Preliminary Experiments with Work-First and Help-First Scheduling Policies in WORMS

Vinícius Garcia Pinto, Nicolas Maillard
Grupo de Processamento Paralelo e Distribuído (GPPD)
Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre - RS - Brasil
{vgpinto, nicolas}@inf.ufrgs.br

Abstract

This paper presents preliminary experiments with two scheduling policies in the work-stealing scheduler of WORMS *runtime system. We evaluate Work-First and Help-First policies with two micro-benchmarks: Fibonacci and Fork-Join. Our experiments achieved expected results with Fibonacci, but unexpected ones with Fork-Join.*

1. Introduction

Managing the increasing degree of concurrency is a major challenge in current and upcoming HPC systems. Task parallelism in combination with an efficient runtime system can assist programmers in exploiting all concurrency of current architectures [2, 1].

An appropriate scheduler is a key component of an efficient runtime system. In WORMS [7] we use a scheduler based in a work stealing algorithm. In this paper, we show preliminary results about the impact of work-first and help-first policies in the work stealing scheduler used by WORMS. The availability of both policies can provide better performance because the programmer is able to choose the most suitable policy to a certain pattern of parallelism.

The remainder of this paper is organized as follows: Section 2 presents an overview about WORMS runtime system. Section 3 presents two scheduling policies used in this work: work-first and help-first, and the changes required to support work-first in WORMS. Section 4 presents our experimental evaluation. Finally, Section 5 presents the conclusions and future works.

2. WORMS Runtime System

WORMS (WORk stealing scheduling for Multi-CPU/GPU Systems) is a runtime system which targets multi-CPU and multi-GPU architectures. WORMS supports task parallelism with dynamic task creation and fully strict dependencies between tasks. The runtime system allows two implementations for the same task: one for CPU execution and another for GPU execution. At execution time, WORMS decides which task implementation will be executed based on the available resources. Task scheduling is done dynamically by a work stealing algorithm.

In the WORMS programming model, each task is represented by an object with two virtual methods *runCPU()* and *runGPU()*. Parallelism is unfolded by methods to create new children tasks, e.g. *taskSub(new Task)*. There are also methods to force synchronization between tasks, allowing parent tasks to wait for their children.

3. Work-First and Help-First Scheduling Policies

The work-first principle is a set of design decisions adopted in Cilk-5 implementation to minimize the scheduling overhead [3]. In this work, we use the work-first and help-first definitions proposed by Guo et al. [5].

The work-first and help-first concepts refer to which part of the code is executed after a new task creation. In work-first policy, the worker[1] executes a created task and pushes a continuation of the current task to the queue. If a work-stealing scheduler is used, this continuation can be stolen by another worker or popped by the same worker when the new task's execution completes. In contrast, in help-first policy,

1 In WORMS, the workers are called *TaskProcessors*.

when a new task is created, the worker executes the continuation of the current task and pushes the new task to the queue. As in the work-first policy, a stacked task can be stolen by another worker.

In general, work-first is used in language approaches to task parallelism (e.g. Cilk [3]) while help-first is adopted in library approaches (e.g. XKaapi [4]).

3.1. Changes in WORMS to support Work-First

In previous versions of WORMS [7, 8], the task creation steps follow the help-first concept. In this model, new tasks are submitted to parallel execution using *taskSub* method. When *taskSub* is called, the new task (child) is pushed into the queue (a deque(ue) in this case), and a counter is incremented in the current task (parent).

In order to add support for the work-first concept, we create two specializations of the *taskSub* method: *taskSubHF* and *taskSubWF*.

The *taskSubHF* method has the same behavior of *taskSub* in the previous versions. The *taskSubWF* implements the work-first policy. So, when a new task is created, the state of the current task is saved, this continuation is pushed to top of the deque and the execution of the new task starts. When the continuation of a paused task is to be resumed (popped by the same worker or stolen by other), the state of this task is recovered and the execution continues in the next line after the line where *taskSubWF* was invoked. In this version of WORMS it is possible to use both specializations of *taskSub* inside the same task without restrictions.

We use symmetric coroutines data structures from Boost.Coroutine and Boost.Context libraries [6] to save and restore the state of a running task.

4. Experiments

In this section, we present initial experiments with help-first and work-first policies in WORMS. We use micro-benchmarks to identify the behavior and the limitations of both scheduling policies. In this case, micro-benchmarks are useful to assess the overhead introduced by the policies in the management of the tasks.

4.1. Platform

Our experimental results were obtained on a multicore system with two Intel Xeon E5-2630 hexa-core processors at 2.30GHz with 32 GB of RAM. This system runs Ubuntu Linux, version 12.04. WORMS and the micro-benchmarks were compiled with g++ 4.8.1 (with -O2 optimization flag) and Boost C++ Libraries 1.56. All experiments were executed with up to 12 threads.

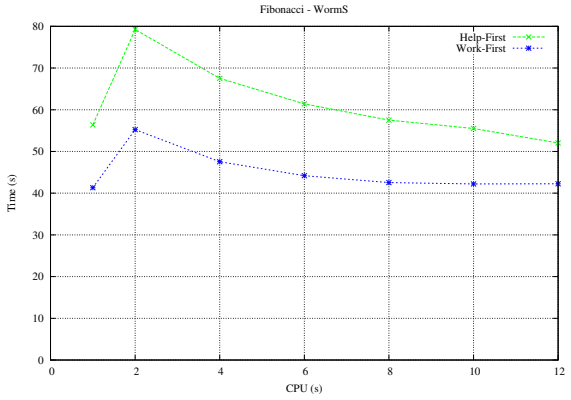

Figure 1. Help-first vs Work-first with Fibonacci micro-benchmark. (Smaller is better)

4.2. Micro-benchmarks

We use two micro-benchmarks: Fibonacci and Fork-Join. Fibonacci is a case of recursive parallelism while Fork-Join is a case of flat parallelism. In general, it is expected that work-first policy is better than help-first for recursive parallelism and that help-first is the best policy for flat parallelism. This is expected because the combination of help-first with recursive parallelism or work-first with flat parallelism increases the number of sequential steals (if there are many workers).

The Fibonacci micro-benchmark computes recursively the n-th number of Fibonacci sequence. In this work, we compute Fib(35) without sequential threshold.

Fork-join micro-benchmark creates iteratively a large number of tasks, and after waits for the end of all created tasks.

4.3. Results

In this section, we present results obtained with the Fibonacci and Fork-Join micro-benchmarks considering an average of thirty executions.

Figure 1 shows the results of the Fibonacci micro-benchmark with help-first and work-first policies. In this experiment, as expected because it is a case of recursive parallelism, the work-first policy outperforms help-first for all number of cores used. Note that the speed-up is not an important issue in this scenario because we are considering only the overhead introduced by both policies and not the real computation performed to find the element in the Fibonacci sequence.

Figure 2 shows the results of the Fork-Join micro-benchmark with help-first and work-first. In this test were created 1000000 dummy tasks. With Fork-Join

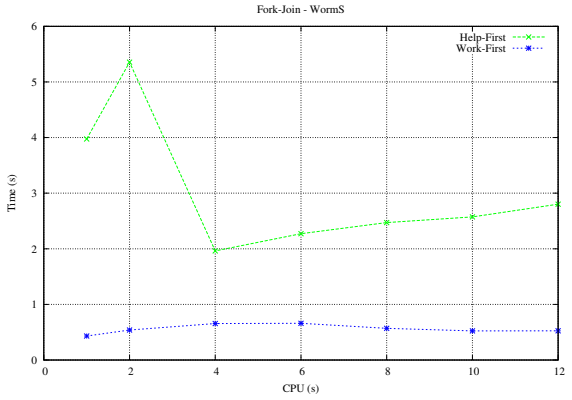

Figure 2. Help-first vs Work-first with Fork-Join micro-benchmark. (Smaller is better)

micro-benchmark, the work-first policy also outperformed help-first. However, for this case (flat parallelism), it was expected that help-first would achieve the best performance. We will investigate this unexpected behavior in future works.

5. Conclusions

In this paper, we presented initial results about the impact of work-first and help-first policies in the work stealing scheduler of the WORMS runtime system. We also presented an overview about the two policies and the changes made in WORMS to include support to work-first.

We evaluated both policies with two micro-benchmarks. For Fibonacci, as expected, work-first presented better performance than help-first. With Fork-Join, were obtained unexpected results for flat parallelism (work-first was better than help-first). We will study this behavior in next works. It is possible that this behavior is related to the cost to save the state of a running task or to the cost of synchronization operations over the queues.

As future work, we plan to extend our evaluation to other benchmarks and study the impact of these policies using not only CPUs but also GPUs.

References

[1] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick. A view of the parallel computing landscape. *Commun. ACM*, 52:56–67, 2009.

[2] K. Bergman, S. Borkar, and D. Campbell. Exascale computing study: Technology challenges in achieving exascale systems. Technical report, 2008.

[3] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. *Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation - PLDI '98*, pages 212–223, 1998.

[4] T. Gautier, J. Lima, N. Maillard, and B. Raffin. Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1299–1308, May 2013.

[5] Y. Guo, R. Barik, R. Raman, and V. Sarkar. Work-first and help-first scheduling policies for async-finish task parallelism. *2009 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 0:1–12, 2009.

[6] O. Kowalke. Boost C++ Libraries 1.56 - Boost.Coroutine, 2014. Available: http://www.boost.org/doc/libs/1_56_0/libs/coroutine/doc/html/.

[7] V. G. Pinto and N. Maillard. Work stealing on hybrid architectures. In *Computer Systems (WSCAD-SSC), Proceedings of 2012 13th Symposium on*, pages 17 –24, Los Alamitos, oct. 2012.

[8] V. G. Pinto and N. Maillard. Worms: Um ambiente de execuo para sistemas multi-cpu e multi-gpu. In *Anais da* 14^a *Escola Regional de Alto Desempenho*, volume 1 of *Anais da Escola Regional de Desempenho*, pages 131–132, Porto Alegre, 2014. Sociedade Brasileira de Computação.