Controlling Task Granularity of Dynamic MPI Programs with threads

João Vicente Lima, Nicolas Maillard
Instituto de Informática – Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
{joao.lima, nicolas}@inf.ufrgs.br

Abstract

Granularity control is an important requirement on parallel programming performance. Regular problems can archive this with suitable data distributions or mappings to processors, whereas on irregular problems the imprecision about workload until execution difficult this approach. Irregular problems solve imprecision through a recursive-like programming model like Divide and Conquer (D&C) but it depends of a tool that supports dynamic task creation. Some tools like Cilk and KAAPI offer dynamism and granularity control, however both have little usage in HPC. MPI, which is a standard in HPC, added dynamic process creation and multithreaded execution but lacks of implementation issues. This work presents the advantages of task granularity control with threads in MPI dynamic programs. A mechanism (libSpawn) changes the default process creation for tasks, and decides when create processes or threads. Our experiments demonstrate that libSpawn offers a good performance improvement near to 85% with Cilksort, which is a sorting algorithm based on D&C model.

1. Introduction

Granularity is an important requirement on parallel programming performance. Regular problems can archive this with suitable data distributions or mappings of task to processors. Irregular problems, however, can not efficiently control the load of each task because of irregular communication patterns and load imbalances. One way to solve this uses a recursive-like programming model, as Divide and Conquer (D&C) strategy. D&C consists on partitioning a problem recursively into smaller sub-problems, until their solution become trivial. A parallel D&C program needs a tool which has support to dynamic task creation and synchronization.

Tools that archive this approach offer dynamism and granularity control through the concept of abstract task, which can be a process or a thread. Cilk [2] and KAAPI [11] provide granularity control at compile time or run-time, as well as extend languages to provide task creation and synchronization. Despite these features, both are either limited to shared memory systems or little used in high performance computing (HPC). MPI, which is a standard in HPC, added dynamic process creation and multithreaded execution at its second version [7].Yet, the standard lacks of implementation issues as task definition and management.

This paper presents the advantages in task granularity control with threads on MPI dynamic programs. A mechanism named libSpawn changes default process creation for tasks, and decides when create a process or a thread. This mechanism spawns POSIX threads until a limit, which means its granularity. After this limit, libSpawn creates tasks as processes. We evaluate libSpawn against a sorting algorithm based on D&C model named Cilksort, originally implemented in Cilk. Our experiments demonstrate that the mechanism offers a good performance improvement near to 85%. The goals of libSpawn appear at task creation and message passing cost reduction.

The remainder of this paper is structured as follows. In section 2 we present the context on irregular and dynamic problems, then section 3 discusses the state of the art on tools for dynamic programming. In the next section, our mechanism is detailed and section 5 shows the results obtained. Finally, section 6 offers some concluding remarks and plans for the future.

2. Dynamism and Parallel Irregular Applications

Irregular problems mean that the load of tasks is usually not known until execution time [6]. This imprecision can be related between two main factors: computing platform and algorithm behavior. An irregular computing platform is an heterogeneous environment where shared resources and hardware configurations influence program performance. For example, Grid, NOW (*Networks of Workstations*), and clusters are computing platforms with heterogeneous resources and sometimes shared [9, 1, 3]. Parallel applications can address irregular platforms through load bal-

ancing or resource allocation on demand. These methods depend of a programming tool which offers primitives to express dynamism and synchronization among tasks.

Irregular algorithms present variations in its computational effort which may be input dependent or may involve the computation itself [15]. The characteristics of which algorithms involve:

- input data can be irregular, like sparse matrices;
- internal structure can be dynamically growing or shrinking during runtime;
- locality of data dependencies.

The D&C programming model works with irregular data inputs. Moreover, a program implementation must depends of a programming tool for dynamic task creation on demand. Such dynamism comes from tasks created to resolve subproblems of the total problem in parallel. The programming tools with dynamism are presented in the next section.

3. Tools for Dynamic Programming

The tools for dynamic programming can be specific to SMP (*Symmetric multiprocessor*) architectures or clusters. Cilk [2] is a C-based run-time system for SMP architectures with 3 keywords to express parallelism and synchronization. Cilk describes tasks as user threads which have asynchronous execution and Work-Stealing scheduling. Data dependencies and available processors are the granularity parameters to Cilk scheduler. The efficiency of Cilk appears in other works [2, 10], but it lacks of a efficient implementation for distributed architectures.

KAAPI [11] is a programming tool to SMP and multicore clusters based on concept of global memory and data dependencies among tasks. The language extends C++ with 2 keywords for express parallelism and synchronization. KAAPI tasks are user level threads mapped to virtual processors (VT's), and the VT's are kernel threads assigned to processors available for execution. Its scheduler follows the Work-Stealing method as Cilk and its granularity depends on idle VT's and data dependencies. KAAPI reaches efficient and stable results [11, 5], however its programming effort does not appear on previous works.

MPI (*Message-Passing Interface*) [7] is a recognized standard for HPC programming in distributed memory environments. The MPI-2 version added new features like dynamic process creation and multithreaded programming. Its task creation is specified as asynchronous and collective in cases of two or more processes. Whereas, MPI lacks of implementation issues about task creation and management. Other recent works proposed improvements on dynamic programs for load balancing like a scheduler module [4] and an Hierarchical Work-Stealing scheduler algorithm [13].

The description of multithreaded MPI programs offers a new level of parallelism on shared memory systems, mainly on SMP multicore architectures. An user program must call `MPI_Thread_init` to indicate the level of thread support desired, and the MPI implementation will return the level it supports. The standard specifies four levels of thread-safety and the `MPI_THREAD_MULTIPLE` level means that when multiple threads make MPI calls concurrently, the outcome will be as if the calls executed sequentially in any order. Also, MPI says that is user responsibility to prevent races when threads within the same program post conflicting communication calls. Some researchers have been studied thread-safety issues [12, 16] on MPI implementations.

4. libSpawn: Controlling Task Granularity with threads on MPI

Granularity represents an important requirement to good performance results in parallel programs. Besides, this requirement is relevant to irregular problems because of data dependencies and task creation costs [8]. On MPI-2 context, granularity can be controlled through creating tasks as threads or processes. This approach offers significant reductions on task creation and communication. Thus, this paper evaluates benefits in controlling granularity using threads in dynamic MPI programs.

libSpawn overloads 10 MPI calls which act on task creation and communication. This mechanism lets a process create some threads until a limit defined at compilation time. The begin of process creation happens when this limit is reached. As well as task creation, libSpawn handles message communications among threads because MPI does not identify threads within a communicator.

4.1. Granularity Control

The procedure `MPI_Comm_spawn` contains the mechanism of granularity control where two tests decide between create a thread or a process. The first matches the command name against the program name on execution, and if them are different a new process will be spawn. In the next test, if the number of execution threads is greater than the limit established a process will be spawn. Otherwise, a POSIX thread will be created.

A compile time macro `MAX_THREADS` defines the thread limit of a process. This is tested on `MPI_Comm_spawn` procedure with a global variable which counts the number of execution threads in a process.

4.2. Message-Passing between threads

libSpawn controls message passing between threads through `MPI_Send` and `MPI_Recv` calls. Two lists of

send and receive messages maintain requests to be finished with mutual exclusion that prevents race conditions.

Requests also carry information that can be used to distinguish messages. This information consist of a fixed number of fields named **message envelope**, as MPI standard defines[7]. These fields are:

- *source* - message sender
- *destination* - message destination
- *tag* - identifier of message
- *communicator* - communicator used

The library creates a thread on its initialization to find and complete matching requests on each `MPI_Send` and `MPI_Recv` call. This thread searches for requests according to message envelope and completes them through a memory copy operation. The completion thread also signals threads involved in the communication, so they can execute again.

5. Results

5.1. Parallel Sorting with Cilksort

Cilksort is a parallel sorting algorithm that follows the D&C programming model [10]. This algorithm divides an input recursively in 4 parts and spawns tasks to sort in parallel. When tasks reach a limit of elements, a sequential sort is applied. The merge phase performs union of sorted parts in parallel like sorting but divides its input in 2 parts.

The MPI version of Cilksort does sorting in same way but limits the synchronization phase. The D&C model permits restrict communications between created tasks (child tasks) and its parent (parent task) [14]. Consequently, MPI sorting has two synchronizations: input send to child tasks and return result to parent. Sending input has two communication calls that send input size and elements respectively. At return result, child tasks send the sorted parts to parent with one communication per child.

Furthermore, Cilk parallelism is MIMD (*Multiple Instruction Multiple Data*) while MPI is MPMD (*Multiple Programs Multiple Data*). So, Cilksort in MPI is composed of 3 programs: **cilksort-start**, **cilksort-sort** and **cilksort-merge**. **cilksort-start** reads input elements from a file and begins the algorithm spawning **cilksort-sort**. **cilksort-sort** and **cilksort-merge** contains sort and merge functions, as well as synchronization operations. The sequential sorting begins with 2.048 elements in each case. The correspondent algorithms are *quicksort* for sort phrase and *mergesort* for merge phrase.

5.2. Performance Evaluation in Function of Granularity

Figure 1 shows Cilksort results and granularity configurations using libSpawn. We evaluated 3 inputs of 3 million elements in random order, hence generating 13.313 tasks. All tests are done on 10 nodes of Grid5000 with 2 Dual Core processors and Gigabit Ethernet network. Each measure is a mean of 20 executions.

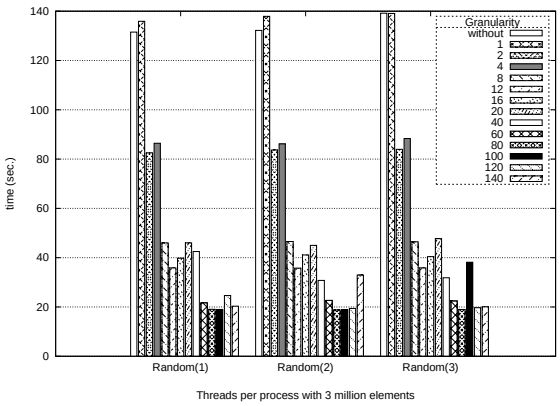

Figure 1. Results obtained from Cilksort with 3 million elements

The configuration with 1 thread shows the worst case because of library costs on creating tasks as processes. Whereas, 2 threads configuration already shows a significant gain of 37%, while a granularity of 100 threads represents the best time around 85% of gain. This performance improvement has archived from cost reduction on task creation and communication. A characteristic of D&C algorithm contributes to our gains since the first tasks transfer bigger messages on sending inputs and returning results.

Measures for 1, 4, and 20 threads evidence cost management of libSpawn when process creation increases, even though last two cases bring some cost reduction. The 2 threads configuration leads to process creation on sorting phase, since program wants create 4 tasks which is greater than the granularity limit. In that configuration, merge phase spawns threads and processes improving performance gain. A granularity of 4 threads leads sorting phase spawns 4 tasks as threads on first recursive call, however on next call the tasks will be processes. Thus, a increase in process creation reduce libSpawn efficiency. Also, the figure shows other characteristic which is a overhead when granularity increases. Results obtained beyond 80 threads are equal or greater than previous because of substantial memory contention.

Measures considered worst case when elements are in decreasing order, and best case when elements are in increasing order. These inputs generated 7.154 tasks and results were similar as presented about random elements, which the lower time was $14,28$ sec. with standard deviation of $1,18$.

6. Conclusion

The imprecision about workload until execution time characterizes irregular problems, which are related to a heterogeneous platform or irregular algorithm. Implementations of these problems archive this imprecision through task creation on demand, or dynamically, that difficult a appropriate granularity control. A tool which supports this control and dynamic task creation is essential to efficient implementations.

Tools like Cilk and KAAPI offer dynamism, but your limitations difficult its usage in HPC. MPI, which is a standard in HPC, has dynamic process creation and multithreaded support on MPI-2 version but it lacks on task implementation issues. This article has presented the advantages on granularity control with threads in MPI dynamic programs, through a mechanism named libSpawn that decides when create threads or processes. Its implementation allows a process spawns many threads per process.

Our evaluation used Cilksort algorithm that is based on D&C programming model. Measures show significant gains about 85% against results without the mechanism, thanks to task creation and communication optimizations. Also, the library overhead influenced fine-grain configurations where granularity is greater than 100 threads.

Future work on libSpawn is archive more MPI procedures like non-blocking and collective communications. Furthermore, we intend improve our mechanism through granularity decisions based on load measures at runtime.

References

[1] M. A. Bender and M. O. Rabin. Online Scheduling of Parallel Programs on Heterogeneous Systems with Applications to Cilk. *Theory of Computing Systems Special Issue on SPAA00*, 35:289–304, 2002.

[2] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An Efficient Multithreaded Runtime System. In *PPOPP '95: Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 207–216, New York, NY, USA, July 1995. ACM Press.

[3] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard. A batch scheduler with high level components. In *Cluster computing and Grid 2005 (CCGrid05)*, 2005.

[4] 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. *Lecture Notes in Computer Science - 13h European PVM/MPI Users Group Meeting*, 4192/2006:247–255, Sept. 2006.

[5] V. Danjean, R. Gillard, S. Guelton, J.-L. Roch, and T. Roche. Adaptive Loops with Kaapi on Multicore and Grid: Applications in Symmetric Cryptography. In *PASCO '07: Proceedings of the 2007 international workshop on Parallel symbolic computation*, pages 33–42, New York, NY, USA, 2007. ACM.

[6] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, editors. *Sourcebook of Paralllel Computing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[7] M. P. I. Forum. MPI-2: Extensions to the Message-Passing Interface. Technical Report CDA-9115428, University of Tennessee, Knoxville, Tennessee, July 1997.

[8] I. Foster. *Designing and Building Paralllel Programs*. Addison-Wesley, 1995.

[9] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[10] M. Frigo, C. E. Leiserson, and K. H. Randall. The Implementation of the Cilk-5 Multithreaded language. In *Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation*, pages 212–223, Montreal, Quebec, Canada, jun 1998.

[11] T. Gautier, X. Besseron, and L. Pigeon. KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In *PASCO '07: Proceedings of the 2007 international workshop on Parallel symbolic computation*. ACM, 2007.

[12] W. Gropp and R. Thakur. Thread-safety in an MPI implementation: Requirements and analysis. *Parallel Computing*, 33(9):595–604, Sept. 2007.

[13] G. P. Pezzi, M. C. Cera, E. Mathias, N. Maillard, and P. O. A. Navaux. On-line Scheduling of MPI-2 Programs with Hierarchical Work Stealing. *SBAC-PAD 2007: 19th International Symposium on Computer Architecture and High Performance Computing, 2007.*, pages 247–254, 2007.

[14] G. P. Pezzi, M. C. Cera, E. N. Mathias, N. Maillard, and P. O. A. Navaux. Escalonamento Dinâmico de programas MPI-2 utilizando Divisão e Conquista. *WSCAD'06 - Workshop em Sistemas Computacionais de Alto Desempenho*, 7:71–78, 2006.

[15] G. Rünger. Parallel Programming Models for Irregular Algorithms. *Lecture Notes in Computational Science and Engineering - Parallel Algorithms and Cluster Computing*, 52:3–23, 2006.

[16] R. Thakur and W. Gropp. Test Suite for Evaluating Performance of MPI Implementations That Support MPI_THREAD_MULTIPLE. *Lecture Notes in Computer Science - 14h European PVM/MPI Users Group Meeting*, 4757:46–55, Sept. 2007.