A Centralized and On-line Scheduling Solution to Dynamic MPI Programs

Márcia C. Cera, Nicolas Maillard and Philippe O. A. Navaux
Universidade Federal do Rio Grande do Sul
Instituto de Informática
{marcia.cera, nicolas, navaux}@inf.ufrgs.br

Abstract

MPI (Message Passing Interface) is a communication library that is a standard to parallel programming in distributed memory architectures. After the specification of MPI-2 norm extending the MPI-1 norm, MPI was nearer to PVM, making possible to implement MPMD applications. Amongst the new features of MPI-2 there is the possibility to dynamically create processes during the application execution. This feature allows to implement dynamic applications (which tasks are available on-the-fly) and use dynamic resources (which become available during the application execution). In a dynamic context, an efficient scheduling strategy is crucial to get a good applications performance. This paper presents a simple centralized and on-line solution to schedule dynamic MPI applications, and further, the use of dynamic resources through MPI-2 features.

1. Introduction

MPI (Message Passing Interface) (10) specification was developed to be a standard used to programming parallel distributed architectures. MPI is a communication library that specify an interface to implement communication among distributed parallel processes. The library become a standard to High Performance Computing (HPC) used to solve important scientific problems – Grand Challenge Problem (12). Originally, the MPI norm follows only the SPMD (*Single Program Multiple Data*) programming model. MPI-2 norm (11) extends the original MPI norm and define an interface to dynamic processes creation, one-sided communication, parallel I/O, and other extensions. These new features allow to program into MPMD (*Multiple Programs Multiple Data*) model with dynamic processes creation adding flexibility to MPI applications. With these new features, MPI was nearer to PVM (Parallel Virtual Machine) (6), which is one of the precursors programming environment to attend dynamicity and heterogeneity issues to distributed memory machines. The new features are been included in the lastest MPI distributions *e.g.* LAM/MPI (16), MPICH-2 (9), OpenMPI (8).

Using MPI-2 dynamic processes creation is possible to program dynamic parallel application. In these applications, tasks are created and processed during the application execution without any previous knowledge about them. To schedule these applications, the decisions need to be taken on-line, *i.e.*, as tasks become available. A possibility to programming dynamic application with MPI is, as tasks are created, spawn dynamic processes to execute them. However, using the dynamic processes creation, the MPI-2 norm imposes some restrictions for communication among processes. In a straight way, the communication is able just between a process that spawn another ones (the father process) and their spawned processes (children processes). This restriction will affect the policy employed to schedule tasks among processes as will be shown in this paper.

Beyond the implementation of dynamic MPI applications, the dynamic processes creation allow use resources that become available during the application execution. In dynamic environments a MPI application can identify new resources available and spawn processes to use them. In this paper we will show how to the application can adapt to use resources that become available on-the-fly.

The aim of this paper is shown a proposal to schedule parallel dynamic application with MPI. It is a simple centralized and on-line strategy which has been shown efficient to schedule the applications target (5). It is well-known that a centralized solution brings bottleneck problems. But this solution is efficient in environments with few resources and make possible identify problems in the use of dynamic processes creation, attending the paper objectives.

The paper is structured as follow: Section 2 presents some related works and some perspectives to use dynamic processes creation in scheduling strategies. Section 3 shows the proposed scheduling solution to MPI dynamic applications. After known how applications will be schedule, Sec. 4 presents some tests made to prove the scheduling efficiency. Finally, Sec. 5 shows the conclusions of this paper and some future works.

2. Related Works

An important question about MPI-2 is why use dynamic processes creation in MPI applications. The answer can be thought in two aspects: resource platform and programming model. In the first, the dynamic processes creation can help giving some workload to resources that become available during the application execution. For programming model, dynamic processes creation allow to implement dynamic applications, in which the parallel algorithm get a better performance creating more processes during execution time.

A dynamic programming model was proposed by Cilk (1), in which programs can be viewed as a directed acyclic graph (DAG) and consist of a collection of Cilk procedures broken into a sequence of threads (vertices of the DAG). The Cilk model was a precursor and got relevant results using a Work Stealing scheduling (2) to efficiently distribute work into shared-memory architectures. In the Java context, Satin (17) was inspired by Cilk and offer a programming model based in Java threads with Cilk-like primitives for Divide and Conquer (*D&C*) programs on distributed-memory systems. Satin map efficiently the dynamic processes using a Work Stealing scheduling design especially to hierarchical wide-area clusters (local steals happen synchronously while a remote steal was processing asynchronously). Due the MPI-2 features, a Cilk-like programming model can be employed to MPI programs, as will be show in this paper.

Below, there are some key points related to the use of dynamic processes creation and some related works.

Program model: To get an efficient scheduling strategy to MPI programs becomes necessary establish the target program model. In general, the model adopted is Bag-of-Tasks (BoT) because it is simple and need few communications due the task independence (15; 7; 3). This paper is focused in a model more complicated in which tasks are known in execution time, like Cilk model. In this context, the dynamic processes creation will be used to attend the program model.

Dynamic resources: Here we will show two related works aiming at adapting the application to changes in the CPU availability. The first one is Adaptive-MPI (15) that is implemented over FT-MPI (18) (MPI-1 norm) and allows to add (to use resources that become available) and remove (to fault tolerance) resources dynamically. Adaptive-MPI automatically spawn new processes (to `add_resource` event) and dynamically adjust the communicator to account arriving or departing nodes. The adjustment of communicator involves a global processes stop, a synchronization of them to establish a new configuration and a restart of all processes. However, this solution was implemented in the MPI-1 norm context with hard implementations issue, like communicator adjust and synchronizations.

Considering the MPI-2 features, there is a system that dynamically reschedule running processes via automatic decision-making and processes migration (7). The system is implemented on top of MPI-2 (LAM/MPI distribution) and HPCM (High Performance Computing Mobility) middleware to offer heterogeneous processes migration. To migrate, a process spawn another one in destination machine and after transfers the execution, memory, and communication states to the new one. In system evaluation, was detected an overhead of the reschedule operation usually less that 4%, and a reduction in the application (BoT) execution time of almost 34%, that are considered a satisfactory result (7).

In this paper, the use of dynamic resources is done by dynamic applications tasks, *i.e.,* as the resources become available, dynamic tasks are allocated to them.

Load Balance: dynamic processes creation can be used to implement scheduling decision aiming to get load balance. An example is presented in (3) that describes a low intrusion implementation of a hybrid scheduling strategy designed to cope with the dynamic behavior of grid environments. This scheduling strategy is part of the EasyGrid (4) middleware that is a hierarchical distributed application management system embedded into MPI applications to facilitate their execution in the grid. The system is implemented upon LAM/MPI and use the MPI-2 dynamic process creation feature to implement the scheduling events. The hybrid scheduling adopted by EasyGrid combine static (to initial processes distribution) and dynamic (to rescheduling) scheduling heuristics. The dynamic scheduling occur in three levels: global, site, and host. In the *host* level is determined the order and instant that a process should be created based in statisticians calculated on-the-fly. *Site* level verifies how much the site is imbalance and if it is high, activate the re-distribution policy (scheduling event). Finally, *global* level verifies if the global application is imbalance, and also, when this imbalance statistic is high, it triggers a scheduling event.

This work presents some important aspects like: the use of an hierarchical scheduling when is intended to use large scale systems and the use of dynamic processes creation to get load balance on-line. In the same context of this related work, can see the on-line scheduling below.

On-line scheduling: the on-line scheduling make possible repair load imbalances during the application execution. It is a powerful option to bypass the effects of dynamicity and heterogeneity of actual parallel architectures, as was shown above. In the context of this paper, the on-line strategy is used to obtain load balance, and also to schedule dynamic applications once tasks became available on-the-fly.

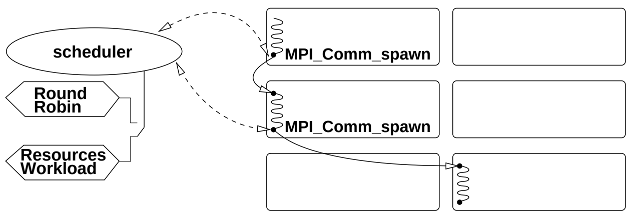

Figure 1. The proposed scheduler and the MPI-2 processes

3. A Simple Scheduling Solution

This section presents a quick introduction to a simple scheduler proposed to schedule MPI-2 parallel dynamic application (more informations look at (5)). The scheduler is centralized and simply determines, on-line, the physical location of new processes. Physical location is determinated following one of two policy: Round-Robin (standard) or based on resources workload. To make easy the use of the proposed scheduler, it was integrated inside a MPI distribution (in this case, LAM/MPI distribution). In this way, an application implemented with dynamic processes creation can be schedule by the proposed scheduler without any change in source files.

The scheduler architecture is simple and consists in a daemon (scheduler) running in one of the resources available to an application and it will be responsable to take scheduling decision on-line. In this case, the scheduling decision consists in determine physical location to new processes following the policy chose by the user. For each dynamic processes creation, the scheduler is requested and it will answer where is the best physical location to a new process.

Figure 1 shows the interaction between scheduler and MPI-2 processes when they create new processes dynamically. Dotted arrows represent the interactions between scheduler and MPI-2 processes and the solid arrow mean the process creation. The scheduling strategy chose in figure is the based on resources workload. The scheduler architecture is simple and allows that new policy are added and support the dynamic resources. The next section shows the efficiency obtained by the proposed scheduler and some important issues about it.

4. Experimental Results

This section shows some experimental results of the proposed schedule with dynamic MPI applications. Section 4.1 starts the experimental results and shows the overhead caused by the use of dynamic processes creation to MPI applications. In a second moment, Sec. 4.2 presents how is possible to use the dynamic processes creation together with dynamic resources and improve the application performance. At end, Sec. 4.3 shows the impact of workload-based policy in performance of irregular applications.

4.1. Dynamic Processes Creation Overhead

To identify the overhead caused by the dynamic processes creation, this section present a comparison between an application using MPI-1 and MPI-2. To make it possible, an BoT application was developed in a MPI-1 context (all processes created in the beginning of application) and in MPI-2 one (using dynamic processes creation).

The problem target is the generation of a fractal, the Mandelbrot Set, that is one of the thirteen Cowichan Problems (19) designed to test parallel programming systems. Mandelbrot Set is a classical embarrassingly parallel problem, in which the calculation of a particular data element is completely independent of the calculation of the other elements. The calculation is the iteration of the equations

$$x' = x^2 - y^2 + y_0$$
$$y' = 2xy + x_0$$

to a given initial coordinates (x_0, y_0) until an iteration limit is reached or the value diverge. In the paper test, the initial coordinates are $(0.5, 0.5)$ and the limit is 30500 iterations. To calculate the fractal, it was divided in blocks with dimension of 40×40 pixels. Each task represent a fractal block composed to (x, y) coordinates and the dimension of the block, which is always the same to all applications versions.

To identify the overhead of MPI-2 dynamic processes creation, it was developed three versions of Mandelbrot application. In the first version, a *master* manager the creation of *workers* in resources available (using `MPI_Comm_spawn`) while there are tasks to execute. Each *worker* receive one task, calculate it, send back to *master* a matrix with fractal points, and finalize its execution. Second Mandelbrot application use MPI-1 which all processes are created at the beginning of the execution, and a *master* send tasks to *workers* and receive the fractal points. To leave the MPI-1 and MPI-2 versions nearer, it was develop another application version. In the third one, a *master* create, dynamically, one *worker* by processor available and they will receive tasks and return the fractal points until exist tasks to execute. To MPI-2 implementations, it was only used the Round-Robin scheduling policy. To Mandelbrot Set application a workload-based policy did not take advantages to performance.

Figure 2 shows a graph with the time of execution by number of workers used to the three applications versions. Each value shown is the average of 15 executions of application to each number of *worker* and the standard deviation

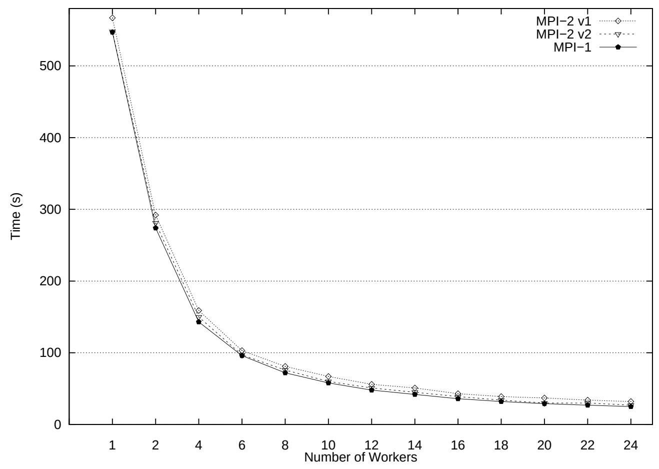

Figure 2. Timing for Mandelbrot Set computation using MPI-1 (solid line) and two version with MPI-2 (dotted lines)

is 1.77 in the worst case. The solid line represent the MPI-1 version and it get the best performance. The worst performance was bring by the MPI-2 version that create one *worker* by task executed, and it was in average approximately 14% worse than MPI-1 one. At last, the second MPI-2 version that is nearer to MPI-1 pattern get a performance, in average, approximately 5% worse than MPI-1. This last result shows that the dynamic processes creation imposes a low overhead to MPI applications, which can be neglected in dynamic applications context.

4.2. Using Dynamic Resources

This section presents how dynamic processes creation can help to use dynamic resources. The application used in tests is a solution to N-Queens problem. This problem consists in placing N queens on a $N \times N$ chessboard, in such a way that no queen may capture any other. That is to say, one has to find all the board configurations in which there exists at most one queen in a given row, column or diagonal. Although direct applications of the N-Queens problem are limited, this problem is often used as a benchmark because it represents a large class of problems, known as Constraint Satisfaction Problems (CSPs) (13).

The standard backtracking algorithm used to solve the N-Queens problem consists in placing recursively and exhaustively the queens, row by row. With MPI, each placement consists in a new spawned task. The algorithm backtracks whenever a developed configuration contains two queens that threaten each other, until all the possibilities have been considered. A maximum depth is defined, in order to bound the depth of the recursive calls.

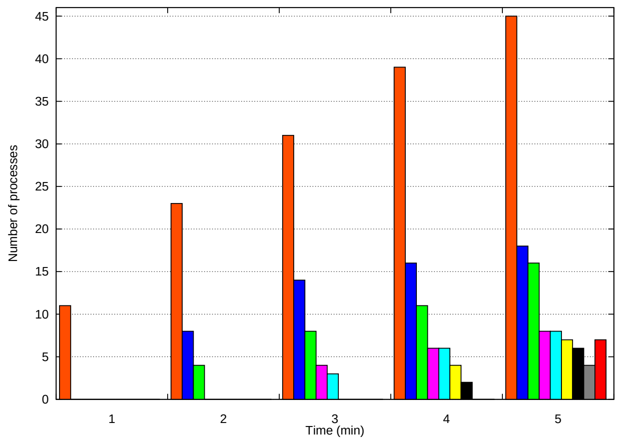

Figure 3. Number of processes by time during an execution of N-Queens with MPI

Figure 3 is a graph of number of processes by execution time of N-Queens using MPI-2. To get dynamic resources, at each minute a new processor is added to the computing environment. To add a new processing node (made of 2 CPU), LAM's `lamgrow` feature has been used. After the inclusion of a resource in the environment, the application detect this inclusion and adapts itself by spawning new processes on the new processors.

The bar diagram shows how many processes are run on each CPU, as a function of the time: at the beginning, all processes are running on the only node available. During the first minute, only one bar appears, which shows that all the 11 processes are running on one node. Each time that a 2-CPUs node turns available, the N-Queens programs adapts itself by spawning new processes on the newly available CPUs. At the end of the computation, all the CPUs of the 5 nodes are running processes of the N-Queens computation. Since no migration if allowed, the initial nodes run, until the end, more processes than those that were added later on.

4.3. Impact of Workload-based Politic

As mentioned in Sec. 3 the proposed scheduler offers two scheduling policy: Round-Robin (results presented in Sec. 4.1) and resources workload-based. The workload-based policy is appropriated to schedule irregular applications (which tasks are processed with variable duration) or in heterogeneous environment. This section shows the behavior of an irregular application schedule with two policy options.

The irregular application used in the tests is a Primality Computation. In this application, the number of prime numbers in a given interval (between 1 and N) is computed by recursive search. To programme a recursive search, a new process is spawned (`MPI_Comm_spawn`) for each recursive subdivision of the interval. Due to the irregular distribution of prime numbers and irregular effort to test a single number, the parallel program is natively unbalanced.

Figure 4 shows the run-times by the workload, as measured by the size N of the interval, with the Round-Robin and workload-based policy. In the latter case, the proposed scheduler, transparently, request to a resource manager in order to obtain on-line information about the load of the processors. The Round-Robin algorithm maintains a natural load balance between the processors, and the workload-based grants that any under-loaded processor executes more processes. Thus, in both cases, the resources are better employed.

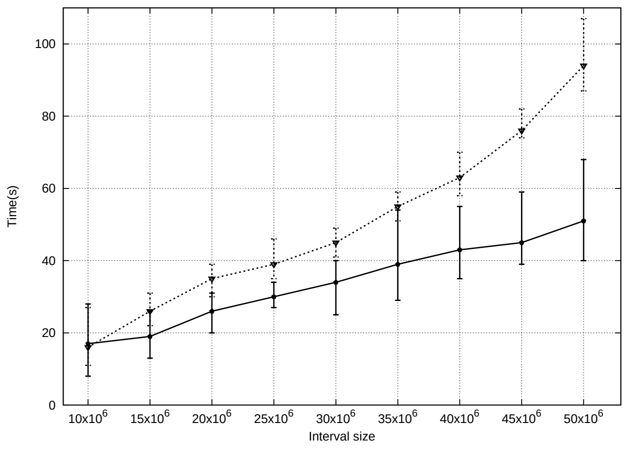

Figure 4. Timings for the Prime computation, with Round-Robin (dotted line) and workload-based (solid line) policy (5)

As can be seen, in the case of this irregular computation, the use of the on-line scheduler with load information enables a consistently better run-time than with Round-Robin, even when the statistical fluctuations are taken into account. Also, it can be seen that the time lasted when used the Round-Robin scheduling increases steady against a smoother increase when using the resource workload-based.

5. Conclusions

MPI-2 extension are highly promising, once allow use dynamic resources and make possible to implement dynamic applications. This paper shows a simple scheduler to MPI dynamic applications. This scheduler is centralized and take scheduling decision on-line. The results show that there is a low overhead using dynamic processes creation, which can be neglected in dynamic environment context. The experimental results shown how to take advantage of dynamic processes creation to use dynamic resources and the impact of a workload-base policy in the scheduling of irregular applications.

The results presented in this paper concentrate the main aspects of programming using dynamic processes creation. To get an efficient program that taking advantage of MPI-2 features, it is necessary to bypass some restrictions of MPI. For instance, the communication restriction that will directly affect the schedule decisions.

In this paper, the dynamicity of the environment was restricted to the inclusion of a new resource, more specifically, the use of `lamgrow` LAM/MPI primitive. The LAM/MPI offer a primitive, called `lamshrink`, to exclude dynamically one node of resources set. But to application execution safety, the usage of this primitive needs the implementation of some fault tolerance aspects, like checkpoints mechanisms, inside of application. Fault tolerance issues was out of scope of this work and will be left to future workers.

Together to the development of the proposed scheduler, it was studied how to program Divide and Conquer (*D&C*) applications using MPI-2 (14). *D&C* model is natively dynamic and can be efficiently schedule by a Work Stealing strategy. In this context, there was developed a Hierarchical Work Stealing algorithm to MPI-2 application. This work shows that the MPI-2 features open possibilities to explore important aspects in a new range of MPI applications.

References

[1] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. C. E. Zhou. Cilk: an efficient multithreaded runtime system. *ACM SIGPLAN Notices*, 30(8):207–216, Aug. 1995.

[2] R. D. Blumofe and C. E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 27(1):202–229, 1998.

[3] C. Boeres, A. P. Nascimento, V. E. F. Rebello, and A. C. Sena. Efficient hierarchical self-scheduling for mpi applications executing in computational grids. In *MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing*, pages 1–6, New York, NY, USA, 2005. ACM Press.

[4] C. Boeres and V. E. F. Rebello. Easygrid: towards a framework for the automatic grid enabling of legacy mpi applications. *Concurrency - Practice and Experience*, 16(5):425–432, 2004.

[5] M. C. Cera, G. P. Pezzi, E. N. Mathias, N. Maillard, and P. O. A. Navaux. Improving the dynamic creation of processes in MPI-2. In *Lecture Notes in Computer*

Science - 13th European PVMMPI Users Group Meeting, volume 4192/2006, pages 247–255, Bonn, Germany, 2006. Springer Berlin / Heidelberg.

[6] J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sundaram. Integrated PVM framework supports heterogeneous network computing. *Computers in Physics*, 7(2):166–175, 1993.

[7] C. Du, S. Ghosh, S. Shankar, and X.-H. Sun. A runtime system for autonomic rescheduling of mpi programs. In *33rd International Conference on Parallel Processing (ICPP 2004)*, pages 4–11, Montreal, Quebec, Canada, 2004. IEEE Computer Society.

[8] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting*, volume 3241 of *Lecture Notes in Computer Science*, pages 97–104, Budapest, Hungary, 2004. Springer.

[9] W. Gropp. MPICH2: A new start for MPI implementations. *Lecture Notes in Computer Science*, 2474:37–42, 2002.

[10] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, Cambridge, Massachusetts, USA, Oct. 1994.

[11] W. Gropp, E. Lusk, and R. Thakur. *Using MPI-2 Advanced Features of the Message-Passing Interface*. The MIT Press, Cambridge, Massachusetts, USA, 1999.

[12] J. L. Gustafson. A paradigm for grand challenge performance evaluation. In R. K. Kalia and P. Vashishta, editors, *Toward Teraflop Computing and New Grand Challenge Applications*, pages 279–290? Nova Science Publishers, Commack, New York, 1995.

[13] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[14] G. P. Pezzi, M. C. Cera, E. N. Mathias, N. Maillard, and P. O. A. Navaux. Escalonamento dinmico de programas mpi-2 utilizando diviso e conquista. In *VII Workshop em Sistemas Computacionais de Alto Desempenho*, pages 71–79, Ouro Preto, Brazil, Oct. 2006.

[15] L. A. Rao and J. Weissman. Mpi-based adaptive parallel grid services. Technical report, Minneapolis, MN, USA, 2003.

[16] J. M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September / October 2003. Springer-Verlag.

[17] R. V. van Nieuwpoort, T. Kielmann, and H. E. Bal. Satin: Efficient Parallel Divide-and-Conquer in Java. In *Euro-Par 2000 Parallel Processing*, number 1900 in Lecture Notes in Computer Science, pages 690–699, Munich, Germany, Aug. 2000. Springer.

[18] J. B. Weissman, L. R. Abburi, and D. England. Integrated scheduling: the best of both worlds. *Journal of Parallel and Distributed Computing*, 63(6):649–668, 2003.

[19] G. Wilson. Assessing the usability of parallel programming systems : The cowichan problems. In *IFIP Working Conference on Programming Environments for Massively Parallel Distributed Systems*, pages 183–193, 1994.