Salsa: Scalable Asynchronous Replica Exchange for Parallel Molecular Dynamics Applications

Li Zhang and Manish Parashar
TASSL, CAIP Center
Electrical and Computer Engineering Department
Rutgers University
94 Brett Road,
Piscataway, NJ 08854
{emmalily, parashar}@caipclassic.rutgers.edu

Emilio Gallicchio and Ronald M. Levy
BIOMAPS Institute for Quantitative Biology
Department of Chemistry and Chemical Biology
Rutgers University
Wright-Rieman Laboratories, 610 Taylor Rd,
Piscataway, NJ 08854
{emilio, ronlevy}@biomaps.rutgers.edu

Abstract— **This paper presents Salsa, a novel, decentralized and asynchronous realization of the "replica exchange" algorithm for simulating the structure, function, folding, and dynamics of proteins. Salsa provides a scalable communication and interaction substrate that presents a virtual shared space abstraction and enables the dynamic and asynchronous interactions required by the simulations to be simply and efficiently implemented. The design, implementation and experimental evaluation of Salsa and replica exchange-based simulations using Salsa are presented.**

I. Introduction

Replica exchange is a powerful sampling algorithm that preserves canonical distributions and allows for efficient crossing of high energy barriers that separate thermodynamically stable states. In this formulation, several copies, or replicas, of the system of interest are simulated in parallel at different temperatures using "walkers". These walkers occasionally swap temperatures to allow them to bypass enthalpic barriers by moving to a higher temperature. The replica exchange algorithm has several advantages over formulations based on constant temperature, and has the potential for significantly impacting the fields of structural biology and drug design - specifically, the problems of structure based drug design and the study of the molecular basis of human diseases associated with protein misfolding, which are the applications currently targeted by this research.

However, efficient and scalable parallel implementations of general formulations of the replica exchange algorithm present significant challenges. These challenges are primarily due to the dynamic and complex coordination and communication patterns between the walkers. These communication/coordination patterns depend on state of the simulation at the replicas and are known only at runtime, and consequently, implementing these simulations using commonly used parallel programming frameworks is non-trivial. Message passing frameworks such as MPI [1] require matching sends and receives to be explicitly defined for each interaction. Programming frameworks based on shared address spaces provide higher-level abstractions that can support more dynamic interactions. However, scalable implementation of global shared address spaces remains a challenge.

As a result, to the best of our knowledge, all the current parallel/distributed implementations of replica exchange simulations in use by the structural biology community are based on a simplified formulation of the algorithm that limits the potential power of the technique in two important ways: (1) the only parameter exchanged between the replicas is the temperature of each replica, and (2) the exchanges occur in a centralized and totally synchronous manner, and only between replicas with adjacent temperatures. The former limits the effectiveness of the method, while the latter limits its scalability to at most tens of homogeneous and relatively tightly coupled processors.

Enabling larger scale implementations of the general replica exchange formulation thus requires a communication substrate that can support decentralized management of exchange schedules, and asynchronous and decoupled communications, eliminating synchronization overheads and allowing implementations to be scalable. The paper describes the design and implementation of Salsa, a communication and interaction substrate that meets these requirements, and enables a novel, decentralized and asynchronous realization of the replica exchange algorithm for simulating the structure, function, folding, and dynamics of proteins. Salsa enables arbitrary walkers to dynamically exchange target temperatures and other information in a pair-wise manner based on an exchange probability condition that ensures detailed balance.

The Salsa-based replica exchange realization distinguishes itself from existing implementations in multiple aspects. As mentioned above, existing replica exchange implementations use a simplification of the replica exchange algorithm where the walkers that can exchange temperatures is limited to those with neighboring target temperatures. In these implementations, the pairs of walkers that exchange temperatures is centrally determined at a single node by periodically gathering temperatures from all the walkers at this node. While such a scheme is acceptable for simulations with a small number of walkers, as the number of walkers increases, the possibility of successful non-nearest neighbor temperature exchange also

0-7695-2636-5/06 $20.00 © 2006 IEEE

increases significantly, and the temperature mixing is severely impeded when exchanges can only occur between neighboring temperatures.

Salsa essentially provides a virtual shared space abstraction that is specifically customized for replica exchange. The pairs of walkers that exchange information are dynamically and asynchronously determined using this virtual shared space abstraction. Walkers periodically post temperature ranges that are of current interest for exchange to the shared space. If this range overlaps with the range of interest posted by another walker, an exchange can occur. The actual exchange is then negotiated and completed by the individual walkers in a peer-to-peer manner. As a result, the exchanges are decoupled, dynamically and asynchronously determined, and not limited to neighboring temperatures. Salsa provides simple tuple-space-like [11] abstractions for accessing the virtual space. Walkers use the *post* operator to post temperature ranges of interest, and use either the blocking *get* or the non-blocking *getp* operator to retrieves a new temperature if the attempted exchange is successful or the old one if the attempted exchange is unsuccessful. Further, since exchanges are decoupled and asynchronous, communications and computation at the walkers can be overlapped to improve overall simulation performance.

The design and implementation of Salsa is based on the following observations about the replica exchange algorithm: (1) the overall temperature range of the simulation and the set of temperatures that are assigned to walkers are determined at the beginning of the simulation and are known to all the walkers; (2) the temperature assigned to a walker only changes when the walker performs an exchange; and (3) exchanges occur between pairs of nodes. The first two observations allows individual walkers to locally determine temperature ranges of interest and exchange decisions to be made in a decentralized and decoupled manner. The third observation allows actual exchanges to occur between pairs of walkers in parallel. Salsa consists of two main components: a directory layer that is implemented as a distributed hash table (DHT) where walkers can post temperature ranges of exchange interest and discover potential exchange partners in a decentralized and asynchronous manner; and a communication layer that manages the actual exchange of data in an efficient and peer-to-peer manner.

Salsa has been implemented within the IMPACT (Integrated Modeling Program, Applied Chemical Theory) molecular mechanics program [2] to enable two specific applications, which require large scale distributed replica exchange implementations: (1) simulations of the binding of ligands to the cytochrome P450 class of enzymes responsible for cellular detoxification and drug metabolism, and (2) simulations of the misfolding of naturally occurring human and mutated forms of the protein synuclein associated with Parkinson's disease. It is not possible to carry out these studies using standard molecular dynamics simulation techniques.

The rest of the paper is organized as follows. Section 2 describes the problem domain and gives an overview of the replica exchange algorithm. The section also describes current MPI-based implementations of the method. Section 3 describes the design of Salsa and presents the implementation of the Salsa-based decoupled and asynchronous replica exchange algorithm. Section 4 describes the implementation and the experimental evaluation of Salsa-based simulation. Section 5 reviews related work. Section 6 concludes the paper and outlines future research directions.

II. Parallel Replica Exchange for Structural Biology and Drug Design

The sequencing of the human genome, in conjunction with rapidly increasing efforts in structural genomics, is producing an explosion in the number of available high resolution protein structures. Molecular simulations of protein structural changes and drug binding to proteins depend critically on the design of highly efficient algorithms to search over the very rough energy landscapes which govern protein folding and binding. Scalable parallel replica exchange implementations can potentially address these molecular search problems and can significantly impact structure based drug design applications.

A. The Replica Exchange Algorithm

Replica exchange is an advanced canonical conformational sampling algorithm designed to help overcome the sampling problem encountered in biomolecular simulations. The method had been proposed independently on several occasions in various disciplines [4], [5], [6], [7]. In this method, several copies, or replicas, of the system of interest are simulated in parallel at different temperatures using walkers. These walkers occasionally swap temperatures based on a proposal probability that maintains detailed balance [3]. These exchanges allow individual replicas to bypass enthalpic barriers by moving to high temperatures. A parallel version of this algorithm was proposed by Hukushima & Nemoto [7]. The replica exchange algorithm is easy to implement and does not require time-consuming preparatory procedures. Further, it can decrease the sampling time by factors of 20 or more, as compared to constant temperature molecular dynamics when applied to peptides at room temperature [8]. Details of the algorithm [3] as well as application examples [9], [10] are available elsewhere.

The MD replica exchange canonical sampling method has been implemented in IMPACT, the molecular simulation package used in this work, following the approach proposed by Sugita & Okamoto [9]. The method consists of running a series of simulations at fixed specified temperatures. Each replica corresponds to a temperature. An exchange of temperatures between replicas i and j at temperatures T_m and T_n is attempted periodically and is accepted according to the following Metropolis transition probability [9]:

$$W = \min\{1, \exp[-(\beta_m - \beta_n)(E_j - E_i)]\} \qquad (1)$$

where $\beta = 1/kT$ and E_i and E_j are the potential energies of replicas i and j, respectively. After a successful exchange, the velocities of replicas i and j are rescaled at the new temperature.

B. Existing Parallel Implementations of Replica Exchange-based Simulations

Molecular dynamics programs are essentially loops over a large number of integration steps, each of which advances the time forward for one step. Replica exchange is attempted periodically at a chosen interval of steps. As mentioned in the introduction, existing MPI-based parallel implementations of replica exchange are centralized and synchronous. For example, in the existing implementation in IMPACT, a central master node collects temperature data about all the replicas from the walker nodes, and then broadcasts the collected data array to the walkers. Each walker node receives this data array and sorts the array locally. Neighboring temperatures in the sorted array are potential partners for temperature exchange. The master node randomly selects between two modes of exchange. One is to exchange with upper neighboring temperature and the other is to exchange with lower neighboring temperature. The master notifies the walkers about the selected mode, and walkers can then mutually exchange temperatures based on this information. During the actual exchange, one of the two walker nodes with neighboring temperatures in the sorted array that are paired up for temperature exchange, acts as a temporary server. This walker collects temperature and potential energy data from the other node, determines whether the exchange is feasible based on the transition probability given in Eq. (1), and replies with either the new temperature, if the exchange is successful, or with a notice of denial otherwise.

The parallel replica exchange implementation described above has several limitations. First, the scheme limits the exchange to only neighboring temperatures. This limitation is not a concern when the number of replicas is small and there is a small chance of exchange between non-nearest temperatures. However, as the number of processors (and correspondingly walkers) increases, the difference between target temperatures becomes small enough to allow exchanges between non-nearest neighbor replicas. In such cases, more flexible schemes which allows non-nearest neighbor temperature exchange are desirable. Second, the implementation is based on a centralized master that gathers and scatters data system wide. Gathering data from all the nodes on a single node may be infeasible in large systems, and a centralized master can quickly become a bottleneck. Further, gather and scatter operations are synchronous and expensive. Also, since the master node also participates in the simulation as a walker, there is a load imbalance which can lead to additional synchronization overheads.

To overcome the above limitations, we propose a scalable decentralized and asynchronous realization of the replica exchange algorithm using the Salsa communication substrate, which is presented in the following section.

III. SALSA: A FRAMEWORK FOR PARALLEL ASYNCHRONOUS REPLICA EXCHANGE

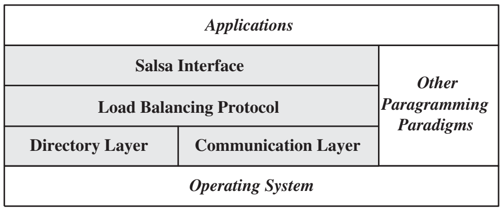


Fig. 1. An schematic of the Salsa architecture.

A. The Salsa Architecture

Salsa provides abstractions and underlying mechanisms to support efficient and scalable parallel implementations of the general replica exchange formulation, where walkers can exchange non-nearest neighbor temperatures in a decoupled, decentralized, and asynchronous manner. It essentially provides the abstraction of a virtual shared space that is specifically customized for replica exchange. The shared space supports tuple-space-like abstractions and is used by walkers to post temperature ranges of exchange interest and discover candidate walkers that it can potentially exchange temperature with. The underlying mechanisms support negotiation and efficient peer-to-peer data exchanges between appropriate walkers. A schematic of the Salsa architecture is presented in Figure 1. The framework consists of two main components: (1) a distributed directory layer; and (2) a communication layer. These components are described below.

1) Distributed Directory Layer: The distributed directory layer provides the Salsa virtual shared space abstraction and supports its associative access operators. It is implemented as a distributed hash table (DHT) where the index of the hash table is derived from the overall temperature range used by the simulation using a simple hashing function. This index is then dynamically partitioned and distributed across the participating nodes. Each node is thus responsible for its portion of the index and the corresponding temperature range, and essentially serves as the rendezvous point for exchange interest posting that intersects with its range. A Salsa service daemon running at each node is responsible for handling these exchange interest postings and storing them locally, and for detecting matches with existing postings of exchange interest at the node.

2) Communication Layer: The communication layer provides the mechanisms for negotiations and efficient data transfers. The negotiation mechanisms enable walkers to mutually agree to exchange data, while the data transfer mechanisms support low latency peer-to-peer data exchanges [21] between the walkers. Note that multiple data exchanges between different pairs of walkers can proceed in parallel.

B. Salsa Programming Interface

The operators provided by the Salsa application programming interface (API) are listed in Table I. These operations provide associative access to the virtual shared space and are similar to the operators provided by the tuple space model [11]. The interface includes operators to initialize the Salsa runtime

TABLE I

Salsa application programming interface.

Operation	Description
init(global-temperature-range)	Initialize the shared space.
post(exchange-temperature-lower-bound, exchange-temperature-upper-bound)	Post a temperature range of exchange interest to the space.
get(?temperature, energy)	Get the exchanged temperature from the space. This is a blocking call and the calling process blocks until a matching temperature is available. The retrieved temperature is removed from the space.
getp(?temperature, energy)	Get the exchanged temperature from the space. This is a non-blocking call and the calling process continues if no matching temperature is available. The retrieve temperature is removed from the space.

```
if (seineinitflag .eq. 0) then
      call init_salsa (global_temperature_lowbound,
                        global_temperature_upperbound)
      seineinitflag = 1
      timestamp = 0
else
      timestamp = timestamp + 1
endif

if (timestamp .eq. (timestamp/exchange_rate)*exchage_rate)
then
      call post (tempt(nspec+1) - GUESSRANGE,
                  tempt(nspec+1) + GUESSRANGE)
endif
call getp (newtemp, epot, accepted)
```

Fig. 2. Pseudo-code illustrating the implementation of replica exchange using Salsa.

(*init*), to allow processes to post (*post*) temperature range of exchange interest and retrieve available exchanged temperatures (*get*/*getp*). Note that the retrieved temperatures are removed from the space.

The replica exchange algorithm can be simply implemented using the Salsa API as illustrated by the psuedo-code presented in Figure 2. The MPI-based implementation tends to be significantly longer and more complex. Note that these Salsa operators can be easily extended to support "temperature plus potential parameter replica exchange" formulation that facilitates barrier crossing by "flattening" the energy barriers in addition to kinetically activating the crossing by heating the system.

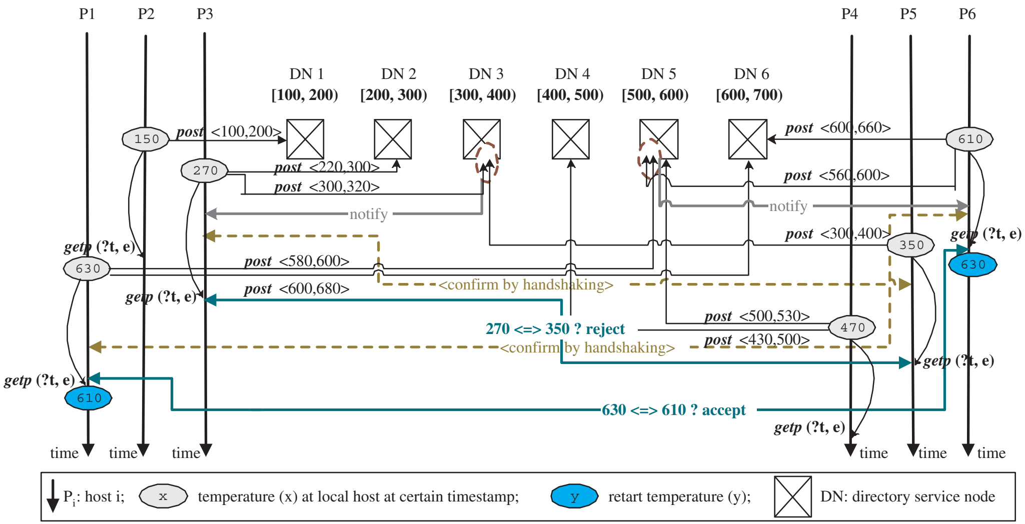
C. Parallel Asynchronous Replica Exchange using Salsa

The overall operation of Salsa is as follows. When a walker attempts to exchange its current target temperature, it computes a temperature range that it is willing to exchange with and posts this range to the shared space using the *post* operator. Based on the temperature range posted, the request is routed to all the service daemons whose index ranges overlap with the hashed posted range. Note that a posted temperature range may be unevenly distributed across the directory nodes resulting in load balancing issues. Currently Salsa addresses the issue using a simple load balancing protocol and plan to further improve the protocol. When a remote *post* request is received by a service daemon, the daemon first checks its storage for potential exchange partners. If a candidate exists (say $walker_2$), the requesting walker (say $walker_1$) is notified. Otherwise, the incoming request is stored.

Since a *post* request typically maps to multiple directory nodes and therefore is sent to multiple service daemons, it is possible that a requesting walker is notified of multiple candidates, if more than one daemons find a potential exchange partner. In this case, the first notification that reaches the requesting walker is accepted. However, the exchanging walkers must mutually agree to exchange data with each other. This requires a two-way handshake. In the example above, $walker_1$ will send out a query message to ask $walker_2$ whether it is available for an exchange. On receiving this query from $walker_1$, $walker_2$ checks it local state. This state can be "free", "onhold", or "finished". The walker is available for an exchange only if it is in the "free" state. The "onhold" state indicates that the walkers has already agreed to exchange with another walker but exchange has not yet occurred. The "finished" state indicates the walker has already finished an exchange with another walker and its posted interest to exchange is no longer valid. In the example, if $walker_2$ is in the "free" state it will respond positively to $walker_1$. At this point both walkers confirm their intent to exchange data with each other and change their state to "onhold". If $walker_2$ had responded negatively, $walker_1$ would have continued to negotiate with other walkers that it had been notified of until it finds a willing exchange partner or it has no more walkers to negotiate with. In the latter case, it just gives up and continues simulation with its current data until the next exchange cycle.

Once a pair of walkers agree to exchange data, they initiate the actual exchange by invoking the *get* or *getp* operator. The exchange is performed using the mechanisms provided by the communication layer. The exchange proceeds as follows. One of the walkers sends its current data (e.g. temperature and energy) to its potential partner. The potential partner determines whether they can exchange based on data it receives and its own data. This step is necessary since the exchange happens asynchronously and in parallel with the computation, and a walkers data (i.e., energy) may have changed since it posted its exchange interest. If the walker decides to continue with the exchange, it will notify its partner of the decision and sends its current local data to complete the exchange. Note that an exchange is between a pair of walkers and multiple exchanges between different pairs of walkers can proceed in parallel.

An implementation of a parallel asynchronous replica exchange using Salsa is illustrated in Figure 3. As described above and shown in the figure, the scheme consists of two



Primitives:
post (exchange-interest-range)
[description] post the interest range of temperature exchange to the shared temperature space.
getp (?temperature, energy)
[description] asynchronous retrieval of an exchange temperature. After posting a temperature interest range, the local processor continues computation using old temperature, meanwhile periodically checks the shared temperature space for available potential exchange partners. If a candidate appears, an exchange will be attempted in the getp operation.

Fig. 3. An example parallel asynchronous replica exchange implementation using Salsa.

phases. In the *post* phase, candidate exchange partners are identified and notified. These walkers negotiate with each other to create potential exchange partnerships. In the *get* or *getp* phase, these potential partners then attempt to exchange data.

In the Salsa-based replica exchange algorithm, a walker specifies the temperature range that it is interested in exchanging with as a parameter of the Salsa *post* operator. Usually, the larger the range, the higher is the probability of finding an exchange partner and will result in better solution quality. However, a larger range will also map to a large number of directory nodes and the *post* request will be forwarded to a large number of service daemons. This in turn increases communication overheads. The service daemons are also more loaded in this case, reducing their performance. Consequently, the temperature range posted must reflect the best tradeoff between solution quality and simulation performance.

IV. Salsa Implementation and Experimental Evaluation

A prototype of Salsa has been implemented using multi-threading and TCP sockets. When Salsa is initialized by the application, a Salsa thread is spawned at each node in the system, which co-exists in the application address space. This thread acts as a Salsa service daemon and handles post request. Salsa service daemons discover and coordinate with each other to construct the directory layer DHT structure using a bootstrap server.

The effectiveness and scalability of the Salsa-based replica exchange algorithm is evaluated using the alanine tripeptide molecule. The tests are conducted on up to 68 processors (due to availability) using two Beowulf cluster, with 64 processors and 8 processors respectively, which consists of Linux-based computers connected by 100 Mbps full-duplex switches. Each processor has an Intel(R) Pentium-4 1.70GHz CPU with 512MB RAM and runs Linux 2.4.20-8 (kernel version). All the tests are configured to run for 10 ns total simulation time using the Hybrid Monte Carlo (HMC) [22] molecular dynamics sampling algorithm. Each run is composed of 250,000 HMC cycles, each including 10 molecular dynamics integration steps with a 4 fs time-step. Replica exchange is attempted every 25 HMC cycles. Replica exchange temperatures are distributed exponentially within the 200-700 K range. The experiments compare the efficiency and performance of the Salsa-based implementation with the original MPI-based implementation in Impact. These experiments are described below.

A. Salsa-based vs. MPI-based Replica Exchange

An important feature of the Salsa-based replica exchange implementation is its ability to support non-nearest neighbor temperature exchanges. This feature is essential for ensuring proper mixing of temperatures across the walkers, especially when the simulation includes a large number of walkers. At

[1] When the system size is large, *post* requests are only sent to a subset of the service daemons that correspond to the requests to reduce communication overheads.

TABLE II

NUMBER OF TEMPERATURE CROSS-WALK EVENTS.

Number of walkers	8	16	32	60	120	136
Temperature range (in Kelvin) for measuring the number of cross-walks (write rate: per 250 steps)	250=> 650=> 250	250=> 650=> 250	250=> 650=> 250	250=> 650=> 250	250=> 650=> 250	250=> 650=> 250
Posted temperature range for replica exchange	Salsa-based simulation					
[-80K, 80K]	106	120	249	347	798 [1]	941 [1]
	MPI-based simulation					
	100	101	94	44	6	3

equilibrium each walker visits each temperature with equal probability. The rate of temperature equilibration is measured by the number of "cross-walks", whereby a walker originally within the low temperature range (200 K $\leq$ T $\leq$ 250 K) reaches the upper temperature range (650 K $\leq$ T $\leq$ 700 K) and then returns to the lower temperature range. As shown in Table II, the Salsa-based replica exchange implementation achieves a larger number of cross-walks as compared to the synchronous MPI-based replica exchange implementation. This is because the synchronous MPI implementation only supports nearest neighbor temperature exchanges, and the walkers have to travel through every temperature in order to complete a cross-walk in the temperature space. This diffusion process is particularly slow when the replica exchange simulation includes many temperatures. Since Salsa supports non-nearest neighbor temperature exchanges, a walker can "jump" through temperature space, resulting in a faster rate of temperature equilibration.

Table II shows the number of temperature cross-walks measured for Salsa-based replica exchange simulations with 8, 16, 32, 60, 120 and 136 walkers, compared with the corresponding number of cross-walks obtained using the MPI-based synchronous implementation. In these experiments, the temperature range posted by walkers in the Salsa-base replica exchange implementation was set to a window of 160 K around its target temperature, i.e., as [temp - 80 K, temp + 80 K]. In this configuration, a walker can exchange temperature with any other walker whose target temperature is less than 160K away, i.e., the ranges that the two walkers post intersect. The temperature ranges that defined a cross-walk in the experiments are listed in Table II. As seen from the results listed in the table, the number of observed temperature cross-walks is larger for the Salsa-based simulation, and it increases as the number of walkers increases. In contrast, the number of cross-walks observed for the MPI-based implementation decreases as the number of walkers increases. With 136 walkers there are only 3 temperature cross-walks observed in the case of the MPI-based implementation. In comparison 941 cross-walks are observed for the Salsa-based implementation. These results demonstrate that using non-nearest neighbors replica exchange algorithm enabled by Salsa provides significant benefits, especially when the density of the temperature distribution is large.

B. Scalability of Salsa

The decentralized design of Salsa enables it to scale well with increasing number of processors. This experiment measures the wall-clock execution time for the Salsa-based and MPI-based implementations of the replica exchange simulation with 8, 16, 32, 60, 120, and 136 walkers. The results are plotted in Figure 4. Note that for the simulation with the cases of 120 and 136 walkers, 2 walkers were mapped to each processor since the combined system used had only 68 processors available. The execution time for both implementations are appropriately scaled for this case. Also note that, since Salsa-based replica exchange executes in a decentralized asynchronous manner, different walkers or processes may proceed at different speeds and therefore have different execution times.

The measurements plotted in Figure 4 correspond to the average execution time across all the walkers/processes. A large portion of the execution time is due to local potential energy evaluations necessary to propagate each walker in time. This component of the execution time is the same for both implementations. The remaining portion of the execution time is due to exchange communications and includes both communication times and the synchronization overheads. This component of the execution time is proportional to the number of exchanges and is different for the two implementations.

Figure 4(a) plots the average run-times for the two implementations for different numbers of processors and correspondingly, walkers. This plot reflects a combination of factors. First, since in the MPI-based implementation, exchanges are centralized and synchronous, they have high probability to be successful. However, in the case of Salsa, exchanges are decentralized and asynchronous, and many initiated exchanges may not succeed. As a result, there is some "wasted" communication in Salsa. However since this communication occurs in parallel and is overlapped with computations its impact is not as significant. Second, the Salsa-based simulations result in a larger number of cross-walks and thus perform a significantly larger number of exchange communications. The combined result of these two factors results in higher execution times for the Salsa-based implementations for up to 60 walkers. For a larger number of walkers, the bottleneck caused by centralization and synchronization in the MPI-based implementation becomes significant as apparent for the 120 and 136 walker cases in the plot. We believe that the

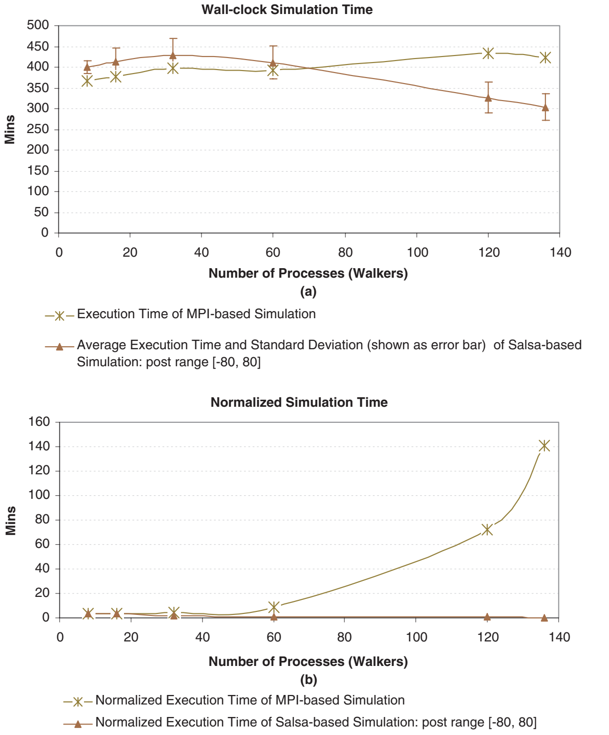


Fig. 4. (a) Average wall-clock execution time and standard deviation with increasing number of processes (walkers); (b) Normalized execution time with increasing number of processes (walkers).

impact of this bottleneck will be even more pronounced for larger systems and for distributed systems where gather/scatter operations are expensive.

It is also useful to consider the relative effective performance of the two implementations. As shown in Table II, the Salsa-based implementation produces more temperature cross-walks that in turn are reflected in a smaller convergence time, i.e., the time required to obtain a particular thermodynamic quantity of the molecular system within a given statistical uncertainty. To evaluate the effective gain in performance achieved using Salsa, Figure 4(b) plots the normalized execution time for both implementations, obtained by dividing the wall clock execution time by the number of cross-walks. This gives the average time required to achieve one temperature cross-walk. As this figure shows, for small number of processes the Salsa and MPI implementations have similar effective performance. However as the number of processes and correspondingly, the number of walkers increases, the performance of the Salsa-based simulation increases noticeably.

C. *Effect of Posted Temperature Ranges on Cross-Walks*

The temperature range posted by a walker ([-80 K, 80 K] in the experiment presented above) is an adjustable parameter that can be optimized for a specific system and number of walkers, to achieve the fastest convergence. To illustrate the effect of the posted temperature range on the number of temperature cross-walks, Figure 5 plots the number of cross-walks obtained using the ranges [-150 K, 150 K] and [-80 K, 80 K]. These results show that the [-80 K, 80 K] temperature range gives better results than the [-150 K, 150 K] range in the cases of 8, 16, 32, and 60 walkers while [-150 K, 150 K] gives better result in the case of 120 and 136 walkers.

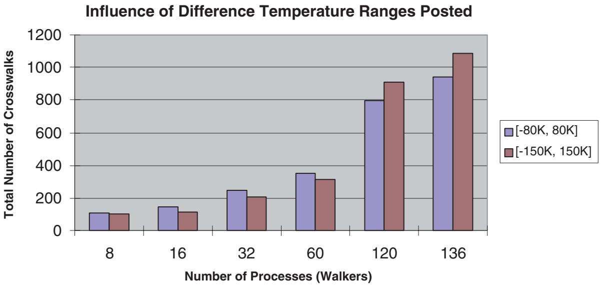


Fig. 5. Influence of different posted temperature ranges on the number of cross-walk events.

V. Related Work

A. *Parallel Replica Exchange*

Existing parallel implementations of replica exchange are based on a simplified formulation of the replica exchange algorithm as described previously, and use a centralized master to periodically schedule and manage exchanges. These implementations are either directly build on sockets, as in [3], or use message passing libraries such as MPI or PVM, as in [13]. In [13], H. J. C. Berendsen et al present a parallel replica exchange implementation developed specially for a ring topology. The implementation is suitable for systems where the processors can be configured as a (logical) ring and which support blocking send and receive calls. In this implementation, each processor only communicates with its two nearest neighbors on the ring. Like the MPI-based implementations, this implementation also supports only nearest neighbor temperature exchanges.

The Folding@home [12] project at Stanford University has proposed a multiplexed replica exchange algorithm. The algorithm uses multiplexed-replicas with a number of independent molecular dynamics runs at each temperature, and attempts exchanges of configurations between these multiplexed-replicas. In this formulation, the efficiency of the simulation is enhanced as a number of independent molecular dynamics simulation replicas are run at each temperature and there are a larger number of potential exchange partners available. Further, the multiplexing between replicas is arranged in such a way that the discrepancy between exchange partners is reduced. In contrast, Salsa improves simulation efficiency by eliminating the limitation of nearest neighbor exchanges, instead of introducing redundant computations. Both algorithms, however, use parallelism to improve the efficiency of the simulation.

To the best of our knowledge, the work presented in this paper is the first to address the decentralized and asynchronous parallel implementation of replica exchange. This not only improves scalability but also improves efficiency by enabling non-nearest neighbor temperature exchanges, which is desirable for simulations with a large number of replicas.

B. Shared Space Abstractions

One of the earlier research efforts on virtual shared space abstractions was the tuple space model proposed by N. Carrier and D. Gelernter in [11]. Subsequently several research projects have addressed the model and its implementation. These include commercial products and research prototypes with varied foci, such as JavaSpaces [14], TSpaces [15], XMLSpaces [16], Lime [17], PeerWare [18], PeerSpace [19], and Comet [20]. The virtual shared space abstraction provided by Salsa is customized towards the application domain, which also allows it to avoid some of the overheads and consistency issues of a general shared space.

VI. Conclusion and Future Work

The paper presents a novel implementation of replica exchange algorithm using Salsa, a framework that presents a shared space abstraction at runtime to user applications. The Salsa-based replica exchange supports exchange between non-nearest neighboring temperatures, asynchronous communication to enable overlapping communication with computation, decentralized communication paradigm to avoid the central bottleneck in a client/server version of the implementation. The approach helps improving the overall efficiency and scalability of the simulation.

The overall goal of the project is to enable large-scale Grid-based parallel and distributed molecular simulations of protein structural changes and drug binding to proteins. Specific tasks include (1) implementing a prototype interaction and coordination framework, based on Salsa, for wide-area distributed replica exchange simulations, (2) developing, deploying and evaluating the Grid-based Impact implementation, and (3) using the grid-based Impact implementation to provide scientific insights .

While Salsa, as presented in this paper, has be specially customized for the replica exchange algorithm, the underlying approach, i.e., developing domain specific and semantically specialized associative interaction/coordination spaces, can potentially support a more general class of applications that exhibit extremely dynamic and complex coordination and communication behaviors. We will investigate other applications that can benefit from this approach and generalize Salsa to support these applications.

Acknowledgements

The research presented in this paper is supported in part by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826, NIH GM-30580 and by the DOE SciDAC CPES FSP.

References

[1] The Message Passing Interface (MPI) Standard. http://www-unix.mcs.anl.gov/mpi/

[2] J.L. Banks, H.S. Beard, Y. Cao, A.E. Cho, W. Damm, R. Farid, A.K. Felts, T.A. Halgren, D.T. Mainz, J.R. Maple, R. Murphy, D.M. Philipp, M.P. Repasky, L.Y. Zhang, B.J. Berne, R.A. Friesner, E. Gallicchio, and R.M. Levy. Integrated Modeling Program, Applied Chemical Theory (IMPACT), J. Comp. Chem., 26, 1752-1780 (2005).

[3] H. Nymeyer, S. Gnanakaran, and A. E. Garcia. Atomic SImulations of Protein Folding, Using the Replica Exchange Algorithm. Methods in Enzymology. Vol. 383. page 119-149. (2004)

[4] R. Swendsen and J. Wang. Phys. Rev. Lett. 57, 2607 (1986)

[5] C. Geyer and E. Thompson. J. Am. Stat. Assoc. 90, 909 (1995)

[6] E. Marinari and G. Parisi. Europhys. Lett. 19, 451 (1992)

[7] K. Hukushima and K. Nemoto. J. Phys. Soc. Jpn. 65, 1604 (1996)

[8] K. Y. Sanbonmatsu and A. E. Garcia. Proteins 46, 225 (2002)

[9] Y. Sugita and Y. Okamoto. Replica-exchange molecular dynamics method for protein folding. Chemical Physics Letters 314 (1999) page141-151.

[10] A. K. Felts, Y. Harano, E. Gallicchio, and R. M. Levy. Free energy surfaces of β -Hairpin and α -Helical peptides generated by replica exchange molecuar dynamics with AGBNP implicit solvent model. Proteins: Stucture, Function, and Bioinformatics 56: 310-321 (2004).

[11] N. Carriero, D. Gelernter. Linda in context. Communications of the ACM, Volume 32, Issue 4, pp.444-458, April 1989, ISSN:0001-0782.

[12] Y. M. Rhee and V. S. Pande. Multiplexed-replica exchanged molecular dynamics method for protein folding simulation. Biophysical Journal. Volume 84. February 2003. Page 775-786.

[13] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen. GROMACS: a message-passing parallel molecular dynamics implementation. Computer Physics Communications 91 (1995) 43-56.

[14] J. et al. JavaSpace Specification 1.0. Technical report, Sun Microsystems, 1998.

[15] T. J. Lehman, S. W. McLaughry, and P. Wycko. TSpaces: The Next Wave. In Proceedings of Hawaii International Conference on System Sciences, 1999.

[16] R. Tolksdorf and D. Glaubitz. CoordinatingWeb-based Systems with Documents in XMLSpaces. In Proceedings of the Sixth IFCIS International Conference on Cooperative Information Systems, 2001.

[17] A. Murphy, G. Picco, and G.-C. Roman. Lime: A Middleware for Physical and Logical Mobility. In Proceedings of the 21st International Conference on Distributed Computing Systems, pages 524C536, 2001.

[18] G. Cugola and G. Picco. PeerWare: Core Middleware Support for Peer-To-Peer and Mobile Systems. Technical report, Politecnico di Milano, 2001.

[19] N. Busi, C. Manfredini, A. Montresor, and G. Zavattaro. PeerSpaces: Data-driven Coordination in Peer-to-Peer Networks. In Proceedings of the 2003 ACM symposium on Applied computing, pages 380C386. ACM Press, 2003.

[20] Z. Li and M. Parashar. Comet: A Scalable Coordination Space in Decentralized Distributed Environments. In Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P 2005), San Diego, CA, USA, IEEE Computer Society Press, July 2005.

[21] V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, and M. Parashar. High Performance Threaded Data Streaming for Large Scale Simulations. 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA, November, 2004.

[22] R. Zhou and B. J. Berne, Smart walking: A new method for Boltzmann sampling of protein conformations. J. Chem. Phys. 107 (1997) 9185-9196.