2012 10th IEEE International Symposium on Parallel and Distributed Processing with Applications

Dynamic Load Balancing for Malleable Model Coupling

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*—Dynamic load balancing both within and between constituent subsystems is required to achieve ultrascalability in coupled multiphysics and multiscale models. Interconstituent dynamic load balancing requires runtime resizing—or *malleability*—of subsystem processing element (PE) cohorts. In our previous work, we developed and introduced the Malleable Model Coupling Toolkit with a load balance manager implementing and providing a runtime load-balancing algorithm using PE reallocation across subsystems. In this paper, we extend that work by adding the ability to adapt to coupled models that have changing loads during execution. We evaluate the algorithm through a synthetic coupled-model benchmark that uses the LogP performance model as applied to parallel LU decomposition.**

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

I. Introduction

Modern science and engineering challenges often involve complex, multidisciplinary, *coupled* systems. These challenges are often investigated by simulating the coupled systems with an application that contains interacting subsystem models, or *constituents*. These interactions lead to data dependencies between the constituents; 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].

Resource allocation is a fundamental issue when constructing scalable coupled simulations. Each constituent must be mapped to a collection of processing elements (PEs), called *cohorts*. Cohorts (under parallel composition, where the constituent PE cohorts do not overlap) should be sized based on the scalability and computational intensity of the corresponding constituent. Such load balancing is further complicated by its interconstituent data traffic and the wall-clock time it consumes. In order to reduce the wait time caused by load imbalance and data-dependency-driven interconstituent synchronization, resources (in our case PEs) should be allocated in a harmonious manner. Generally, such allocations are determined by trial and error, consuming much time and effort. Furthermore, often these resource allocations are static, even though the runtime load per constituent may change during the run. Moreover, load-balance configurations are platform-dependent. Thus, users of parallel coupled models would benefit significantly from infrastructure that supports automatic, interconstituent load balance at runtime. This need will become more pronounced with trends toward more complex coupled models (with increasing numbers of constituents) and exascale computer hardware (with increasing numbers of PEs).

The ability to reallocate constituent PE cohorts is termed *malleability* [4]. A coupled model that allows dynamic interconstituent load balance is a *malleable coupled model* (MCM). In previous work [5], [6], we introduced the Malleable Model Coupling Toolkit (MMCT), which is an extension of the Model Coupling Tooklit (MCT; [7], [8]) that supports runtime load balancing in parallel coupled models with a load balance manager providing a runtime load-balancing algorithm. Although our previous work performed load balancing at runtime, dynamic load conditions were not addressed; the object of our previous work was system implementation and application to performance tuning to find optimal static load balance solutions. Dynamic load conditions might result as a system's state evolves to one requiring more or less computation to simulate, for example the well-known sensitivity of computational load in climate models to instantaneous simulated weather state [9]. In this paper, we extend our system to support the ability to load-balance not only at runtime but under changing load conditions. We implemented a synthetic load based on the $\log P$ performance model of parallel LU decomposition [10], consisting of computation and communication costs, to simulate realistic throughput of a common parallel application, and used it for a benchmark. The benchmark adopts the typical structure and intercommunication pattern across constituents of a coupled climate model. We show that our work performs on under this synthetic coupled-model benchmark.

We introduce malleable model coupling and MMCT in Section II. The dynamic load-balancing algorithm and its performance for studies using the synthetic benchmark are presented in Sections III and IV, respectively. We describe related work in Section V and present our conclusions and outline future work in Section VI.

II. Malleable Model Coupling and MMCT

In a coupled model, each of its N constituents evolves as it solves its equations for each time step, communicating flux and state data as needed during coupling events [3]. These events can be irregular, driven by various thresholds,

978-0-7695-4701-5/12 $26.00 © 2012 IEEE
DOI 10.1109/ISPA.2012.28

or regular, according to some schedule. A coupled system whose coupling events are scheduled according to a repeated sequence within a constant time interval ΔT has a coupling cycle with period ΔT [3]; ΔT can be considered 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) [11], [12] 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]. The value τ_i can be decomposed as $\tau_i = \tau_i^{comp} + \tau_i^{coup}$, the sum of its constituent computation (including intraconstituent communication) and (non-overlapped) constituent coupling wall-clock times, respectively. The object of dynamic, interconstituent load-balancing is minimization of $\tau_G = \max\{\tau_i, \ldots, \tau_N\}$, through constituent PE pool reallocation—malleability—to harmonize the values of $(\tau_i^{comp}, \tau_i^{coup}), i = 1, \ldots, N$.

MMCT provides infrastructure for interconstituent PE reallocation and global PE cohort (i.e., MPI_COMM_WORLD) resizing. MMCT extends MCT with a centralized load-balance manager (LBM) and a dynamic process and communicator management system (PCMS). At startup, the head node of each constituent is created by mpiexec [13], 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 $(\tau_i^{comp}, \tau_i^{coup}), i = 1, \ldots, N$—to make PE cohort reallocation decisions; its analyses are guided by a load balance algorithm. The head node or process of each constituent measures timings and provides throughput information to the LBM. 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 an integral multiple of 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. Further details on a runtime architecture, including PCMS, LBM, and the software implementation of MMCT, and its relationship to MCT can be found in [5], [6].

III. Predictive Load-Balancing Algorithm

A constituent's computation time is assumed to be a function of p_i, the number of PEs in its cohort: $\tau_i^{comp} = f_i(p_i)$; and it 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. When $p_i > p_i^*$, overheads such as communication cost begin to overwhelm any benefit from additional PEs. Our algorithm starts from some initial allocation $\vec{P}^0 = (p_1^0, \ldots, p_N^0)$ to the system's N constituents. At every jth LBI, the LBM determines a PE cohort reallocation $(p_1^j, \ldots, p_N^j) \rightarrow (p_1^{j+1}, \ldots, p_N^{j+1})$ by analyzing measurements of τ_G and $\{\tau_1^{comp}, \ldots, \tau_N^{comp}\}$. Reallocation is guided by prediction of τ_G and proceeds in two phases: predict τ_i^{comp} (Section III-A), and use τ_i^{comp} to predict τ_G (Section III-B). Load balance decisions are made in a subsequent *optimization* phase (Section III-C) using predicted values of τ_i^{comp} and τ_G to determine better PE cohort allocations.

A. Prediction of Constituent Computation Time

We did not need to modify the τ_i^{comp} prediction scheme from our previous work to handle dynamic constitutent loads; rather, we use recent—as opposed to all—timing history to inform τ_i^{comp} prediction. The τ_i^{comp} are estimated by using a 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 ($\tau_i^{(comp,j)}$) as the ordinate (abscissa). Constituent computation time analysis is performed for each constituent in its respective two-dimensional (p_i, τ_i^{comp}) space. After the first LBI, the algorithm has only one measurement $(p_i^0, \tau_i^{(comp,0)})$. We shoot linearly from this point with slope $\delta\tau_i^{comp} = -\tau_i^{(comp,0)}/p_i^0$. This initial step is used for both the linear and the 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 operates on these data as follows. If there is more than one timing measurement, 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; otherwise we use the 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 using piecewise linear interpolation over the whole domain.

The cubic spline interpolation algorithm operates as follows. For the initial LBI, it uses linear shooting. For subsequent LBIs multiple timing observations are used, and the algorithm interpolates over the entire domain using a global method [14]. Pseudocodes for the piecewise linear interpolation and cubic spline interpolation algorithms are presented in [6].

B. Prediction of Global Iteration Time

Values of τ_G are affected by the constituents' PE allocations 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

```
{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 = τ_G;
    Collect or Update τ_i^comp and τ_G with current PE configuration;
    Perform linear or cubic spline interpolation for measurements of τ_i^comp
    if cur_iter_time ≤ 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. 1. OPTIMIZATION algorithm. Measurements of τ_i^{comp} are interpolated using the piecewise linear algorithm or cubic spline algorithm

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 N_E estimators of τ_G, each a weighted sum of τ_i^{comp}; here N_E is a complex, application-specific function of N and the number of interconstituent couplings present. The forecast for τ_G is

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

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.

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 $N_E = 2$ estimators and is

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

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

```
{max_models is an array of models whose τ_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 ≤ 0 then
        recipient_cand = max_models[i];
        donor_cands = all other models except recipient_cand;
        donor_recipient_cands = all possible reallocating directions with PEs
        ≤ 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 ≥ 0 then
        donor_cand = max_models[i];
        recipient_cands = all other models except donor_cand;
        donor_recipient_cands = all possible reallocating directions with PEs
        ≤ 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. 2. SELECTION algorithm using prediction information. It is used by the OPTIMIZATION algorithm.

```
Predict all τ_G of target PE allocations in donor_recipient_cands;
Sort donor_recipient_cands in increasing τ_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].τ_G ≤ 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. 3. DECIDE algorithm used by the SELECTION algorithm.

C. Optimization

The OPTIMIZATION algorithm (Figure 1) determines PE cohort reallocations to reduce τ_G, and this aspect also follows our previous work. 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 $\Delta\vec{P}^{j+1} = \vec{P}^{j+1} - \vec{P}^j$ 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 LBM's 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 2).

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 the *prediction counter* $\widehat{\Delta P}$. The prediction counter limits the search radius in PE space; that is, $|\Delta \vec{P}^{j+1}| \leq \widehat{\Delta P}$. Initially, we set $\widehat{\Delta P} = 2^2$, and it is always increased or decreased geometrically by a factor of 2, but restricted to the range $2^0 \leq \widehat{\Delta P} \leq 2^4$. 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 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.

The DECIDE algorithm (Figure 3) picks the PE space reallocation direction. It first calculates τ_G for all possible candidate PE reallocation vectors, sorting them 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. To address this situation we implemented a handler that strategically of deletes timing and prediction history and cancels a scheduled reallocation when load changes are detected. Load changes are detected by searching the LBM's history for each constituent to find previous τ_i^{comp} values corresponding to its current value of P_i, and comparing these historical values with the most recent value of τ_i^{comp}. If the current value of τ_i^{comp} differs from its historical values (for the current value of P_i) by more the value of an empirically determined threshold, this is considered to be a load change, and the LBM's timing database is altered to remove past, stale values of (P_i, τ_i^{comp}).

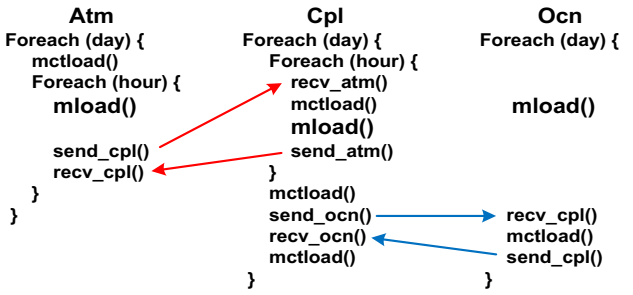

Fig. 4. Synthetic climate benchmark application using mload(). The arrows indicate the intercommunication pattern between constituents.

IV. Performance Studies

In our previous work, we used measured performance data from CCSM. Here, we extend that work by using an enhanced performance model based on a synthetic coupled model, adopting the simplified structure and intercommunication pattern of a parallel, coupled climate model introduced in [5].

The synthetic benchmark depicted in Figure 4 comprises atmosphere (atm), ocean (ocn), and coupler (cpl) constituents; these exchange interfacial flux and state data. To simulate a plausible τ_i^{comp} values for the constituents, we implemented modeled load (mload()) functions, which use the unix sleep() function to wait a period of time corresponding to τ_i^{comp}. The sleep time is computed by using the computational complexity of a of parallel LU decomposition [10], combined with the well-known LogP machine model [15]. This approach allows us to vary mload()—and with it τ_i^{comp} values—by varying the values of the following parameters: the upper bound on communications latency L; the communications overhead o; the gap between consecutive communications g; the number of PEs P; and the number of rows of the problem matrix N. A cyclic (block) data layout assumption was used to model load for cpl (atm and ocn). Modeled LogP-based costs are converted to wall-clock time by a scaling factor chosen to render the τ_i^{comp} cost curves into shapes and value ranges similar to those for benchmark measurements for CCSM's cpl, atm, and ocn constituents.

Two different parametric tunings were used in our LogP-based model, resulting in the load functions mload1() and mload2(), which are shown, respectively, in the left and right panels of Figure 5. The settings for mload1() are $\mathsf{L} = 30$ and $\mathsf{o} = 2$ for all constituents, and $\mathsf{g} = \{16, 8, 4\}$ and $\mathsf{N} = \{100, 120, 200\}$ for $\{\mathsf{cpl}, \mathsf{atm}, \mathsf{ocn}\}$. The settings for mload2() are $\mathsf{L} = 4$ and $\mathsf{o} = 2$ for all constituents, and $\mathsf{g} = \{8, 8, 4\}$ and $\mathsf{N} = \{100, 160, 120\}$ for $\{\mathsf{cpl}, \mathsf{atm}, \mathsf{ocn}\}$, respectively. Note the uniformity with respect to constituent in the values of the hardware-specific parameters L and o in mload1() and mload2(); this is because the target platform is a homogeneous commodity cluster. The values of the parame-

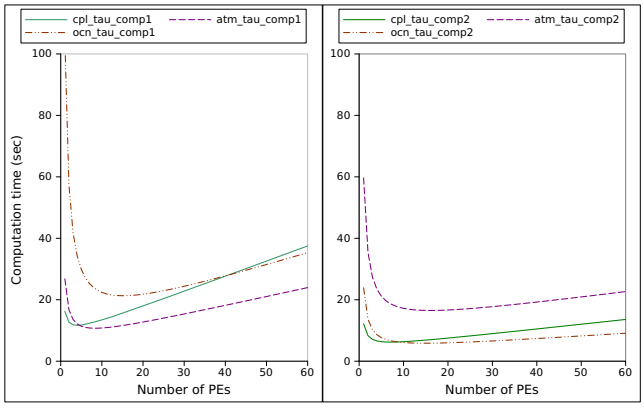

Fig. 5. Computation time of constituents τ_i^{comp} simulated by mload1() (left) and mload2() (right) based on the performance model with varying number of PEs.

ters N and g are functions of problem size and communications pattern, and thus constituent-specific.

Simulated τ_i^{comp} of constituents based on mload1() with varying number of PEs is shown on the left plot in Figure 5. Here, ocn is the slowest constituent and the simulated τ_i^{comp} for all constituents slopes downward initially but upward at some point because of the modeled communication cost that dominates τ_i^{comp} as the number of PEs increases. Values of τ_i^{comp} for each constituent, computed by using mload2(), are depicted in the right-hand plot of Figure 5. Note that mload2() makes atm the slowest constituent and the timing curves agree with our assumed constituent performance model.

In addition to mload(), constituents call mctload(), send(), and recv() in each time step. The mctload() function performs MCT distributed data transformation. Interconstituent communications between atm and cpl and between ocn and cpl are performed through the MCT send() and receive() communication calls. This pattern allows ocn to run concurrently while atm performs its computation and communicates with cpl in a nested time-loop representing 24 hours [3].

Performance studies were carried out under three scenarios. In Scenario 1 and Scenario 2, the benchmark ran with mload1(), but with differing processor set sizes $P^0 = 32$ and $P^0 = 64$, respectively. In Scenario 3, we evaluated the LBM's ability to cope with dynamically changing loads by switching between load functions mload1() and mload2().

In each study we evaluated the LBM optimizer, predicting $\tau_i^{(comp,j+1)}$ 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 [16] with ocn on its own cohort and (atm, cpl) composed sequentially on a shared cohort:

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

Reallocation requires three coupling cycles to complete because of synchronization between constituents and the LBM: one cycle waiting for the beginning of a timing cycle, another for timing, and a third waiting for reallocation instructions from the LBM. The new allocation can then be used for the next cycle.

Experiments were performed on a 64-node 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 MPICH2 implementation version 1.2.1 of the MPI-2 standard 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.

Average experimental results obtained by running the benchmark six times in three scenarios are summarized in Table I. Included are: initial values of τ_G with three sets of initial configurations P^0, final values of τ_G obtained by SEL1 and SEL2 load-balancing schemes, the number of coupling cycles required to find a PE configuration with the number of reallocations (#REALLOC) and the percentage of reallocations that were undone (UNDO(%)).

Performance analyses and studies of the three scenarios are described in following subsections.

Note that we define a coupled model PE allocation as $\vec{P} = (N_{\rm cpl}, N_{\rm atm}, N_{\rm ocn})$ and used it to describe PE allocation or configuration of the benchmark.

A. Scenario 1: Using mload1() with 32 PEs

We calculated the ideal solution of $\tau_G = 22.4$ s and PE configuration $\vec{P} = (4, 8, 20)$ based on the performance model with the specified parameter values for mload1() with 32 PEs.

For Scenario 1, we set the size of the global PE pool $P^0 = 32$ in order to maximize the benchmark's throughput. We then ran the benchmark with mload1() from three initial PE configurations: INIT1, with $\vec{P}^0 = (11, 11, 10)$, INIT2, with $\vec{P}^0 = (4, 24, 4)$, and INIT3, with $\vec{P}^0 = (4, 4, 24)$. 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.

For all initial cases with SEL1 and SEL2, final solutions for τ_G in Table I were almost identical and fairly similar to the ideal value of $\tau_G = 22.4$ s that was discovered. We emphasize that our benchmark never obtains the ideal value of τ_G because it uses mctload() in addition to mload1().

SEL1, however, found a PE allocation in 35% fewer coupling cycles than did SEL2 on average for all initial cases, since the curve of the simulated τ_i^{comp} of constituents plotted with the number of PEs in Figure 5 is closer to linear except the area around the inflection point. Moreover, the cubic spline interpolation cannot be guaranteed to fit properly to the area around the inflection point where the curve begin to slope upward. Moreover, the error value that is a side effect of misprediction with SEL2 should be bigger than that from SEL1 since the curve created by cubic spline interpolation is more widely varying and flexible than the one by piecewise linear interpolation. Thus SEL1, with mload1() in Scenario 1, which uses piecewise linear interpolation, made more accurate predictions than did SEL2 thus preventing unnecessary reallocations.

The convergence behavior of $\vec{P}$ and τ_G from INIT1 through

TABLE I
SUMMARY OF GLOBAL ITERATION TIME STATISTICS (SECONDS) AND LBM CONVERGENCE PROPERTIES STATISTICS.

	Scenario 1 with 32 PEs and mload1						Scenario 2 with 64 PEs and mload2						Scenario 3 with 32 PEs and both mload1 and mload2					
	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	25.1	25.2	34.4	34.4	24.7	24.6	32.6	32.3	33	33	28	28.1	25.2	25.2	34.4	34.4	24.8	24.8
Final τ_G	23.21	23.27	23.55	23.41	23.17	23.26	26.36	26.37	26.36	26.37	26.74	26.99	23.7	24.09	23.58	23.51	23.45	23.87
Found at	34.7	58.7	45.3	74.7	49.3	64.7	36.7	43.3	60.7	78.7	34	34	136.7	128.7	145	143.3	148.7	142.7
#Realloc	8.7	14.7	11.3	18.7	12.3	16.2	9.2	10.8	15.2	19.7	8.5	8.5	28.7	28.7	34.2	32.7	33	34.5
Undo(%)	31	34	28.3	31	36.6	32.1	23.9	25	27.6	29.4	29.4	35.3	31.4	31.4	31.3	31.2	31.8	29.6

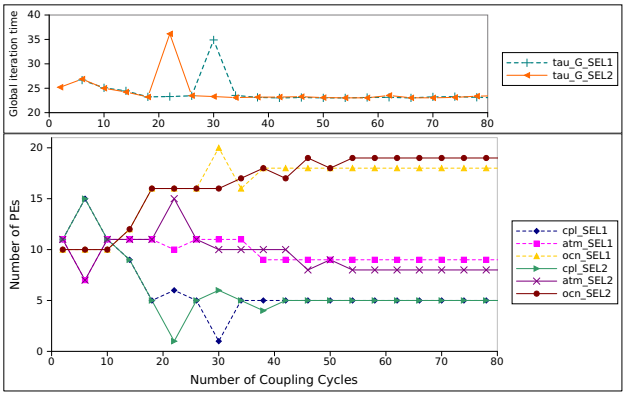

Fig. 6. Scenario 1: Optimization using mload1() with SEL1 and SEL2 for INIT1 case with 32 PEs: SEL1 found (5, 9, 18) with the global iteration time $\tau_G = 23.07$ s at the 36th coupling cycle, and SEL2 found (5, 8, 19) with $\tau_G = 23.04$ s at the 52nd coupling cycle.

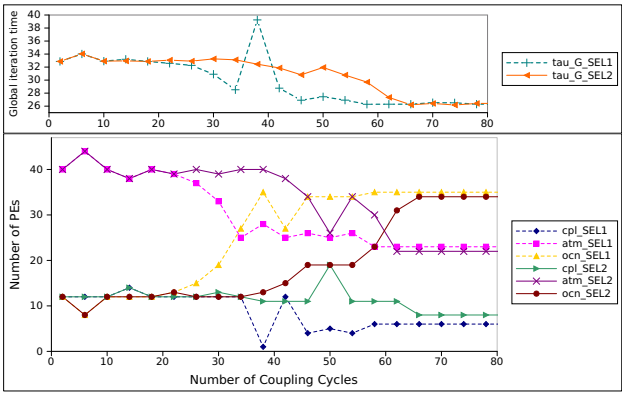

Fig. 7. Scenario 2: Optimization using mload1() with SEL1 and SEL2 for INIT2 case with 64 PEs: SEL1 found (6, 23, 35) with $\tau_G = 26.33$ s at the 56th coupling cycle, and SEL2 found (8, 22, 34) with $\tau_G = 26.33$ s at the 64th coupling cycle.

the optimization algorithm using SEL1 and SEL2 is shown in Figure 6. Both SEL1 and SEL2 forced cpl to donate PEs to ocn because both predicted that τ_G can be reduced if the throughput of ocn is improved by expanding its PE cohort's size. The algorithm using SEL1 forecasted accurately and led to reallocations with a proper number of PEs. After two serious mispredictions that introduced unnecessary reallocations, which were undone subsequently, at the 4th and 30th coupling cycles, the optimization converged at the 36th coupling cycle through 9 allocations with the solution $\tau_G = 23.07$ s. By contrast, although SEL2 found an almost equivalent ideal solution $(\vec{P}, \tau_G)$, the prediction by SEL2 was relatively inaccurate, introducing four more reallocations between cpl and ocn as well as between atm and ocn than did SEL1.

B. Scenario 2: Using mload1() with 64 PEs

For Scenario 2, we ran the benchmark using mload1() with a total of 64 PEs, more than optimal, in comparison with the amount of modeled load simulated by mload1(), from three initial configurations: INIT1, with $\vec{P}^0 = (11, 11, 10)$, INIT2, with $\vec{P}^0 = (4, 24, 4)$, and INIT3, with $\vec{P}^0 = (4, 4, 24)$.

We also computed the ideal solution with 64 PEs for τ_G and PE allocation through the performance model, resulting in 25.76 s with $\vec{P} = (6, 24, 34)$ to be used for reference.

Final values of τ_G obtained by the optimization with SEL1 and SEL2 for all initial cases were almost identical and close to the ideal value of τ_G. Furthermore, the optimization with SEL2 converged in 16% more coupling cycles on average than the one with SEL1; in comparison, SEL2 required 35% more coupling cycles than did SEL1 to converge the optimization in Scenario 1 because the τ_i^{comp} of the constituents in Scenario 2 were more stable relative to Scenario 1 while load balancing through reallocation since the oversupplied resource of 64 PEs allowed two constituents generally to be allocated more PEs than when the τ_i^{comp} of the constituents begin to go up according to the modeled τ_i^{comp} curve of constituents simulated by mload1() in Figure 5.

Figure 7 shows the optimization convergence behavior for SEL1 and SEL2 with INIT2 configuration in Scenario 2. As the figure shows, atm kept donating redundant PEs to ocn with both SEL1 and SEL2 in common until ocn's computation time was reduced and balanced with the sum of computation times of cpl and ocn according to the estimation scheme defined in (2). SEL1 tried to perform reallocations with a more appropriate number of PEs than did SEL2 with accurate prediction, even though 4 reallocations (including the one introduced a noticeable change of τ_G at 36th coupling cycle) were undone subsequently. By contrast, SEL2 performed reallocations conservatively and defensively, with fewer PEs than did SEL1, because of relatively inaccurate predictions. As a result, SEL2 required 10 more coupling cycles than did SEL1 to finish the optimization.

Experiments with mload() providing the realistic τ_i^{comp} with respect to the number of PEs reveal and emphasize the necessity of load balancing through expanding or shrinking the global PE pool of coupled model. For example, the ideal

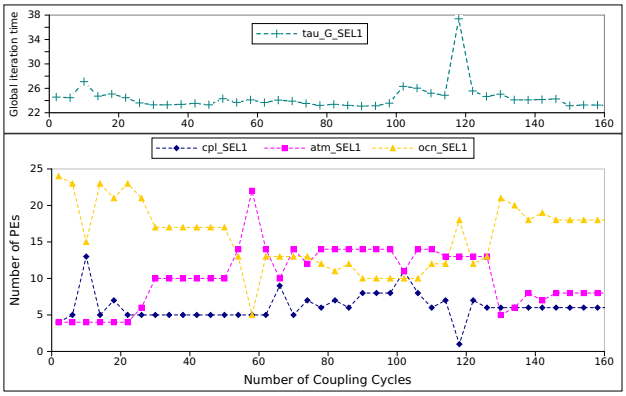

Fig. 8. Scenario 3: Optimization using mload1() and mload2() with SEL1 for INIT3 case with 32 PEs: SEL1 found (6, 8, 18) with $\tau_G = 23.22$ s at the 146th coupling cycle.

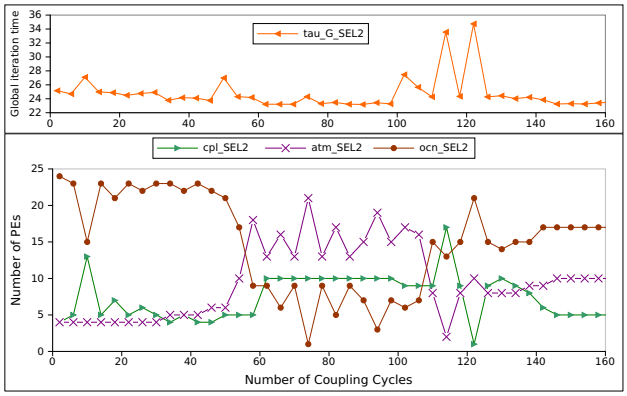

Fig. 9. Scenario 3: Optimization using mload1() and mload2() with SEL2 for INIT3 case with 32 PEs: SEL2 found (5, 10, 17) with $\tau_G = 23.55$ s at the 144th coupling cycle.

value of τ_G is 22.4 s with 32 PEs in Scenario 1. Not only an underprovisioned but also an overprovisioned resource can degrade the overall performance of coupled system.

C. Scenario 3: Using mload1() and mload2() with 32 PEs

To evaluate our algorithm with SEL1 and SEL2 when computation and communication costs of constituents are changed dynamically, we modified the benchmark slightly to enable it to switch between mload1() and mload2() during execution.

The benchmark employing mload1() initially began to run with 32 PEs from the three same initial PE configurations as those in Scenario 1. Then, the benchmark switched from mload1() to mload2() at the 50th coupling cycle and switched back from mload2() to mload1() at the 100th coupling cycle. Moreover, we set the threshold value of 1.5s to detect the load changes so that if the τ_i^{comp} of any constituent was increased or decreased by more than the value of the threshold, the handler described in Section III was invoked by the LBM.

The optimization convergence behaviors with SEL1 and SEL2 from INIT3 to handle a dynamic changeable load of constituents are presented in Figures 8 and 9, respectively.

Both SEL1 and SEL2 kept forcing ocn to hand over PEs to atm, since ocn is the slowest model with mload1(), until the computation time of constituents was changed by switching mload1() to mload2() at the 50th coupling cycle. We also observe that τ_i^{comp} changes to the value of τ_G from 23.3 s to 24.3 s and from 23.8 s to 27 s with SEL1 and SEL2, respectively. Once the load was changed, both SEL1 and SEL2 took PEs from ocn and donated them to atm or cpl to balance the τ_i^{comp} of the constituents following the estimation scheme, since atm became the slowest model. In contrast to the convergence behavior with SEL1, SEL2 failed to find a PE configuration that balanced the τ_i^{comp} of the constituents for mload2() until mload1() was employed again at the 100th coupling cycle instead of mload2() for the same reasons presented in Section IV-A. By contrast, SEL1 found the solution $(\tau_G, \vec{P}) = (23.1 \text{ s}, (8, 14, 10))$ at the 88th coupling cycle for mload2(). However, the algorithms with SEL1 and SEL2 converged successfully at the 146th and the 144th coupling cycles with good solutions of $\tau_G = 23.22$ s and $\tau_G = 23.55$ s, respectively.

All final τ_G values obtained by optimization with SEL1 and SEL2 with the three initial PE configurations were similar to the ideal value of $\tau_G = 22.4$ s with 32 PEs; moreover, we verified that the algorithm converged with a similar number of coupling cycles to that presented in Scenario 1, once mload() was switched back at the 100th coupling cycle. These results show that our dynamic load-balancing algorithm for malleable model coupling can support a type of coupled model whose throughput is changed at runtime as a result of dynamic computation and communication costs of constituents.

We also detected an issue with our algorithm during experiments in Scenario 3. It could disrupt the optimization using the derivative $\delta\tau_i^{comp} = -\tau_i^{comp}/p_i^0$ as the initial step for τ_i^{comp} prediction because the optimization could converge based on an unreliable guideline produced by only one timing measurement. However, this issue could be addressed simply by assigning 0 to the slope of the linear interpolation to predict the τ_i^{comp} of the constituents that are not the slowest if only one timing measurement exists.

V. Related Work

Malleability has previously been applied to parallel iterative applications for dynamic load-balancing. SRS [17] adjusts PE cohorts by allowing a parallel application to stop and restart its execution. PCM/IOS [18], [19] profiles parallel applications and triggers reconfigurations to support malleable, iterative MPI applications. The ReSHAPE [20] framework changes PE allocations of malleable parallel applications during job scheduling for system resource utilization. Utera et al. [21] also utilize 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.

Ko et al. [22] presented a coupled multiphysics simu-

lation system that optimizes the PE allocation. The algorithm, however, supports only coupled models with two constituents and assumes that computation time scales ideally with parallelization. The CSCAPES project [23] has as one of its foci dynamic load-balancing for parallel applications. Hypergraph-based repartitioning with Zoltan [24], 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.

VI. Conclusions and Future Work

Model coupling for complex scientific multiphysics and multiscale systems is an emerging research challenge. We are developing the Malleable Model Coupling Toolkit, which provides a dynamic load-balancing scheme by reallocating resources, specifically the PEs of a coupled model's constituents' cohorts.

In this paper, we have shown that by using a heuristic to detect load changes, our algorithm can adapt to changing loads of coupled models at runtime.

We also have presented enhanced performance studies of our dynamic load-balancing scheme for malleable model coupling using a synthetic coupled model benchmark that was run with a modeled load of parallel LU decomposition simulating realistic computation and communication costs of coupled model subsystems. The experiments highlighted the need for a load balancing algorithm to guide how to shrink and expand the size of the global PE cohort. Future work will explore such an algorithm. In addition, our experiments confirmed that the optimization algorithm balances loads of communication-intensive constituents adequately.

We will continue to improve our dynamic load-balancing algorithm by reducing the convergence time and improving the accuracy of prediction. As an ultimate goal, we hope to apply MMCT with LBM to deployed applications.

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," *Procedea Computer Science*, vol. 4, pp. 312–321, 2011.

[6] ——, "Malleable model coupling with prediction," Preprint ANL/MCS-P1984-1211, 2011. [Online]. Available: http://www.mcs.anl.gov/publications/

[7] 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.

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

[9] J. G. Michalakes, "Analysis of workload and load balancing issues in the NCAR community climate model," Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, Technical Memorandum ANL/MCS-TM-144, 1991.

[10] E. E. Santos and M. Muraleetharan, "Analysis and implementation of parallel lu-decomposition with different data layouts," Preprint, 2010. [Online]. Available: http://math.ucr.edu/~muralee/

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

[12] 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.

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

[14] D. B. Carl, *A Practical Guide to Splines (Revised Edition)*. Springer, 2001.

[15] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: towards a realistic model of parallel computation," in *Proceedings of the fourth ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*. ACM, 1993, pp. 1–12.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

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