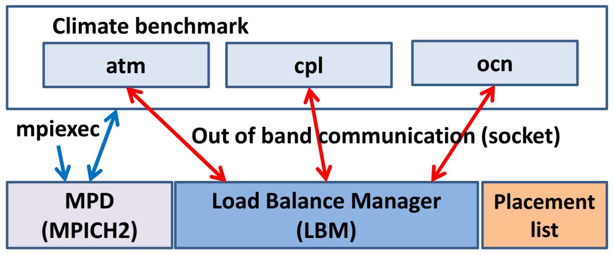
Climate benchmark
atm
cpl
ocn
mpiexec
Out of band communication (socket)
MPD (MPICH2)
Load Balance Manager (LBM)
Placement list

Fig. 1. Climate testbed runtime architecture with MPD and LBM. The benchmark consists of atmosphere and ocean constituents with coupler.

Ko et al. [13] introduced a coupled multiphysics simulation system that is able to optimize PE allocation. Their load-balancing algorithm, however, is applicable only to a coupled model with two constituents and with the assumption that computation time is reduced ideally by parallelization. The CSCAPES project [14] has as one of its foci dynamic load-balancing for parallel applications. Hypergraph-based repartitioning with Zoltan [15], one of CSCAPES contributions, performs dynamic load-balancing through data/computation migration within a model's PE cohort. Data/computation migration cannot, however, occur between different constituents.

III. Malleable Model Coupling and MMCT

A coupled model's state evolves in time as its N constituents solve their respective equations of evolution, exchanging flux and state data with their peers as required. Interconstituent data exchanges and attendant processing are called *coupling events* [3]. These events can be threshold-driven and unpredictable or regularly scheduled. A coupled system whose coupling events are scheduled and fall into a repeatable sequence within a constant time interval ΔT has a *coupling cycle* [3] with *period* ΔT; ΔT may be viewed as the irreducible overall "timestep" of the coupled system because it represents the minimum time over which all interconstituent data dependencies arise. For example, the Community Climate System Model (CCSM) [16], [17] has $\Delta T = 1$ model day.

The *global iteration time* τ_G and *constituent iteration time* τ_i represent the respective wall-clock times required to complete a coupling cycle by the coupled system and its ith constituent [5]. One may decompose τ_i as $\tau_i = \tau_i^{comp} + \tau_i^{coup}$, the sum of its *constituent computation* including intraconstituent communication time and (non-overlapped) *constituent coupling* wall clock times, respectively. The object of dynamic, interconstituent load-balancing is minimization of $\tau_G = \max\{\tau_i, \dots, \tau_N\}$, through constituent PE pool reallocation—malleability—to harmonize the values of $(\tau_i^{comp}, \tau_i^{coup}), i = 1, \dots, N$.

MMCT enables straightforward MCM construction by offering infrastructure for interconstituent PE reallocation and global PE cohort (i.e., MPI_COMM_WORLD) resizing. MMCT extends MCT with a dynamic process and communicator management system (PCMS) and centralized load-balance manager. The runtime architecture of a simple MMCT-based MCM is shown in Figure 1. At startup, the head node of each constituent is created by mpiexec [18], and the LBM sends it a PE placement list to initialize its cohort. The LBM communicates with each constituent's head node via socket-based, out-of-band communication. The LBM gathers and analyzes throughput information—values of τ_G and $(\tau_i^{comp}, \tau_i^{coup}), i = 1, \dots, N$—to make PE cohort reallocation decisions; its analyses are guided by a load balance algorithm, which is the central focus of this paper. The head node/process of each constituent gathers timings from all its nodes and provides the LBM summary statistics. The PCMS executes the LBM's decisions through dynamic process creation and termination. The system performs these operations over a predefined load balance interval (LBI), which corresponds to the coupling cycle period ΔT. At the end of each LBI, each constituent performs one of three actions: SHRINK (EXPAND) to reduce (increase) the number of PEs in its cohort, or PRESERVE to keep fixed the current number of PEs in its cohort. Resizings require checkpointing and redistributing data and rehandshaking of $M \times N$ interconstituent communications schedules. The former must be supplied by the user; the latter is an MCT function. Further details on the software implementation of MMCT and its relationship to MCT can be found in [5].

MMCT's LBM initially used a gradient descent-based method [19] to make its allocation decisions [5]. This algorithm worked by selecting a direction composed of two constituents: a *donor* constituent releases a PE that is subsequently allocated to a *recipient* constituent. Selection and reallocation occur until no other possible direction exists and τ_G is reduced. This approach was sufficient to demonstrate the viability of malleable model coupling, but it suffered from the following problems. First, it could reallocate only one PE per LBI because we had no clear method for determining how many PEs could be reallocated at a time. Second, all possible directions had to be tried because of the sensitivity of τ_G to τ_i^{comp} and τ_i^{coup}. Third, LBM decisions were adversely impacted by timing noise; if the changes in τ_G were on the order of or less than the timing noise in τ_i^{comp}, the LBM could easily make the wrong decision, undoing (confirming) a wise (an unwise) reallocation. In sum, the gradient descent-based LBM algorithm was slow to converge.

IV. Predictive Load-Balancing Algorithm

We assume that $\tau_i^{comp} = f_i(p_i)$, a function of p_i, the number of PEs in its cohort. The constituent computation time τ_i^{comp} decreases (increases) monotonically with p_i for $p_i < p_i^*$ ($p_i > p_i^*$). Scaling saturates at $p_i = p_i^*$, where τ_i^{comp} has its minimum. For $p_i > p_i^*$, overheads such as communication cost begin to dominate τ_i^{comp}. Our algorithm starts with an initial PE allocation $\vec{P}^0 = (p_1^0, \dots, p_N^0)$ to the system's N constituents. Analyzing measurements of τ_G and $\{\tau_1^{comp}, \dots, \tau_N^{comp}\}$ at the jth LBI, the LBM determines a PE cohort reallocation $(p_1^j, \dots, p_N^j) \to (p_1^{j+1}, \dots, p_N^{j+1})$. Quick and accurate optimization through reallocation requires prediction of τ_G for candidate PE cohort allocations. Prediction proceeds in two phases: (1) predict τ_i^{comp} (Section IV-A) and (2) use τ_i^{comp} to predict τ_G (Section IV-B). Load balance decisions are made in a subsequent *optimization* phase (Section IV-C) that utilizes forecast values of τ_i^{comp} and τ_G to identify more advantageous PE cohort allocations.

A. Constituent Computation Time Prediction

We estimate τ_i^{comp} using either modified piecewise linear interpolation or cubic spline interpolation. Values of τ_i^{comp} are measured in each LBI and stored with their corresponding number of PEs as points $(p_i^j, \tau_i^{(comp,j)})$, that is, with p_i^j (τ_i^{comp}) as the ordinate (abscissa). Analysis for each constituent is performed in its respective two-dimensional (p_i, τ_i^{comp}) space. After the first LBI, the algorithm has only one measurement (p_i^0, τ_i^{comp}). We shoot linearly from this point with slope $\delta\tau_i^{comp} = -\tau_i^{comp}/p_i^0$. This initial step is used for both the linear and cubic interpolation algorithms. For subsequent LBIs, timing data is collected, building up a timing database $\{(p_i^0, \tau_i^{(comp,0)}), \ldots, (p_i^j, \tau_i^{(comp,j)})\}$.

The piecewise linear interpolation algorithm (Figure 2) operates on these data as follows. If there is more than one measurement for a constituent, we divide the domain into two regions at $p_i = \frac{1}{2}\max\{p_i^0, \ldots, p_i^j\}$. Within each region, we connect the data points with line segments if there is more than one point, or else we use the aforementioned linear shooting technique for a single measurement. At the interface between the two regions, we extend the rightmost line segment in the left region and the leftmost line segment in the right region until the lines meet. If these two lines do not intersect within the region between the rightmost and the leftmost points, the points are merely connected. The justification for dividing the domain into two regions and interpolating separately is that in situations with few measurements this approach can more reasonably estimate $\min\{\tau_i^{(comp,j)}\}$ and its corresponding PE value p_i^* than can using piecewise linear interpolation over the whole domain.

The cubic spline interpolation algorithm (Figure 3) operates as follows. For the initial LBI, it uses linear shooting. For subsequent LBIs, there are multiple timing observations, and it interpolates over the entire domain using a global method [20].

B. Global Iteration Time Prediction

Values of τ_G are affected by the constituents' PE allocation and their intercommunication pattern, which are related to p_i and τ_i^{comp}, respectively. In the simplest case of purely concurrent constituent execution with communications isolated to the beginning or ends of their respective time loops, $\tau_G \geq \max\{\tau_i^{comp}, \ldots, \tau_N^{comp}\}$, tracking execution of the slowest constituent. Based on this observation, we defined a linear heuristic approach to predict τ_G with respect to PE allocation by using τ_i^{comp} and to determine the constituent whose τ_i^{comp} should be reduced to improve coupled model throughput. The heuristic model computes multiple *estimators* of τ_G, each a weighted sum of τ_i^{comp}. The forecast for τ_G is

$$\tau_G^{j+1} = \max\left\{\sum_{i=1}^{N} W_{ki}\tau_i^{(comp,j+1)}\right\},\ k = 1, \ldots, N_E. \quad (1)$$

```
{$\tau_i^{comp} = a \times x^3 + b \times x^2 + c \times x + d$}
{comp is an array of references storing $\tau_i^{comp}$ with the number of PEs and [a, b, c, d]. It is sorted in increasing order of the number of PEs.}
{$x$ is the number of PEs. $y$ is $\tau_i^{comp}$ with $x$.}
comp[0].a = comp[0].b = 0;
comp[0].c = -comp[0].y / comp[0].x;
comp[0].d = comp[0].y - (comp[0].c $\times$ comp[0].x);
**if** comp.size $>$ 1 **then**
    center_x = comp[comp.size-1].x / 2;
    comp[i].a = comp[i].b = 0 where $0 \leq$ i $<$ comp.size;
    left_comp = comp[0] to comp[m] where comp[i].x $\leq$ center_x;
    right_comp = comp[m+1] to comp[comp.size-1];
    Calculate [c, d] of elements of left_comp;
    Calculate [c, d] of elements of right_comp;
    **if** left_comp.size $\geq$ 1 **and** right_comp.size $\geq$ 1 **then**
        **call connect**;
    **end if**
**end if**
**procedure connect**
{y is set to 0 if its calculated value is less than 0}
**if** right_comp.size = 1 **then**
    y = left_comp[m].c $\times$ center_x + left_comp[m].d;
    Insert (center_x, y) into left_comp;
    Calculate [c, d] of (left_comp[m], left_comp[m+1], right_comp[0]);
**else if** linear lines of left_comp[m] and right_comp[0] intersect **then**
    Calculate intersect(x, y) and insert it to left_comp;
    Calculate [c, d] of (left_comp[m], left_comp[m+1], right_comp[0]);
**else**
    Calculate [c, d] of left_comp[m] and right_comp[0];
**end if**
**end connect**
```

Fig. 2. Algorithm for constituent computation time prediction by linear interpolation. The input to this algorithm is the comp array with x and y members set in each element. The output is the same array but with the a, b, c, d members properly computed and set in each element.

```
comp[0].a = comp[0].b = 0;
comp[0].c = -comp[0].y / comp[0].x;
comp[0].d = comp[0].y - (comp[0].c $\times$ comp[0].x);
**if** comp.size = 2 **then**
    center_x = comp[1].x / 2;
    **if** center_x $\leq$ comp[0].x **then**
        Calculate [c, d] of comp[0] and comp[1] via linear interpolation;
    **else**
        y = comp[0].c $\times$ center_x + comp[0].d;
        Insert (center_x, y) into comp; {Now, comp.size is 3}
    **end if**
**end if**
**if** comp.size $>$ 2 **then**
    Calculate [a, b, c, d] of elements of comp via cubic spline interpolation;
**end if**
```

Fig. 3. Constituent computation time prediction by cubic interpolation. Variables, input, and output are described in Fig. 2 and its caption.

The user supplies an $N_E \times N$ weight matrix $\mathbf{W}$ whose elements W_{ki} are tuned to represent interconstituent serializations and communications patterns. The number N_E of estimators is a complex, application-specific function of N and the number of interconstituent couplings present.

We present two simple examples to illustrate this approach. For two constituents in sequential composition with negligible communications costs, $\tau_G = \tau_1^{comp} + \tau_2^{comp}$. For a more complex example in which two constituents are running sequentially for 40% of their coupling cycle and the balance is concurrent execution, τ_G has two estimators and is

$$\begin{aligned}\tau_G &= \max\{[0.4(\tau_1^{comp} + \tau_2^{comp}) + 0.6\tau_1^{comp}],\\ &\quad [0.4(\tau_1^{comp} + \tau_2^{comp}) + 0.6\tau_2^{comp}]\}.\end{aligned}$$

```
{num_procs is an array storing numbers of PEs potentially reallocated for
each pair of a donor and a recipient}
{prediction_conuter indicates the upper bound of the total number of PEs
potentially reallocated in a direction}
call initialize;
repeat
  cur_iter_time = $\tau_G$;
  Collect or Update $\tau_i^{comp}$ and $\tau_G$ with current PE configuration;
  Perform linear or cubic spline interpolation for measurements of $\tau_i^{comp}$
  if cur_iter_time $\leq$ prev_iter_time then
    if [prev_donors, prev_recipients, prev_num_procs] exists then
      Increase the value of prediction_counter;
    end if
    Choose [donors, recipients, num_procs] using SELECTION;
    call update;
  else
    {Undo condition}
    Decrease the value of prediction_counter;
    [donors, recipients, num_procs]
    = [prev_recipients, prev_donors, prev_num_procs];
    call initialize;
  end if
  if [donors, recipients, num_procs] exists then
    reallocate([donors, recipients, num_procs]);
  end if
until [donors, recipients, num_procs] exists
procedure initialize
prev_iter_time = infinity;
[prev_donors, prev_recipients, prev_num_procs] = [-1, -1, -1];
end initialize
procedure update
prev_iter_time = cur_iter_time;
[prev_donors, prev_recipients, prev_num_procs]
= [donors, recipients, num_procs];
end update
```

Fig. 4. OPTIMIZATION algorithm. Measurements of τ_i^{comp} are interpolated using the piecewise linear algorithm (Fig 2) or cubic spline algorithm (Fig 3).

Each quantity in square brackets is a τ_G estimator. Note that each estimator can be reorganized into a weighted sum as in (1).

C. Optimization

The OPTIMIZATION algorithm (Figure 4) determines PE cohort reallocations to reduce τ_G. A PE cohort configuration $(p_1^j, \ldots, p_N^j)$ is a vector $\vec{P}^j \in \mathbb{N}^N$, and reallocation to $\vec{P}^{j+1} = (p_1^{j+1}, \ldots, p_N^{j+1})$ corresponds to the *vector difference* $\vec{P}^{j+1} - \vec{P}^j$ that defines the *direction* of the reallocation in PE space. Donor (Recipient) constituents in a reallocation $\vec{P}^j \rightarrow \vec{P}^{j+1}$ are identified by negative (positive) values of $p_i^{j+1} - p_i^j$; unchanged allocations correspond to $p_i^{j+1} = p_i^j$. The original MMCT LBM navigated this multidimensional space only one dimension and one PE at a time, making it slow and highly sensitive to measurement noise in τ_i^{comp} and τ_G.

The LBM's new optimization algorithm identifies possible reallocations that can comprise multiple donors and recipients, and many—as opposed to one—PEs may be moved between donor/recipient pairs. The result is larger steps through the PE configuration space, resulting in fewer time-consuming PE reallocations and reduced sensitivity to measurement noise.

The LBM optimizer collects and updates τ_i^{comp} and τ_G values at each LBI. Linear or cubic interpolation is performed for all available measurements of τ_i^{comp}. A candidate reallocation is determined by the SELECTION algorithm (Figure 5).

The previous reallocation direction is deemed successful (unsuccessful) if $\tau_G^{j+1} \leq \tau_G^j$ ($\tau_G^{j+1} > \tau_G^j$). The algorithm tries to select another direction after increasing the value of prediction counter. The prediction counter whose initial value is 2^2 is always incremented or decremented by a multiple of 2 from 2^0 to 2^4 and defines the upper bound of the total number of PEs that can be reallocated in a direction. Once a direction fails to improve throughput, the previous reallocation is undone to recover the previous PE allocation, and the value of prediction counter is decreased. PE configurations previously determined by the LBM will not arise unless a subsequent PE reconfiguration is undone. The SELECTION

```
{max_models is an array of models whose $\tau_i^{comp}$ is a component of term
in an estimator having the greatest value in decreasing term's value order.}
{pre_map is a map storing prediction information (pre_info) including the
slope of linear model or cubic spline model's tangent keyed by a PE
allocation. pre_info is retrieved by pre_map.find}
for $i = 0$ to $max\_models.size - 1$ do
  model_id = max_models[i].id;
  pre_info = pre_map.find(model_id, max_models[i].cur_npes);
  if linear_interpolation then
    decision_slope1 = pre_info.right_slope;
    decision_slope2 = pre_info.left_slope;
  else if cubic_spline_interpolation then
    decision_slope1 = pre_info.tangent_line_slope;
    decision_slope2 = decision_slope1;
  end if
  max_npes = the value of prediction_counter;
  if $decision\_slope1 \leq 0$ then
    recipient_cand = max_models[i];
    donor_cands = all other models except recipient_cand;
    donor_recipient_cands = all possible reallocating directions with PEs
    $\leq$ max_npes and corresponding target PE allocations;
    Pick [donors, recipients, num_procs] among donor_recipient_cands
    using DECIDE;
    if [donors, recipients, num_procs] exists then
      return [donors, recipients, num_procs];
    end if
  end if
  if $decision\_slopes2 \geq 0$ then
    donor_cand = max_models[i];
    recipient_cands = all other models except donor_cand;
    donor_recipient_cands = all possible reallocating directions with PEs
    $\leq$ max_npes and corresponding target PE allocations;
    Pick [donors, recipients, num_procs] among donor_recipient_cands
    using DECIDE;
    if [donors, recipients, num_procs] exists then
      return [donors, recipients, num_procs];
    end if
  end if
end for
return [-1, -1, -1];
```

Fig. 5. SELECTION algorithm using prediction information. It is used by the OPTIMIZATION algorithm.

```
Predict all $\tau_G$ of target PE allocations in donor_recipient_cands;
Sort donor_recipient_cands in increasing $\tau_G$ value order;
for $i = 0$ to $donor\_recipient\_cands.size - 1$ do
  PE_alloc = donor_recipient_cands[i].PE_allocation;
  if PE_alloc is untried and donor_recipient_cands[i].$\tau_G \leq$ cur_iter_time
  then
    donors = donor_recipient_cands[i].donors;
    recipients = donor_recipient_cands[i].recipients;
    num_procs = donor_recipient_cands[i].num_procs;
    return [donors, recipients, num_procs];
  end if
end for
return [-1, -1, -1];
```

Fig. 6. DECIDE algorithm used by the SELECTION algorithm.

algorithm chooses donor and recipient constituents by first identifying constituents whose τ_i^{comp} must be decreased to reduce τ_G based on its modeled value using (1). Constituents are sorted in decreasing order by their linear term contributions to the estimators in (1). The algorithm then iterates through the sorted constituents in search of donor/recipient candidates. A constituent is selected as a recipient (donor) candidate if its τ_i^{comp} can be reduced by adding (removing) PEs to (from) its cohort. The slope of either the piecewise linear model or the cubic spline model's tangent is used to determine whether τ_i^{comp} would be reduced by adding or removing PEs. Once a donor (recipient) is identified, the algorithm searches for a reallocation direction in PE space by identifying corresponding recipients (donors) among other constituents. We consider all possible PE reallocation vectors whose magnitude is less than or equal to the value of prediction counter.

The DECIDE algorithm (Figure 6) picks the PE space reallocation direction. It first calculates τ_G for all possible candidate PE reallocation vectors, sorting them keyed by increasing order of predicted τ_G. Then, a direction is chosen that results in a previously untried PE allocation that is expected to reduce τ_G.

Note that our system depends critically on the timing history of the various PE cohort configurations. If the load changes at runtime, as is known to occur in climate models, previous values of (p_i, τ_i^{comp}) will not necessarily reflect current conditions. A simple strategy would be deleting the history upon detecting such a situation, but this requires further investigation and is beyond the scope of the current work.

V. Performance Studies

To evaluate our algorithms, we have constructed a malleable coupled model benchmark application that emulates a parallel coupled climate model (Figure 7). The benchmark comprises an atmosphere (atm), ocean (ocn), and coupler (cpl) constituent. The ocn and atm exchange interfacial flux and state data via cpl; the cpl has grid information for both atm and ocn and carries out intergrid interpolation of state and flux data. The atm and ocn have the same grids as corresponding constituents of CCSM. MCT's functionality is used to describe the domain decomposition and perform $M \times N$ data transfer between constituents and data interpolation in cpl. We used two types of load: a simulated load using sleep() based on linear interpolation of CCSM benchmarking data [21] (sload()), and an example computational load using a parallel conjugate gradient (CG) solver to emulate the computation and intraconstituent communication done by each constituent to solve its state equations (rload()). The rload() CG solves a system of linear equations with 1500×1500 and 2000×2000 symmetric matrices for atm and ocn, respectively.

In addition to sload() or rload(), constituents call mctload(), send(), and recv() in each time step. The mctload() function performs MCT distributed data transformation, such as intergrid interpolation and time integration of flux and state data. Interconstituent communications between atm and cpl and between ocn and cpl are performed through the MCT

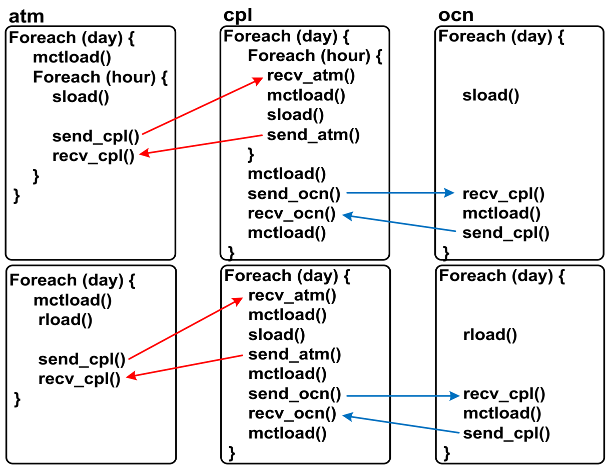

Fig. 7. Climate benchmark application using sload() (top) or rload() (bottom). The arrows indicate the intercommunication pattern.

send() and receive() communication calls. This pattern allows ocn to run concurrently while atm performs its computation and communicates with cpl. Two patterns are used, depending on which load functions are used. When models use sload(), atm is working with cpl in a nested time loop representing 24 hours [3]. When atm and ocn use rload(), atm communicates with the coupler once in a time step, because the matrix problems we used for rload() could not reasonably be sized small enough to send hourly data.

Performance analyses of the sload() testbed are presented in Sections V-A and V-B. A performance study of the rload() testbed is presented in Section V-C. Benchmarking was performed on a 64-node Linux cluster at SUNY Binghamton. Each node has dual 2.66 GHz Xeon dual-core processors with 8 GB memory. Nodes communicate via an InfiniBand 40 interconnect. The application was compiled with gcc/g++/gfortran version 4.4.6, and the MPICH2 version 1.2.1 MPI-2 implementation was used. We distributed each constituent's PEs randomly across the cluster. We found that doing so helps prevent network congestion and packet loss, minimizing timing variability.

In each study we evaluated the LBM optimizer, forecasting $\tau_i^{(comp,j+1)}$ using both piecewise linear (SEL1) and cubic spline (SEL2) interpolation schemes. Forecasts of τ_G used the following estimation scheme, which corresponds to a nested composition [22] with ocn on its own cohort and (atm, cpl) composed sequentially on a shared cohort:

$$\tau_G = \max\{\tau_{ocn}^{comp}, \tau_{cpl}^{comp} + \tau_{atm}^{comp}\}. \qquad (2)$$

Reallocation requires three coupling cycles to complete, because of synchronization between constituents and the LBM: cycle 1 waiting for the beginning of a timing cycle, cycle 2 for timing, and cycle 3 waiting for reallocation instructions from the LBM. The new allocation can then be used for the next cycle. Reallocation is an expensive process because of constituent synchronization wait time, operating system overhead, and constituent checkpointing and restart costs.

We also varied initial sizes of the global PE pool. For

	TESTBED1 using sload() with 160 PEs						TESTBED2 using sload() with 100 PEs						TESTBED3 using rload() with 40 PEs					
	INIT1		INIT2		INIT3		INIT1		INIT2		INIT3		INIT1		INIT2		INIT3	
	SEL1	SEL2	SEL1	SEL2	SEL1	SEL2	SEL1	SEL2	SEL1	SEL2	SEL1	SEL2	SEL1	SEL2	SEL1	SEL2	SEL1	SEL2
Inital τ_G	24.84	24.85	31.6	31.51	25.12	25.12	35.97	35.97	52.48	52.6	42.05	42.32	20.85	20.79	27.52	28.23	20.79	20.74
Final τ_G	19.07	19.05	19.15	19.11	19.17	19.12	26.18	26.36	26.09	26.02	26.15	26	15.21	15.38	15.33	15.97	14.94	15.77
Found at	24.8	45.6	27.2	56	28	34.4	41.6	40.8	23.2	41.6	19.2	35.2	42.8	31.2	30.8	34.8	54.8	38
#Realloc	6.2	11.4	7	14	7	8.6	10.4	10.2	5.8	10.4	4.8	6.8	10.7	7.8	7.7	8.7	13.7	9.5
Undo(%)	19.4	33.3	20	28.6	17.1	27.9	17.3	23.5	6.9	15.4	4.2	8.8	23.4	24.4	18.2	24.1	33.6	27.4

Fig. 8. Summary of LBM convergence properties and global iteration time statistics (seconds).

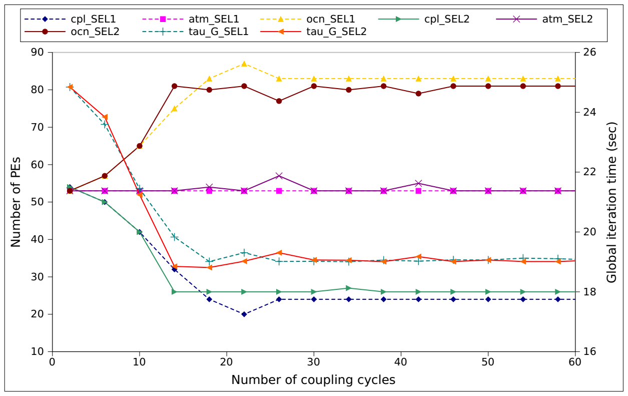

Fig. 9. Optimization using TESTBED1 with SEL1 and SEL2 for INIT1 case: SEL1 found (24, 53, 83) with τ_G = 19.1s at the 24th coupling cycle, and SEL2 found (26, 53, 81) with τ_G = 19.1s at the 44th coupling cycle.

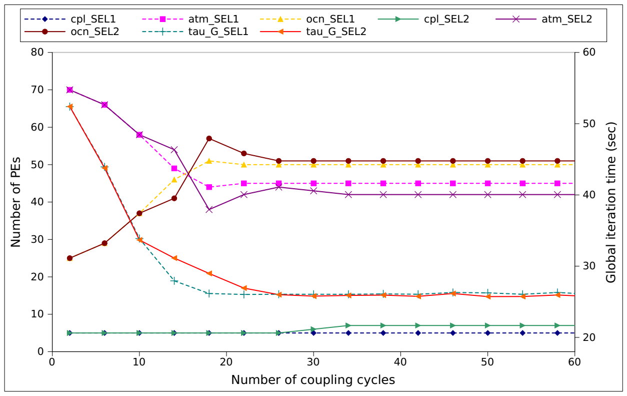

Fig. 10. Optimization using TESTBED2 with SEL1 and SEL2 for INIT2 case: SEL1 found (5, 45, 50) with τ_G = 26.1s at the 20th coupling cycle, and SEL2 found (7, 42, 51) with τ_G = 25.9s at the 32nd coupling cycle.

the sload() benchmark we used $P^0 = 160$ (TESTBED1) and $P^0 = 100$ (TESTBED2); the former a standard CCSM3 configuration, the latter an underprovisioned situation. For the rload() benchmark we assigned $P^0 = 40$ (TESTBED3). Throughout this section, we define a coupled model PE allocation as $\vec{P} = (N_{\mathrm{cpl}}, N_{\mathrm{atm}}, N_{\mathrm{ocn}})$.

The results for experiments using sload() and rload() are summarized in Figure 8. Values of τ_G are obtained by the LBM employing SEL1 and SEL2 schemes for three sets of initial conditions P^0. Sections V-A, V-B, and V-C provide further experimental setup details. The number of coupling cycles required to find a PE allocation solution is presented, as is the number of reallocations (#REALLOC) and percentage of reallocations that were undone (UNDO(%)).

A. *TESTBED1 using sload() with 160 PEs*

We ran TESTBED1 with $P^0 = 160$ PEs from three initial configurations: INIT1, with $\vec{P}^0 = (54, 53, 53)$, INIT2, with $\vec{P}^0 = (10, 110, 40)$, and INIT3, with $\vec{P}^0 = (10, 40, 110)$. INIT1 allocates PEs between constituents uniformly and oversupplies cpl with PEs. INIT2 and INIT3 were intended to oversupply atm and the slowest constituent ocn, respectively. We also ran TESTBED1 without the LBM, confirming the "ideal" value of $\tau_G = 19.13$s for $\vec{P} = (16, 64, 80)$, a configuration determined by CCSM developers through trial and error [21].

For all initial cases with SEL1 and SEL2, solutions for τ_G were almost equivalent to the ideal τ_G value with similar corresponding PE allocations to the ideal configuration. The number of coupling cycles required to finish the optimization is slightly different between SEL1 and SEL2. For INIT1 and INIT2, SEL1 takes fewer coupling cycles than does SEL2. Since sload() was implemented by using piecewise linear interpolation of the constituents' previously measured timing curves, SEL1 apparently is more suitable than SEL2, resulting in the higher rate of correct predictions than SEL2.

We performed the same experiments with the former LBM with INIT1, INIT2, and INIT3. The current optimization algorithm with SEL2 required approximately 23% of the coupling cycles required by the former LBM to reach final solutions for $\vec{P}$.

Figure 9 shows the convergence behavior of $\vec{P}$ and τ_G from INIT1 with LBM optimization using SEL1 and SEL2. In this case cpl donated redundant PEs to ocn mainly because both SEL1 and SEL2 predicted that the computation time of ocn must be reduced and balanced with the sum of computation time of cpl and atm to improve the throughput as per (2). SEL1 did not try to reallocate PEs for atm since it predicted that ocn's τ_i^{comp} cannot be reduced by either obtaining or shedding PEs after undoing the donation from ocn to cpl that occurred at 24th coupling cycle. SEL2, however, persisted in trying donations from ocn to cpl and atm several times because it mispredicted that ocn's τ_i^{comp} could be reduced by donations, and it undid four reallocations as a result.

B. *TESTBED2 using sload() with 100 PEs*

The TESTBED2 experiments have the following initial conditions: INIT1, with $\vec{P}^0 = (34, 33, 33)$; INIT2, with $\vec{P}^0 = (5, 70, 25)$; and INIT3, with $\vec{P}^0 = (5, 25, 70)$. The idea behind these experiments is that overall the system is underprovisioned with respect to an optimal configuration. Both SEL1 and SEL2 reduced τ_G on average by 27%, 50%, and 38% for the INIT1, INIT2, and INIT3 cases, respectively. For the same reason as in TESTBED1, SEL1 required fewer coupling cycles than did SEL2 to find a final solution for $\vec{P}$ for the INIT2 and INIT3 initializations. SEL1 and SEL2 required almost the same number of coupling cycles to find a final solution for $\vec{P}$ for INIT1.

Although sload()'s underlying performance model is a piecewise linearization of the constituent scaling curves, SEL1 and SEL2 perform comparably, since in the INIT1 case, τ_i^{comp} for atm and ocn decreased dramatically as their PE cohorts were increased in size by the LBM.

LBM convergence behavior for SEL1 and SEL2 with INIT2 initialization is shown in Figure 10. Here SEL1 kept forcing atm to donate PEs to ocn, which was the slowest model and chosen as a recipient. This donation was stopped, and ocn donated a PE to atm, which was chosen as recipient at the 20th coupling cycle because SEL1 predicted that atm had handed over a PE to ocn unnecessarily. Although the value of τ_G was reduced in this case, the LBM determined that the value of τ_G could be reduced further if atm obtained back a PE from ocn. In this case, the optimization was completed at the 20th coupling cycle with five reallocations.

SEL2 forced atm to donate too many PEs to ocn at the 16th coupling cycle because of inaccurate prediction. It caused two reallocations to find a more appropriate PE configuration. Moreover, cpl was chosen as a recipient instead of atm at 28th coupling cycle since SEL2 predicted that if ocn and atm were chosen as a donor and recipient, respectively, the value of τ_G would be increased because τ_i^{comp} of ocn would be greater than the sum of τ_i^{comp} of cpl and atm. This decision introduced two reallocations. As a result, the optimization with SEL2 was completed at the 32nd coupling cycle with eight reallocations.

C. TESTBED3 using rload() with 40 PEs

We used 40 PEs for the TESTBED3 experiments because that is approximately the scalability limit for the conjugate gradient method we use in rload(). LBM experiments were run from the following initializations: INIT1, with $\vec{P}^0 = (14, 13, 13)$; INIT2, with $\vec{P}^0 = (5, 30, 5)$; and INIT3, with $\vec{P}^0 = (5, 5, 30)$. In contrast to the TESTBED1 and TESTBED2 experiments, SEL1 required more coupling cycles than does SEL2 for the INIT1 and INIT3 cases. SEL1, however, for all initial cases, found a superior solution for $\vec{P}$ with lower τ_G than did SEL2. For example, the best solution found by SEL1 has $\tau_G = 14.94$s versus $\tau_G = 15.77$s for SEL2 (Figure 8).

For the INIT3 case (Figure 11), both SEL1 and SEL2 primarily chose atm as a recipient and ocn as a donor. Once atm was allocated enough PEs to be chosen as a donor, SEL1 reallocated PEs between cpl and atm several times until 8 and 11 PEs were assigned to cpl and atm, respectively. In contrast, SEL2 tried a reallocation that was subsequently undone with only one PE from atm to cpl once. SEL1 thus found a better solution of $\vec{P} = (8, 11, 21)$ than SEL2's $\vec{P} = (4, 14, 24)$. SEL1 and SEL2 forecasts for cpl and atm compute times (Figure 12)

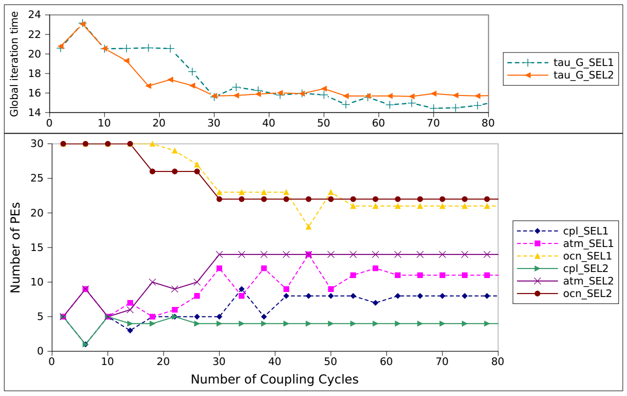

Fig. 11. Optimization using TESTBED3 with SEL1 and SEL2 for INIT3 case: SEL1 found (8, 11, 21) with τ_G = 14.9s was found at the 60th coupling cycle, and SEL2 found (4, 14, 22) with τ_G = 15.9s at the 28th coupling cycle. It clearly shows that cpl should be allocated more than 5 PEs to reduce τ_G.

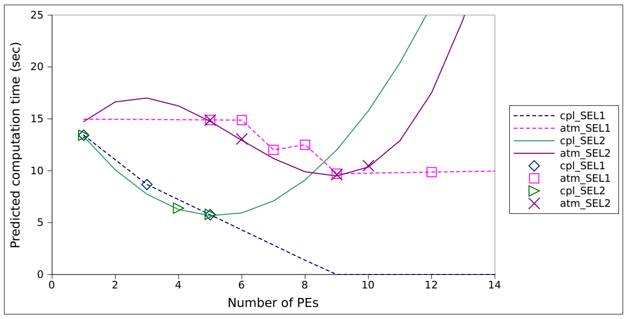

Fig. 12. Predicted τ_i^{comp} of cpl and atm with SEL1 on (5, 12, 23) at 31st coupling cycle and with SEL2 on (4, 10, 26) at 19th coupling cycle for TESTBED3/INIT3 case. Marker indicates measured τ_i^{comp}. Reallocation cannot be tried to allocate cpl more than 5 PEs with SEL2

show the information the LBM was using before donations occurred from atm to cpl at the 32nd and at 20th coupling cycles, respectively, for the INIT3 case. SEL1 predicted that cpl's τ_i^{comp} could be improved by expanding its allocation, in accordance with its linear τ_i^{comp} prediction line for more than five PEs—this in spite of the absence of timing data in this vicinity. SEL2's cubic spline interpolation, however, predicted that cpl's τ_i^{comp} would be increased dramatically if it was allocated more than five PEs. As a result, SEL1 performed more reallocations that were subsequently undone. SEL1 reallocated PEs for cpl aggressively because of defective linear prediction beyond its supporting measurements. In contrast, SEL2 reallocated PEs for cpl defensively because its model for τ_i^{comp} showed growth beyond the set of available measurements so that SEL2 performed fewer reallocations but obtained a solution for τ_G larger than that for SEL1.

These findings suggest a hybrid prediction strategy for τ_i^{comp}: Suppose that the SEL2 prediction curve for atm's τ_i^{comp} is instead a linear fit to measurements between 6 PEs and 10 PEs. In this case, SEL2 could reallocate PEs more defensively when atm was chosen as a donor. Thus, if we require aggressive load balancing between general scaling constituents, we could use linear extrapolation to predict τ_i^{comp} into data-poor regions of the PE domain and use cubic spline interpolation to predict τ_i^{comp} in well-observed regions of the PE domain. In the unusual situations where SEL2 forecasts for τ_i^{comp} dramatically decrease with increasing PE count into an unobserved region of the PE domain but grow dramatically with PE count in observed regions, using cubic spline extrapolation into the unobserved region and linear interpolation within the observed portion of the PE domain would enable aggressive reallocation.

For INIT2, SEL1 found a slightly better solution for $(\vec{P}, \tau_G)$ than did SEL2—the SEL1 solution for τ_G is approximately 4% lower than for SEL2. In this case, linear interpolation made slightly more accurate predictions than did SEL2. This can be recognized by comparing the percentage of allocations undone for SEL1 and SEL2—18.2% versus 24.1% for SEL1 and SEL2, respectively.

VI. Conclusions and Future Work

Coupled models of complex multiphysics and multiscale systems present grand computational science challenges. We are working toward a generic, runtime, dynamic load-balance coupling infrastructure that will bring ultrascalability in these systems closer to a reality. To this end, we have built on previous MMCT development work to improve its LBM algorithm's convergence rate and resultant load-balance configuration to increase coupled model throughput. We have improved the MMCT's LBM algorithm by predicting the computation time of constituents and the global iteration time of a coupled model through linear and cubic interpolation of constituent computation and coupling timings, with the object of reducing the coupled system's global iteration time. The predictive LBM allows testing of more beneficial reallocations, removal of unnecessary reallocations, and allocation of multiple PEs across multiple constituents, resulting in fast, efficient, and precise load-balancing. Experimental results using a simplified coupled model application similar to CCSM show that the new LBM with prediction discovers a well-balanced PE configuration for a coupled model approximately 77% faster than did the former algorithm. We have also distinguished different effects between linear and cubic spline interpolation prediction methods.

We plan to refine MMCT's LBM algorithm by applying it to more realistic, coupled model testbed problems. The current work has relaxed the previous assumption of parallel process composition to support nested and sequential compositions, but further work is needed to refine the τ_G estimation procedure. We also anticipate investigating other avenues of model throughput prediction that might improve the LBM, such as Monte Carlo coupled model throughput simulation based on performance statistics. Ultimately, we hope to apply the work described here to a full coupled model such as the Community Earth System Model.

Acknowledgment

Work at Argonne National Laboratory is supported by the U.S. Department of Energy under Contract DE-AC02–6CH11357. Work at SUNY Binghamton is supported by the National Science Foundation under award number 0941573.

References

[1] F. Bertrand, R. Bramley, D. E. Bernholdt, J. A. Kohl, A. Sussman, J. W. Larson, and K. Damevski, "Data redistribution and remote method invocation for coupled components," *J. Parallel Distrib. Comput.*, vol. 66, no. 7, pp. 931–946, 2006.

[2] R. Jacob, J. Larson, and E. Ong, "M×N communication and parallel interpolation in CCSM3 using the Model Coupling Tookit," *Int. J. High Perf. Comp. App.*, vol. 19, no. 3, pp. 293–308, 2005.

[3] J. W. Larson, "Ten organising principles for coupling in multiphysics and multiscale models," *ANZIAM Journal*, vol. 48, pp. C1090–C1111, 2009.

[4] D. G. Feitelson and 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] D. Kim, J. W. Larson, and K. Chiu, "Toward malleable model coupling," in *Procedea Computer Science (Proceedings of the International Conference on Computational Science, ICCS 2011)*, vol. 4, 2011, pp. 312–321.

[6] J. Larson, R. Jacob, and E. Ong, "The Model Coupling Toolkit: A new Fortran90 toolkit for building multi-physics parallel coupled models," *Int. J. High Perf. Comp. App.*, vol. 19, no. 3, pp. 277–292, 2005.

[7] "Model Coupling Toolkit Web site," http://mcs.anl.gov/mct/.

[8] S. S. Vadhiyar and J. J. Dongarra, "SRS - a framework for developing malleable and migratable parallel applications for distributed systems," *Parallel Processing Letters.*, vol. 13, no. 2, pp. 291–312, 2003.

[9] K. E. Maghraoui, B. K. Szymanski, and 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.

[10] K. El Maghraoui, T. J. Desell, B. K. Szymanski, and 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.

[11] R. Sudarsan and C. Ribbens, "ReSHAPE: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment," in *Parallel Processing, 2007, ICPP2007*. IEEE, 2007.

[12] G. Utrera, J. Corbaln, J. Labarta, and D. D. D. Computadors, "Implementing malleability on MPI jobs," in *Proceedings 13th International Conference on Parallel Architecture and Compilation Techniques (PACT'04)*. IEEE Computer Society, 2004, pp. 215–224.

[13] S.-H. Ko, N. Kim, J. Kim, A. Thota, and 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.

[14] "Institute for Combinatorial Scientific Computing and Petascale Simulations," http://www.cscapes.org/.

[15] U. Catalyurek, E. Boman, K. Devine, D. Bozdag, R. Heaphy, and L. Riesen, "Hypergraph-based dynamic load balancing for adaptive scientific computations," in *Procceedings, 21st International Parallel and Distributed Processing Symposium (IPDPS'07)*. IEEE, 2007.

[16] "Community Climate System Model Web Site," http://www.cesm.ucar.edu/models/ccsm4.0/.

[17] A. P. Craig, B. Kaufmann, R. Jacob, T. Bettge, J. Larson, E. Ong, C. Ding, and H. He, "CPL6: The new extensible high-performance parallel coupler for the Community Climate System Model," *Int. J. High Perf. Comp. App.*, vol. 19, no. 3, pp. 309–327, 2005.

[18] "The Message Passing Interface (MPI) standard," http://www-unix.mcs.anl.gov/mpi/.

[19] J. C. Meza, "Steepest descent," in *Wiley Interdisciplinay Reviews: Computational Statistics*, vol. 2, no. 6, 2010, pp. 719–722.

[20] D. B. Carl, "A practical guide to splines (revised edition)." Springer, 2001.

[21] 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, and P. Worley, "Benchmarking a parallel coupled model," *poster presented at Supercomputing '03*, 2003.

[22] J. W. Larson, "Visualizing process composition and load balance in parallel coupled models." *Procedia CS*, vol. 4, pp. 831–840, 2011.