Domain Decomposition vs. Master-Slave in Apparently Homogeneous Systems

Cyril Banino-Rokkones

Department of Computer and Information Science (IDI)
Norwegian University of Science and Technology (NTNU)
Sem Sælandsvei 7-9 NO-7491 Trondheim, Norway
Cyril.Banino@idi.ntnu.no

Abstract

This paper investigates the utilization of the master-slave (MS) paradigm as an alternative to domain decomposition (DD) methods for parallelizing lattice gauge theory (LGT) models within distributed memory environments. The motivations for this investigation are twofold. First, LGT models are inherently difficult to parallelize efficiently with DD methods. Second, DD methods have proven useful for homogeneous environments, but are impractical for heterogeneous and dynamic environments. Besides, many modern supercomputer architectures that look homogeneous (such as multi-core or SMP), are in fact heterogeneous and dynamic environments. We highlight this issue by comparing a traditional first-come first-served MS implementation to a simple but yet efficient selective MS scheduling strategy that automatically accounts for system heterogeneity and variability. Our experimental results with the parallelization of our LGT model, reveal that the selective MS implementation achieves good efficiency, but lacks of scalability. In contrast, the DD method is highly scalable, but at the expense of a poor efficiency. These results open up for a hybrid approach, where the MS and the DD methods would be combined for achieving scalable high performance.

1. Introduction

Domain decomposition (DD) methods have been studied extensively because of their utility in a wide range of application areas such as, physics, chemistry, solid and fluid mechanics, or climate modeling. Domain decomposition on parallel computers consists in splitting the computational domain into smaller sub-domains, each of which is assigned to one processor. Then, during the execution, computation and communication phases alternate, as neighboring processors (in the topological decomposition) need to periodically exchange data located on the boundaries of their local domains.

On the one hand, the efficiency of DD methods is strongly affected by the heterogeneity and variability present in the underlying computing system. Indeed, DD methods are efficient only when the computational load is well balanced among the processors. The processors being tightly coupled by the communication phases, the execution proceeds at the pace of the slowest processor. For homogeneous and stable systems, the computational domain needs simply to be decomposed into p equally-sized sub-domains. For heterogeneous environments on the other hand, things get complicated as the domain must be decomposed into p sub-domains whose size must be proportional to the processor computational speeds. In dynamic environments, where resources exhibit unforeseeable performance fluctuations, things get even worse, as it becomes necessary to frequently redistribute the computational domain among the processors.

In addition, many modern supercomputer architectures (such as multi-core or SMP clusters) that look homogeneous, hide in fact an heterogeneous and dynamic environment. For instance, processors located within the same node are actually competing for shared resources (e.g. caches), and intra-node communication is typically much faster than inter-node communication. The impact of the heterogeneity and variability hidden in the system on the performance of DD methods is difficult to evaluate, but undoubtedly degrades the performance.

Last but not least, an important issue concerns fault tolerance. Currently, the most common technique for handling fault tolerance within DD methods is checkpoint/restart. That is, checkpoints are saved to disk periodically, and if a processor fails, the computation halts and restarts from the last consistent checkpoint. For lengthy applications that make use of a large number of processors, failures are more likely to be the rule rather than the exception. In these con-

1-4244-0910-1/07/$20.00

ditions, the checkpoint/restart technique could take longer than the time to the next failure. Hence, there is a need to survive failures without relying on global recovery operations.

On the other hand, the efficiency of DD methods is directly subject to the characteristics of the scientific problem to be solved. Some problems are more suited to DD methods than others. In this paper, we are interested in lattice gauge theory (LGT) models, a class of Monte Carlo (MC) simulations particularly difficult to parallelize efficiently with DD decompositions in distributed memory environment (i.e. when message passing is unavoidable).
LGT models belong to the wide class of *stencil computations*, where typically, each site in a multi-dimensional lattice is updated with contributions from a subset of its neighbors (see Figure 1). For each iteration, the stencil kernel is applied to all the lattice sites - usually the boundaries receive a special treatment.
When parallelizing LGT models with DD methods, one must ensure at all times that processors owning neighboring sub-domains do not update adjacent sites simultaneously. Although neighboring lattice sites may be updated in any order, physical properties impose these updates to happen sequentially, creating thus constraining data dependencies.
The message passing paradigm provides a simple and natural way to orchestrate the lattice updates without violating these data dependencies. Communication events can be used as *tokens*, such that incoming messages from neighboring processors trigger the update of the corresponding sub-domain boundary. However, in the case of LGT models, this technique introduces a significant amount of idle time on the processors, degrading significantly the parallel efficiency.

Thus, there are two main reasons for considering an alternative way to DD methods: The inadequacy of DD methods for dealing with heterogeneous and dynamic environments; and the lack of efficiency of DD methods for parallelizing LGT models. In this paper, we study the suitability of the master-slave paradigm (MS) as an alternative to DD methods for implementing LGT models within distributed memory environments. The MS paradigm admittedly comes along with some limitations, but presents most of the features required for dealing not only with LGT models, but also with heterogeneous and dynamic environments.
In its simplest form, the MS paradigm works as follows. The master initially distributes one task to every slave. The slaves compute their tasks and send the results back to the master, which triggers the latter to send additional tasks. The main assets of the MS paradigm are *flexibility* and *robustness*. As slaves execute tasks at their own paces, they will automatically request tasks proportionally to their computing speeds. This is popularly known as *self-scheduling*, *demand-driven* or *first-come first-served* (FCFS). By construction, FCFS adapts well to the performance fluctuations of the computational resources. If a slave suddenly gets some external load, it will process tasks less rapidly, and hence request tasks less frequently. When the conditions get back to normal, the slave will request tasks at its maximal pace. However, FCFS is not efficient when point-to-point communication times are heterogeneous. In that case, resource selection strategies become necessary in order to efficiently utilize the available computing and communication resources. In this paper, we show that a simple, yet effective, *selective* scheduling scheme is more appropriate for dealing with heterogeneous and dynamic environments than the traditional FCFS strategy.
Finally, the loosely coupled structure of the MS paradigm presents only one *single point of failure* in the form of the master process. This means that one only needs to backup the master node to achieve reliability. If some slave processes die, the computation can still carry on with the remaining slaves.

The rest of this paper is organized as follows. Section 2 reviews previous work related to LGT model parallelizations, DD methods and the MS paradigm. Section 3 introduces the LGT model considered in this study. The DD and MS parallelizations of the LGT model are presented respectively in Section 4 and Section 5. In addition, our MS selective scheduling strategy is exposed and compared to the FCFS strategy in Section 5. Section 6 reports an experimental comparison between the MS and the DD implementations. Future work is discussed in Section 7. Finally concluding remarks are given in Section 8.
The experiments reported in this study have been performed on a SMP cluster composed of 100 HP Integrity rx4640 server nodes. Each SMP node comports 4 itanium2 processors clocked at 1.3 GHz sharing 4 GB of memory. The 100 SMP nodes are interconnected with the Infiniband network.
In all the experiments reported in this paper, the number of iterations was arbitrarily fixed to 500 in order to highlight differences between the different implementations while keeping measurement times relatively low. The experiments were performed on a dedicated set of computing nodes, which reduces external interferences. Finally, all the performance curves reported in this study correspond to the average values over 3 runs.

2. Related Work

Several studies have considered parallelizing MC simulations using DD methods [3, 13, 17, 21, 29]. MC simulations for the Ising model, which uses only nearest-neighbors interactions (6-point stencil in 3 dimensions), have been successfully implemented on shared-memory systems with checker-board algorithms. The lattice sites are sorted into

a red set (where sum of coordinates is even) and a black set (where sum of coordinates is odd) in a checker-board fashion. Thus, all the red sites can be updated simultaneously, and so it is for the black sites. Checker-board algorithms have been ported onto distributed memory systems by several studies [13, 15, 21]. For each iteration, all the processors start by updating one color set, say the red one. Thereafter the nodes exchange the red sites located on the boundaries, and do the same with the black set. This approach performs boundary-exchanges with two messages per boundary. For longer-range or more complex interactions models, such as the one presented in this study, new updating schemes that fit with the stencil kernel must be applied. In these conditions, the checker-board is likely to be composed of at least four colors, leading to boundary-exchanges with four messages per boundary.

Santos et al. [28, 29] conducted research on MC simulations for 2D and 3D Ising models in another direction. Each local domain is partitioned into different sets, that are updated one after another, in alternation with communication events. For each iteration, all the processors update the same set of their local domain, in order to avoid situations where remote but adjacent site updates would enter in conflict. Then some boundary-exchanges take place, allowing the parallel computation to proceed with the next set. The number of sets composing the local domain is dependent on the chosen decomposition (2 for 1D, 3 for 2D, and 4 for 3D). The result is an increase of the number of messages required for the boundary-exchanges (namely 2 for 1D, 8 for 2D, and 24 for the 3D decomposition).

The two aforementioned methods handle the data dependencies between neighboring sites by increasing the number of messages per boundary-exchange, which considerably increases the communication run-time cost. Recently, we provided token-passing algorithms based on DD methods that minimize the number of messages exchanged between neighboring processors [3]. Our algorithms are presented and explained in great details in [3], but we provide a brief summary in Section 4.

Although DD methods are relatively easy to deploy efficiently on homogeneous environments, dealing with heterogeneous and dynamic environments is a much more complicated task. Several studies have been conducted on deploying DD methods within heterogeneous environments [4, 5, 18, 19, 22]. In most cases, the problem is reduced to the problem of partitioning some mathematical objects, such as matrices, sets or graphs [9]. The main difficulty resides in the combinatorial nature of the problem which typically turns out to be NP-complete. Even though efficient (i.e. polynomial) heuristics are derived, the dynamic nature of the underlying platform makes static strategies not well suited to these environments. In dynamic environments, the processor speeds and network contention will fluctuate during the execution requiring online load redistribution mechanisms. Online redistribution is difficult to handle, as it poses the question of when should one redistribute the load? And how to measure the quality of a load distribution? Beaumont et. al. [6] consider the matrix multiplication problem in heterogeneous and dynamic environments, and propose to redistribute the load only between large static-phases. Still, one must find a good load redistribution frequency, since a too conservative approach may not result in significant improvements, wheras being too aggressive may incur too much overhead. An important point stressed by Beaumont et. al. is the necessity to minimize the amount of communication when redistributing the load. The amount and location of the data should be taken into account in order to maintain the relative position of the processors, otherwise the cost of the redistribution may be prohibitive. Similarly Mahanti and Eager [22] find that data migration costs should be minimized for efficient redistribution, and propose redistribution policies that try to leave the relative position of the nodes unaltered. In their work, Mahanti and Eager consider data redistribution following addition/removal of processors.

Although these studies on DD methods within heterogeneous environments present interesting results that give insights on the problem difficulties, these different strategies typically rely on a centralized algorithm to (re)distribute the work among the heterogeneous processors. This clearly poses the question of the scalability of the approach. On the other hand, the problem of online load redistribution frequency is difficult to address without disposing of some form of centralized information about the platform state.

Similarly to DD methods, the MS paradigm is well known and has been the subject of a wealth of studies both in the context of Cluster computing [10, 24, 25] and of Grid computing [7,12,16]. Usually the applications implemented under the MS paradigm are composed of a large number of independent tasks. All the popular scheduling strategies designed for minimizing the total execution time, hand out tasks by chunks of decreasing size, in order to reduce the scheduling overhead while achieving a good load balance at the end of the execution [14]. However, this kind of MS strategies cannot be utilized in our study, because the tasks composing our target applications are not fully independent of each other (more on this in Section 5).

3. Our lattice gauge theory model

Lattice spin and gauge theories are studied extensively in many areas of physics, especially in particle and condensed matter physics. The spin and gauge field variables are defined on every site of a multi-dimensional lattice, and the thermodynamic properties of the system can be deduced from the partition function, which is a sum over all possible

configurations of the fields. Exact solutions to these multi-dimensional sums are rare and in general one must resort to some numerical approximation. The largest and most important class of numerical methods used for this problem is the Monte Carlo (MC) method, which in stead of doing the sum over all configurations, utilizes random numbers to mimic the random thermal fluctuations of the system from one configuration to the other. A considerable proportion of the computational resources used by physicists around the world is spent on MC simulations.

The LGT model studied in this paper is a superconductor model in which a real valued scalar field is coupled to a real valued vector field. This model is a simplified version of the one presented in [1]. The MC algorithm used for the simulations is the celebrated Metropolis algorithm [23] which can be described the following way.

1. Pick one site in the lattice and suggest new values for the fields at that site.
2. Calculate the difference in energy $\Delta E = E_{new} - E_{old}$ for the move, or update.
3. Draw a random number r$\in [0, 1\rangle$ and accept the new values if $\min\{1, e^{-\Delta E/T}\} > r$.
4. Repeat step 1 to 3 until enough statistics are gathered.

The computational domain is a 3-dimensional lattice with periodic boundary conditions. The charge of the system (reparted among all the lattice sites) couples a scalar field and a three-dimensional vector field. Hence, to each lattice site are associated 4 double precisions real numbers. The local energy E_s at one site s in the lattice is dependent on the nearest neighbor of s, and half of the next nearest neighbors of s. More formally, all the sites *adjacent* to s are involved in the computation of E_s.

Definition 1 *Two lattice sites* $s_1 = (x, y, z)$ *and* $s_2 = (t, u, v)$ *are said to be* **adjacent** *if and only if* $(t, u, v) \in \{(x, y, z-1), (x, y+1, z-1), (x+1, y, z-1), (x-1, y, z), (x-1, y+1, z), (x, y-1, z), (x, y+1, z), (x+1, y-1, z), (x+1, y, z), (x-1, y, z+1), (x, y-1, z+1), (x, y, z+1)\}$*, as depicted in Figure 1.*

4. Domain decomposition implementation

In [3], we proposed token-passing algorithms based on DD methods that minimize the amount of communication, i.e. one message per neighboring processor per iteration. Our token-passing algorithms are built upon a classic technique for allowing communication overlap with computation in DD computations. The idea is to partition each local domain into an inner set and an outer set [2, 26, 27].

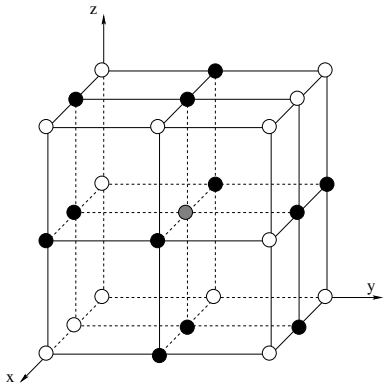
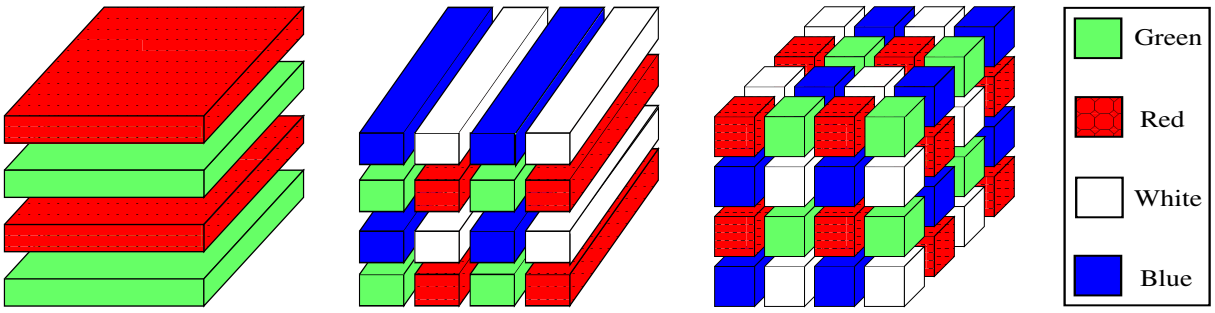


Figure 1. Stencil of the LGT model. Black sites are used to update the grey site, they are *adjacent* to the grey site.

The inner set is updated while waiting for the boundaries from neighboring processors, and thereafter the outer set is in turn updated. The reception of all the boundaries is liken to the reception of a virtual token, that allows updating the outer set. Thereafter processors hand on the token by sending their updated local boundaries to their neighbors.

In order to respect the data dependencies imposed by the LGT model (sequential updates of adjacent lattice sites), the processors are sorted into different color sets (see Figure 2), such that processors of the same color can update their exterior sites simultaneously. For the 1D case, two colors are necessary and sufficient, whereas four colors are required for the 2D and 3D decompositions. Then, an ordering is established among the colors to orchestrate the updates of the outer sets. For the 1D case, green processors start ahead of the red processors, while the color ordering of 2D and 3D decompositions is 1) green, 2) red, 3) white and 4) blue. Figure 3 sketches the parallel execution of the token-passing algorithm based on a 2D decomposition.

MPI features like persistent requests and derived datatypes have been used for implementing the successive boundary-exchanges. Special care has been taken when posting and completing the communication requests such that the MPI *ready* communication mode could be used. All these decisions contribute to keep the communication overhead to a minimum. Also, we used the *diagonal communication elimination* technique [8, 13], which consists of including ghost cells within messages in order to avoid diagonal communications for exchanging lattice sites located on the edges of the sub-domains.. At last, for the sake of portability, non-blocking requests have been used in order to exploit the inherent computation-communication overlap of the partitioning method, even though many implementations cannot overlap without extra hardware in the form of a communication co-processor.



(a) 1D. (b) 2D. (c) 3D.

Figure 2. Domain decomposition and color partitioning schemes.

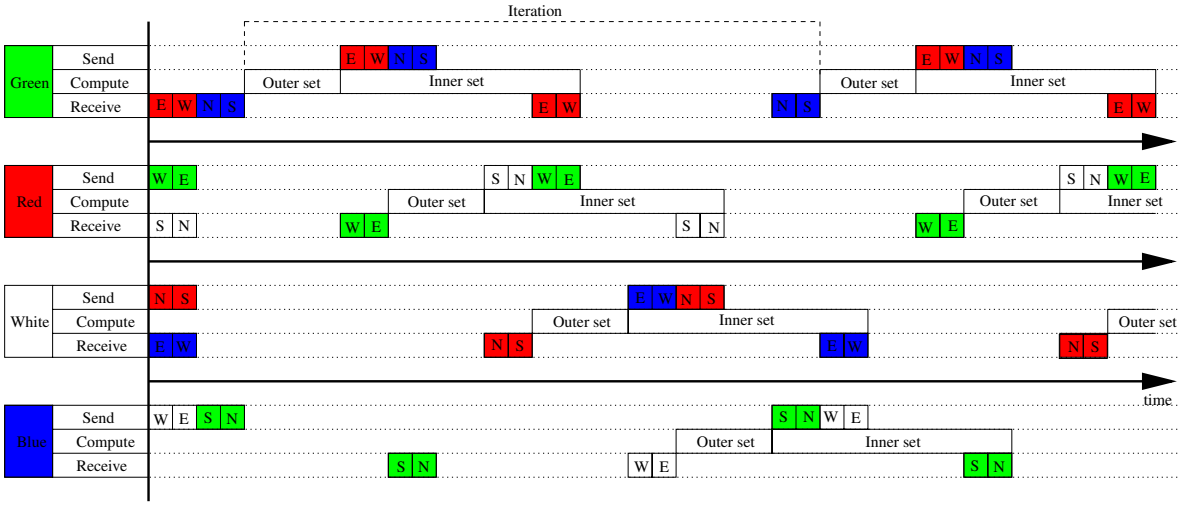


Figure 3. Sketch of the token-passing algorithm with a 2D decomposition.

Figure 4 (a) depicts the respective speed-up of the three different token-passing algorithms when using 32 processors. The better performance of the 1D decomposition over the 2D and 3D decompositions is certainly due to the complicated token round-trip trajectories of the latter decompositions as opposed to the much simpler trajectory for the 1D case. Indeed, the 1D token round-trip imposes only 1 outer set update and 4 messages, as opposed to 3 outer set updates and 8 messages for the 2D case, and 3 outer set updates and 12 messages for the 3D case (see [3] for a thorougher performance analysis). For all the experiments, the token round-trip dominates the total iteration run-time cost, meaning that processors are starving, waiting for the token to arrive. The run-time costs for updating the inner sets were roughly equivalent for the 3 domain decompositions, which means that processors are starving longer under the 2D and 3D decompositions than for the 1D decomposition.

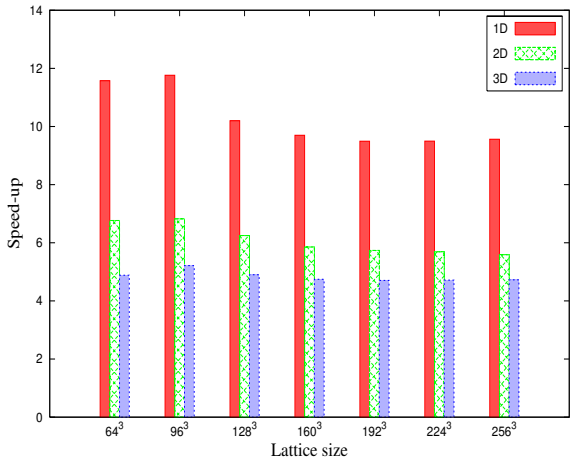
Although, Prieto et al. [26] showed that the separation of the inner and outer set updates may degrade the performance due to the large distance between the memory locations of the exterior sites (causing a poor cache memory exploitation when updating the outer set), we found that our token-passing algorithms were scalable with an efficiency comprised between 0.25 and 0.5 depending on the problem size and number of processors utilized (see Figure 4 (b)).

5. Master-slave implementation

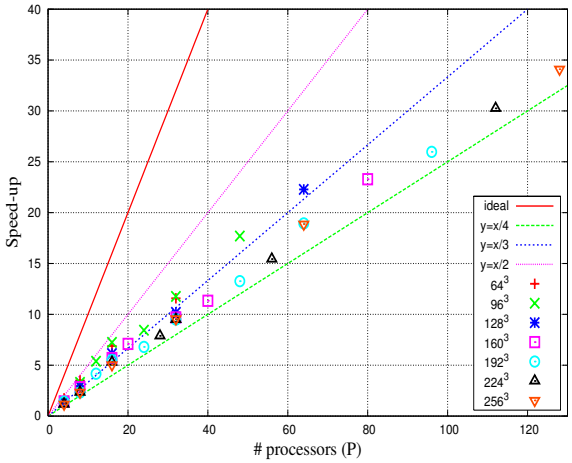
5.1. Task partitioning

The sites of the 3-dimensional lattice must be partitioned into disjoint sets to allow for parallel execution. The goal is to enable the processors to work on different parts of the lattice simultaneously. We rely on the same domain decomposition and the same color code than the ones used for the DD implementation of the LGT model (see Figure 2), such that blocks of the same color can be processed simultaneously. In our context, each block represents a task to be scheduled by the master.

Depending on the chosen decomposition and the LGT



(a) Speed-up with 32 processors.



(b) 1D Speed-up

Figure 4. Speed-up of our token-passing algorithms based on DD methods. Reproduced from [3].

model stencil, different dependencies take place between neighboring blocks of different colors. For the 1D decomposition, each block is dependent on 2 blocks of the opposite color (above and beneath). For the 2D decomposition, each block is dependent on 2 blocks of each of the other colors. At last, for the 3D decomposition, each block is dependent on 4 blocks of each of the other colors.

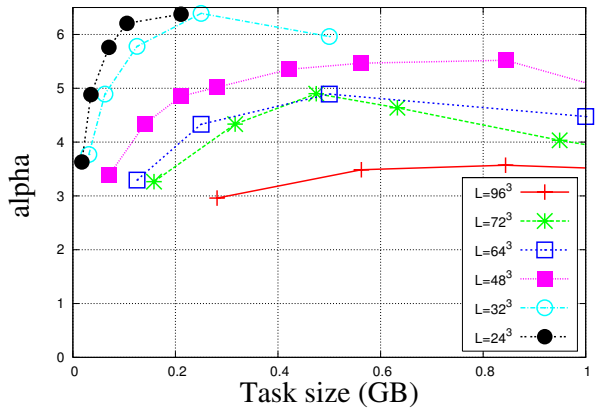
In order to respect the site update dependencies, the master deals with one color at a time. Thus the scheduling overhead on the master node is alleviated by only keeping track of block dependencies from one color to another. To detect block eligibility, the master maintains for each block a dependency variable (integer), as well as pointers to the dependency variables of the adjacent dependent blocks. Initially all the dependency variables are set to the number of dependencies generated by the task partition. Upon reception of a computed block, the variables of all the adjacent blocks are decremented, and if some of them become equal to zero, the corresponding blocks become eligible for computation. In that case, the block pointers are inserted into a FIFO queue holding all the blocks eligible for computation. This mechanisms relieves the master from waiting for the termination of a given color to switch over the next color. Instead, the color transition happens smoothly by delegating blocks as soon as they become eligible for computation.

The master must decompose the global lattice in such a way that there are enough blocks available to the slaves. On the one hand, the number of blocks should be large enough in order to dispose of enough eligible tasks at all times to keep the slaves busy. On the other hand, the master should determine an appropriate task size in order to reduce the overhead incurred by the total amount of communication combined with the post processing of the blocks (copy operations due to the periodic conditions of the 3-dimensional lattice).

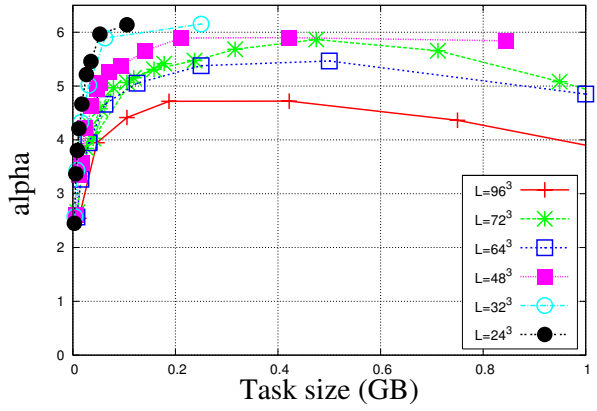
The first thing to determine is which task partitioning scheme gives the best performance. This includes finding the best domain decomposition and the optimal block size. A simple way to compare the different task partitioning consists in estimating the ratio α between the time it takes a slave to process a task, and the time it takes the master to send the task, receive the associated results and post-process the task. The ratio α gives an indication on how scalable is the MS implementation. The larger this ratio is, the better will perform the MS implementation, as it would be able to use more processors. Actually, this ratio gives an indication on the number of slaves that the master can handle, assuming that the slaves have homogeneous computing and communication characteristics.

Figure 5 reports the results of an experiment with 2 processors (a master and a slave), for different problem and task sizes. The experimental values for the different α ratios obtained during this experiment indicate that the master would not be able to handle more than 6 slaves on the testbed machine, which might seem a low number at first sight. In addition, when problem size increases, the α ratio decreases. For every decomposition, a good task size seems to be situated between 250 and 500 MB.

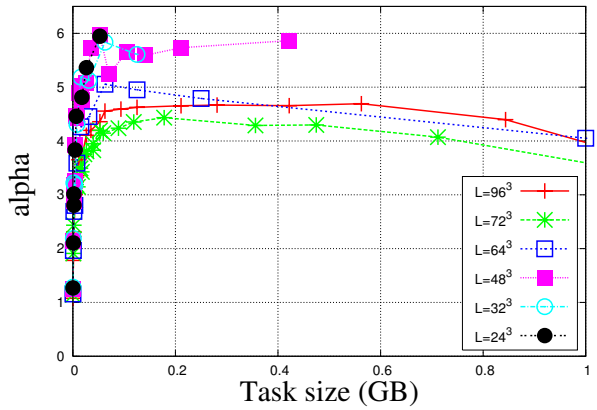
The 2D decomposition performs slightly better than the other ones, most likely due to the shape of the blocks and consequently to the derived datatypes involved in the communications. Indeed, when delegating a task, the master must extract a block from the global lattice, whose shape depends on the chosen decomposition. The 2D decomposition is a good compromise between few large blocks (1D) and many small blocks (3D).



(a) 1D task partition.



(b) 2D task partition.



(c) 3D task partition.

Figure 5. Estimating the α ratio for different problem and task sizes. Note that the task decomposition schemes must deal with lattice size constraints that dictate the sizes and shapes of the blocks. Hence, the more dimensions are used by the task partitioning scheme, the smaller the tasks can be.

5.2. Selective scheduling

Because our computational domain is decomposed into relatively few tasks that become eligible for computation alternatively throughout the computation, we aim at task throughput maximization instead of total run-time minimization. Our scheduling strategy consists in handing out the tasks one-by-one, in a demand-driven fashion. If several slaves are competing for a task, then the master must decide which one to serve according to a priority scheme.

Since all the tasks are computationally identical, we let $P_s(t)$ denote the time it takes to slave s to process a task at time-step t. Further, it takes $C_s(t)$ time units for the master to send a task to slave s at time-step t, and $C'_s(t)$ time units for the slave s to return the results to the master at time-step t.

Our MPI parallel implementation involves advanced programming techniques such as derived datatypes, non-blocking communications and persistent requests. This complicates the online monitoring of the different communication events. Therefore, for each slave s, we define the *task round-trip* at time step t, noted $R_s(t)$, as follows: $R_s(t) = C_s(t)+P_s(t)+C'_s(t)$, that corresponds to the time it takes for sending a task to slave s plus the time it takes slave s to compute the task plus the time it takes to send the results back to the master. Throughout the computation, the master can monitor the value R_s of each slave in order to make efficient scheduling decisions. Thus, we account for the possible performance fluctuations of both computation and communication resources throughout the computation. Monitoring R_s simply consists in starting a timer right before sending a task to a slave, and stopping the timer when the results have returned.

When several slaves are in competition for receiving a task, the master will choose the one with the smallest R_s value. Indeed, no distinction is made between the computation and communication run-time costs relative to a slave, since the tasks are not really independent of each other. It is indeed, important that tasks come back as soon as possible in order to allow other tasks to become eligible for computation.

At the beginning of the execution, all the slaves are given a task, which allows to initialize all the R_s values. Then, as the computation proceeds, the R_s values are updated with the newest value measured by the master. More advanced methods based on averages over the last n values or on performance predictions exist [30], but our simple method gave satisfactorily results.

5.3. FCFS vs. selective scheduling

To demonstrate the need for priority schemes, we compared our selective scheduling strategy to the FCFS scheme, which works without priorities. Figure 6 (a) depicts the ratio of the task throughput of the selective scheme over the task throughput of the FCFS scheme. Clearly, selective scheduling achieves a higher throughput than FCFS in most situations. This phenomenon strengthens as the number of slaves increases, which corroborates our hypothesis that intra-node interferences as well as inter-node communications introduce heterogeneity and variability in the computing environment.

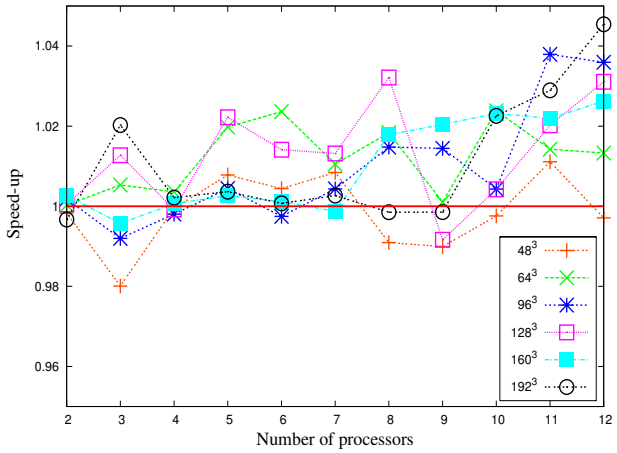
Figures 6 (b) and (c) report the task share among the slaves for the selective and FCFS schemes respectively. We observe that for the selective scheme, 3 slaves get a bigger share of the tasks than the others. This is not surprising since each SMP node is composed of 4 processors, meaning that 3 slaves are located on the same node as the master. The intra-node communication (shared-memory) being faster than inter-node communication (message passing), the slaves located on the same SMP node as the master will be prioritized if they are in concurrency with other slaves located on a different SMP node because exhibiting a smaller R_s value. This phenomenon is less visible for the FCFS strategy.

Finally, note that the cluster was used in dedicated mode, meaning that no external load other than operating system calls or network contention fluctuations interfered with our application. Consequently, all the nodes have roughly the same computing power which explains the linearity of the curves. Nonetheless, when using a high number of slaves, the selective scheme seems to adapt to some interferences that take place, while the FCFS scheme maintains a fair share of the tasks. Thus, although the system *looks homogeneous*, there are still a certain amount of heterogeneity and variability in the system that degrade the overall performance.

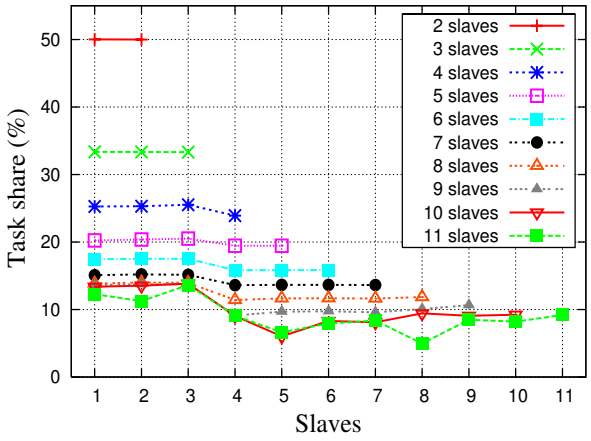
6. Domain decomposition vs. master-slave

Figure 7 reports the speed-up and efficiency obtained with the selective MS implementation using a 2D task partitioning scheme (Figures (a) and (c)), and with the token-passing algorithm using a 1D decomposition (Figures (b) and (d)). One can observe that the MS implementation scales well up to 6 slaves, and thereafter begins to saturate. This result conforms with the experiment conducted for determining the appropriate task partitioning schemes (see Section 5.1) predicting that the master could not handle more than 6 slaves efficiently.

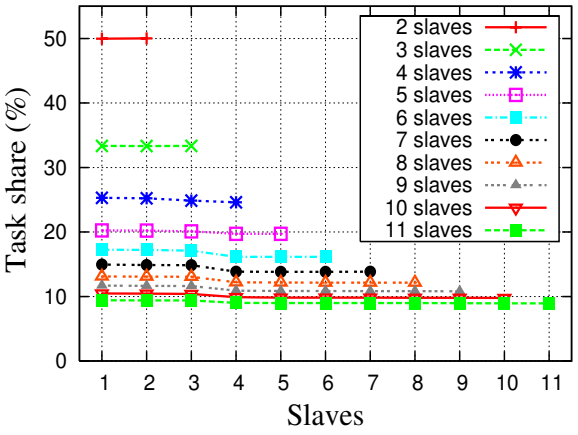
Note that the MS implementation achieves a perfect speed-up up to 3 slaves (if the master is not accounted). When



(a) Task throughput ratio $\frac{selective}{FCFS}$.

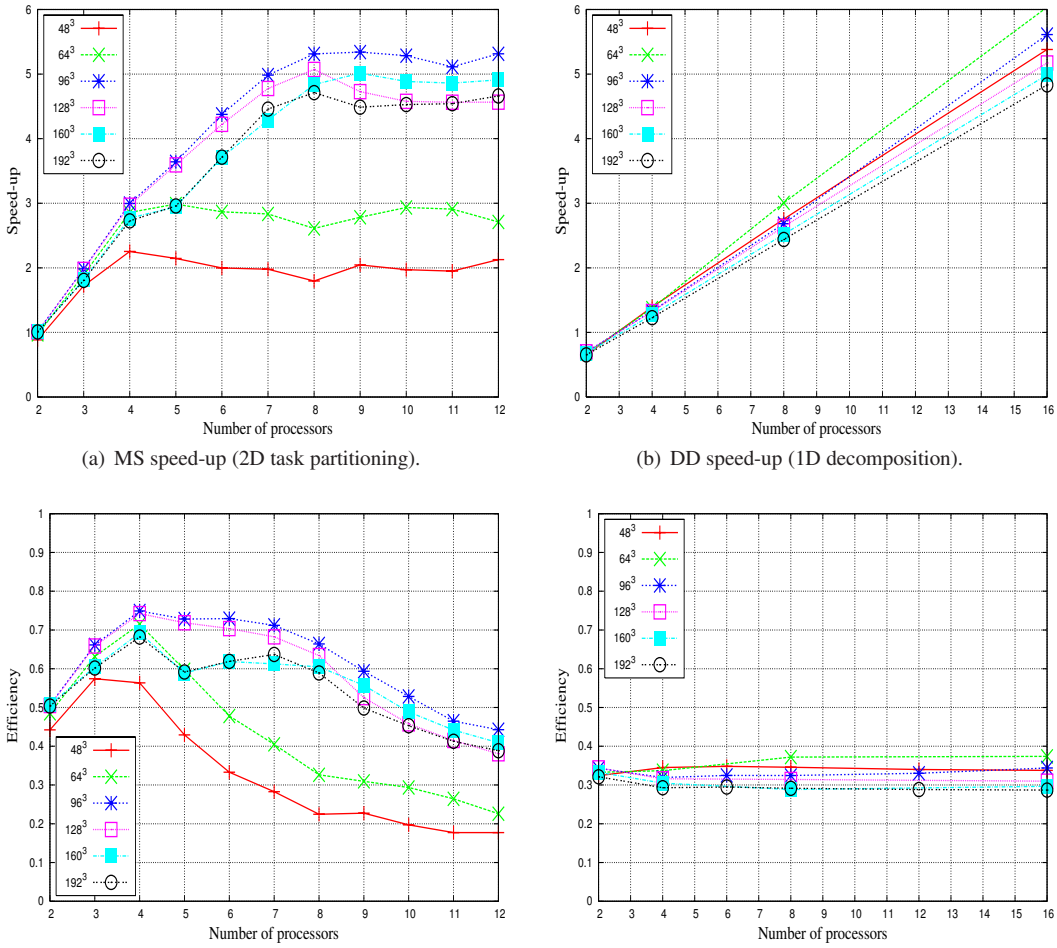


(b) Task share for the selective scheme.



(c) Task share for the FCFS scheme

Figure 6. Comparison of the selective and FCFS schemes for $L = 128^3$.



(a) MS speed-up (2D task partitioning).

(b) DD speed-up (1D decomposition).

(c) MS efficiency (2D task partitioning).

(d) DD efficiency (1D decomposition).

Figure 7. Comparison of the selective MS and the DD implementations.

using more than 3 slaves, the inter-node communications begin to drive the performance away from optimality.
The poor performance of the MS implementation for small problem sizes ($L = 48^3$ and $L = 64^3$) comes from our task partitioning scheme that utilizes blocks of size greater than 250 MB. Thus, for small problem sizes, there were simply not enough independent tasks to feed all the slaves. In such situation, one should use a finer grained task partition scheme, albeit there is a limit on how small a task can be.

As opposed to the MS implementation, the DD implementation is highly scalable, albeit this comes at the expense of a poor efficiency. For a small number of processors, the DD implementation is less efficient than the MS implementation. Hence, it seems that the MS approach is better suited for dealing with our LGT model than parallel algorithms based on DD methods. However, the lack of scalability of the MS implementation makes it useless for large-scale simulations.

7. Future work

The natural solution to tackle the lack of scalability of the MS paradigm, is to deploy several masters [20]. A direction for future work would be to design a hybrid approach where the MS paradigm and DD methods would be used in concert. The computational domain would be decomposed among few processors (the masters), but each master would update its sub-domain using the MS paradigm. The number of masters to deploy depends on the problem to be solved as well as on the underlying computing system. For our LGT model and our SMP cluster, a master could manage a SMP node (or span over two SMP nodes).
Such hybrid approach combines the benefits of the two paradigms: The MS flexibility with the DD scalability. Interestingly, inter-master load balancing could be tackled at two levels. First, the computational load can be redistributed between masters. This approach makes it possible to use existing load redistribution strategies [4–6,18,19,22].

But a more promising approach would be to handle the load redistribution as a slave redistribution. If a master experiences a lack of computing power from its slaves, it could request additional slaves from other masters. Hence, slaves could be traded between masters on demand. This "computing-power" balancing mechanism is more flexible and practical than traditional load-balancing algorithms, as data would not need to be migrated throughout the computation.

Fault tolerance still becomes easier to handle as one needs only to back-up the master processes. For that matter, note that any slave can act as a master whenever needed. Hence, masters can periodically back-up their data, by sending a copy to one or few slaves that would replace them in case of failure. For such implementations, one could use the Fault Tolerant MPI library (FT_MPI) [11], which offers a range of recovery options other than just returning to some previous check-pointed state. This is especially useful in the case of slave failure since the computation can in principle proceed seamlessly.

8. Conclusion

High performance computing systems are no longer stable and fully homogeneous. This greatly complicates the efficient deployment of traditional DD methods, since applications must deal with system heterogeneity, resource performance fluctuations and resource failures. In addition to that, there are certain classes of problems for which DD methods are inappropriate, such as the LGT model presented in this paper. Hence, there are two good reasons for considering an alternative way to DD methods.

In this paper, we study the suitability of the MS paradigm as an alternative to DD methods for implementing LGT models within distributed memory environments. We provide three different MS implementations based on three task partitioning schemes. More importantly we demonstrate, via a comparison between a selective and the FCFS scheduling strategies, that apparently homogeneous systems used in dedicated mode are actually heterogeneous environments subjects to unforeseeable resource performance fluctuations.

Overall, our experimental results reveal that the MS implementation achieves very good efficiency on few processors, but lacks of scalability. In contrast, the DD method is highly scalable, but at the expense of a poor efficiency. The peculiarity of the LGT model, namely the constraining data dependencies, is better handled with a MS implementation than with DD methods. Hence, the MS paradigm is a good candidate for small-scale LGT models with high computation-to-communication ratios. Finally, we discussed a promising future work direction by sketching an hybrid approach that combines the MS paradigm and DD methods for achieving scalable high performance.

References

[1] E. Babaev, A. Sudbø, and N. W. Ashcroft. A Superconductor to Superfluid Phase Transition in Liquid Metallic Hydrogen. *Nature*, 431:666, 2004.

[2] S. B. Baden and S. J. Fink. Communication overlap in multi-tier parallel algorithms. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–20, Washington, DC, USA, 1998. IEEE Computer Society.

[3] C. Banino-Rokkones, J. Amundsen, and E. Smørgrav. Parallelizing Lattice Gauge Theory Models on Commodity Clusters. In *2006 IEEE International Conference on Cluster Computing (CLUSTER 2006), September 25-28 2006, Barcelona, Spain*. IEEE Computer Society, 2006.

[4] O. Beaumont, V. Boudet, and A. Petitet. A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers). *IEEE Trans. Comput.*, 50(10):1052–1070, 2001.

[5] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix Multiplication on Heterogeneous Platforms. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1033–1051, 2001.

[6] O. Beaumont, A. Legrand, F. Rastello, and Y. Robert. Dense Linear Algebra Kernels on Heterogeneous Platforms: Redistribution Issues. *Parallel Comput.*, 28(2):155–185, 2002.

[7] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive Computing on the Grid Using AppLeS. *IEEE Trans. Parallel Distrib. Syst.*, 14(4):369–382, 2003.

[8] C. Ding and Y. He. A Ghost Cell Expansion Method for Reducing Communications in Solving PDE Problems. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 50–50, New York, NY, USA, 2001. ACM Press.

[9] J. Dongarra and A. Lastovetsky. An Overview of Heterogeneous High Performance and Grid Computing. In *Engineering The Grid: Status and Perspective*. American Scientific Publishers, 2006.

[10] A. Espinosa, T. Margalef, , and E. Luque. Automatic Performance Analysis of Master/Worker PVM Applications with Kpi. In *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 47–55, London, UK, 2000. Springer-Verlag.

[11] G. E. Fagg and J. J. Dongarra. Building and Using a Fault-Tolerant MPI Implementation. *Int. J. High Perform. Comput. Appl.*, 18(3):353–361, 2004.

[12] J.-P. Goux, S. Kulkarni, M. Yoder, and J. Linderoth. Master-worker: An enabling framework for applications on the computational grid. *Cluster Computing*, 4(1):63–70, 2001.

[13] F. Gutbrod, N. Attig, and M. Weber. The SU(2)-lattice gauge theory simulation code on the Intel Paragon supercomputer. *Parallel Comput.*, 22(3):443–463, 1996.

[14] T. Hagerup. Allocating Independent Tasks to Parallel Processors: An Experimental Study. *J. Parallel Distrib. Comput.*, 47(2):185–197, 1997.

[15] D. W. Heermann and A. N. Burkitt. *Parallel Algorithms in Computational Science*. Springer-Verlag New York, Inc., New York, NY, USA, 1991.

[16] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive Scheduling for Master-Worker Applications on the Computational Grid. In *GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 214–227, London, UK, 2000. Springer-Verlag.

[17] W. Janke and R. Villanova. Ising model on three-dimensional random lattices: A Monte Carlo study. *Physical Review B*, 66(13):134208–+, Oct. 2002.

[18] M. Kaddoura, S. Ranka, and A. Wang. Array Decompositions for Nonuniform Computational Environments. *J. Parallel Distrib. Comput.*, 36(2):91–105, 1996.

[19] A. Kalinov and A. Lastovetsky. Heterogeneous Distribution of Computations Solving Linear Algebra Problems on Networks of Heterogeneous Computers. *J. Parallel Distrib. Comput.*, 61(4):520–535, 2001.

[20] T. Kindberg, A. Sahiner, and Y. Paker. Adaptive Parallelism under Equus. In *Proceedings of 2nd International Workshop on Configurable Distributed Systems*, pages 172–182. IEEE, March 1994.

[21] M. Luscher. Solution of the Dirac equation in lattice QCD using a domain decomposition method. *Comput. Phys. Commun.*, 156:209–220, 2004.

[22] A. Mahanti and D. L. Eager. Adaptive Data Parallel Computing on Workstation Clusters. *J. Parallel Distrib. Comput.*, 64(11):1241–1255, 2004.

[23] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[24] P. D. Michailidis and K. G. Margaritis. Performance Evaluation of Load Balancing Strategies for Approximate String Matching Application on an MPI Cluster of Heterogeneous Workstations. *Journal of Future Generation Computing Systems*, 19(7):1075–1104, 2003.

[25] A. Morajko, E. César, P. Caymes-Scutari, T. Margalef, J. Sorribes, and E. Luque. Automatic Tuning of Master/Worker Applications. In J. C. Cunha and P. D. Medeiros, editors, *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 95–103. Springer, 2005.

[26] M. Prieto, I. M. Llorente, and F. Tirado. Data Locality Exploitation in the Decomposition of Regular Domain Problems. *IEEE Trans. Parallel Distrib. Syst.*, 11(11):1141–1150, 2000.

[27] M. J. Quinn and P. J. Hatcher. On the Utility of Communication-Computation Overlap in Data-Parallel Programs. *J. Parallel Distrib. Comput*, 33(2):197–204, 1996.

[28] E. E. Santos, S. Feng, and J. M. Rickman. Efficient Parallel Algorithms for 2-Dimensional Ising Spin Models. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 135, Washington, DC, USA, 2002. IEEE Computer Society.

[29] E. E. Santos and G. Muthukrishnan. Efficient Simulation Based on Sweep Selection for 2-D and 3-D Ising Spin Models on Hierarchical Clusters. In *IPDPS*, page 229b, 2004.

[30] R. Wolski. Experiences with Predicting Resource Performance On-line in Computational Grid Settings. *SIGMETRICS Perform. Eval. Rev.*, 30(4):41–49, 2003.

9. Bibliography

Cyril Banino-Rokkones is a PhD student at the Department of Computer and Information Science of the Norwegian University of Science and Technology (NTNU). He is mainly interested in distributed computing in heterogeneous and dynamic environments, parallel algorithms for automatic load balancing, as well as combinatorial optimization. He is currently hired at the high-performance computing center at NTNU.