Reconfigurable Sequential Consistency Algorithm

Christiane V. Pousa, Luís F. W. Góes, Dulcineia O. Da Penha and Carlos A. P. S. Martins
Graduation Program in Electrical Engineering – Computational and Digital Systems Group
Pontifical Catholic University of Minas Gerais - Belo Horizonte – Minas Gerais – Brazil
christ.bh@terra.com.br, lfwgoes@yahoo.com.br, dulcineia@pucmg.br, capsms@pucminas.br

Abstract

In this paper, we propose, implement and analyze the performance of a Reconfigurable Sequential Consistency Algorithm (RSCA) using simulation. Extending the concepts of reconfigurable devices to the algorithmic level, we model RSCA that is a reconfigurable sequential consistency algorithm for asynchronous distributed systems that manage concurrent objects stating. As our main results, we present that, on average, the performance of RSCA was 36% better than the traditional sequential consistency algorithms. The main contributions of this paper are: the definition, proposal, implementation and performance analysis of RSCA.

1. Introduction

In a distributed computing environment, when its application has concurrent access in several data, the users must use a program model that helps the management of concurrent accesses and simplifies the programming. A software DSM (Distributed Shared Memory) is an abstraction that provides an illusion of a shared memory in a distributed system. It manages a set of virtual memory pages or a set of objects, in which, the read and write operation semantics (Consistency Model) guarantee that objects will be consistent during the application execution.

Consistency models should match easy programming model and low application response time, in order to maximize the system’s performance for all workloads. The main problem is that workload, system architecture and consistency model semantics change continuously [1] [2] [3].

Reconfigurable computing emerged as a paradigm to fill in the gap between hardware and software, reaching better performance than software and more flexibility than hardware [4] [5] [7]. The reconfigurable devices, including FPGAs, contain an array of computing elements or constructive blocks, whose functionalities are determined through the programming of configuration bits. Thus, an FPGA can implement different behaviors not established at design time. Because of this, reconfigurable devices (hardware) are improving the solutions for problems from different areas [4] [5] [7].

Our basic idea in this paper is to extend the concepts of reconfigurable devices to a consistency algorithm to improve application and system performance.

The main objectives of this paper are: to define, propose, develop and implement the RCSA; to analyze the performance of RCSA using simulation.

2. Reconfigurable Algorithms

Nowadays, reconfigurable systems are composed of reconfigurable hardware (FPGAs etc.) and compilation environments that generate bitstreams that configure the reconfigurable hardware. Generally, the development of a reconfigurable application on a reconfigurable system first involves the partitioning of an algorithm into modules to be implemented on hardware, and modules to be implemented on software running on a general-purpose processor. The modules destined to the hardware are synthesized and mapped to the reconfigurable device. On the other hand, the other modules are processed by software, which can also control the run-time reconfigurations of the reconfigurable hardware [4].

In this context, both software and algorithm are not reconfigurable systems themselves. They are sequential and traditional systems that cannot change their structure at run-time. Compared to a parallel system, in which algorithm, software and hardware are parallel, nowadays, a reconfigurable system is composed only of a reconfigurable hardware.

This analogy makes it clearer that reconfigurable hardware has not being used at its full efficiency. Thus, we believe that reconfigurable software and algorithms are needed to achieve a higher efficiency in reconfigurable systems.
So, extending the concepts of reconfigurable devices (hardware), we define a reconfigurable algorithm as an algorithm that is composed of constructive blocks, allowing its behavior to be modified through the form of its configuration [13].

A reconfigurable algorithm is composed of three layers: Configuration Control Layer (CCL), Reconfigurable Layer (RL) and Basic Layer (BL), as shown in Fig.1. The BL consists of a frame set and data structures. A data structure may be a list, a queue, an array or some structure that stores data [13].

![Figure 1. The general architecture of a reconfigurable algorithm.](image)

A frame represents a part or phase of an algorithm. For example, in a sequential consistency algorithm, a frame may represent a coherence protocol that update or invalidate an object that is in the Shared Object Table, which means it is only a part of a sequential consistency algorithm. There are two frame types: control and action frames. A control frame controls a specific characteristic of a data. An action frame is responsible for process or move data between or inside data structures and frames.

The Reconfigurable Layer represents a configuration or an instance of the BL, in which every frame is filled out with one or more compatible blocks at a certain moment. A constructive block is a possible implementation that can fill out with a specific frame.

The Configuration Control Layer chooses the constructive blocks that will fill out each frame at a given moment, thus it controls the configuration swapping. The choice is made based on entry parameters. The CCL can be implemented as a static table with pre-defined decisions, a learning-based algorithm etc.

3. Reconfigurable Sequential Consistency Algorithm

In our Reconfigurable Sequential Consistency Algorithm (RCSA), as show in Fig. 2, each part is a different frame with one or two constructive blocks, to simplify our study. In this algorithm, some frames are action frames (Access Policy, Event Ordering, Constraint and Coherence Protocol) and one is a control frame (Replication Protocol).

![Figure 2. The Basic Layer of the (RCSA).](image)

The Access Policy Frame can be filled out with only one access protocol, single writer/multiple reader (SWMR), because in our algorithm is not allowed multiple writes in the same object at the same time. The Event Ordering Frame is filled out with the sequential ordering. The Constraint Frame can use the write-write (WW) or object-ordered (OO) approach [2]. The Replication Protocol Frame can be filled out with total or partial replication protocol. Finally, the Coherence Protocol Frame can be filled out with the Update Eager (UE) or Invalidate Eager (IE) constructive blocks.

![Figure 3. The Configuration Control Layer (CCL) for our RSCA.](image)

The reconfiguration in the RCSA algorithm is done between the jobs of a workload. In our RCSA, the CCL (Fig. 3) is responsible for selected the configuration for the RCSA reconfiguration. It is implemented as a table that knows the best configuration according to some workload parameters.
3. Related Works

In this research, we found many works about sequential consistency [1] [2] [3] [8] [9] [10] [11] [12], few works about reconfigurable software and algorithms [6] [11] [7] [14], and none about reconfigurable sequential consistency algorithms. In this work, we will discuss some papers that are more relevant and close to our work [1] [10].

In [1], a flexible consistency algorithm is proposed and implemented. As well as our proposal, it uses a different algorithm depending on the user choice. The consistency algorithm implements three-consistency models (Sequential, Causal, Cache), but it uses just the traditional implementation of each one. And, in [10], a sequential consistency algorithm with dynamic protocols switching is proposed and verified by means of formal proofs.

4. Experimental Method

In order to analyze a consistency model, we selected some metrics, architecture and workloads [11] [15] [16].

The most common metrics in the analysis of a consistency algorithm are: mean job response time and number of messages. The mean job response time (in seconds) of distributed shared memory architecture is the mean time interval between the submission and end of a job. And the number of messages is the total number of control messages and object moving messages exchanged between the nodes.

The selected distributed architecture is a cluster composed of 8 nodes interconnected by a Fast Ethernet switch. It was modeled in ClusterSim, a simulation tool developed by our group [14] [17]. As the workload, we made combinations with some characteristics: object size (2K), number of objects (1, 4 and 8) and percentage of write operations (20%, 40% and 60%), generating 9 workloads.

Table 1. RSCA configurations

<table>
<thead>
<tr>
<th>Configurations</th>
<th>Constraint</th>
<th>Coherence Protocol</th>
<th>Replication</th>
<th>Accessors Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conf 01</td>
<td>WW</td>
<td>IE</td>
<td>Total</td>
<td>SW/RE</td>
</tr>
<tr>
<td>Conf 02</td>
<td>WW</td>
<td>IE</td>
<td>Total</td>
<td>SW/RE</td>
</tr>
<tr>
<td>Conf 03</td>
<td>00</td>
<td>IE</td>
<td>Total</td>
<td>SW/RE</td>
</tr>
<tr>
<td>Conf 04</td>
<td>00</td>
<td>IE</td>
<td>Total</td>
<td>SW/RE</td>
</tr>
</tbody>
</table>

In order to test and analyze the performance of the RSCA, we created some configurations (Table 1). We made a total of 36 (4 configurations X 9 workloads) simulations.

It is important to note that each RSCA configuration is a traditional sequential algorithm. In a traditional algorithm, its parts are fixed and cannot be changed over time. For example, in Table 1, Conf01 has the WW constraint, Write Update protocol, and it cannot changes over time. Through the rest of this paper, traditional algorithm and configuration will be treated as synonyms.

5. Experimental Results

In order to analyze the performance of RSCA, we need to compare it to each traditional sequential consistency algorithm. Like we said in the experimental method, the workload is divided in sub-workloads that fit into one of the workload classes (Ex: 1 object, 20% writes and objects size of 2K). Then the CCL, as we show, evaluates the entry parameters, reconfiguring the RSCA to the best configuration for this workload.

Due to the limited number of pages, we present only the results for the number of messages metric. In Fig. 4, we can see that, on average, the configuration 2 presented the best results for number of messages in workloads with many writes.

![Figure 4. The number of messages for all workloads and configurations.](image)

In Table 2, we observe that on average, considering all metrics, RSCA is 36.01% better than the 4 traditional sequential consistency algorithms.

<table>
<thead>
<tr>
<th>Metrics Configurations</th>
<th>Response Time</th>
<th>Number of Messages</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conf 1</td>
<td>79.15</td>
<td>20.65</td>
<td>56.58</td>
</tr>
<tr>
<td>Conf 2</td>
<td>0.47</td>
<td>24.69</td>
<td>56.44</td>
</tr>
<tr>
<td>Conf 3</td>
<td>79.15</td>
<td>19.27</td>
<td>56.35</td>
</tr>
<tr>
<td>Conf 4</td>
<td>17.31</td>
<td>24.40</td>
<td>19.67</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td><strong>44.02</strong></td>
<td><strong>20.00</strong></td>
<td><strong>36.01</strong></td>
</tr>
</tbody>
</table>

The configuration 1 and 3 reach the worst results, because they use the update eager coherence protocol, and this protocol are good just when we have a larger number of writes (higher than 60%) with a small number of reads at the same object (not present in our workloads). Because in this case, the read operations will need the new value of the object and the update protocol (object message) will give it to the reads
operations with a smaller number of messages than the invalidate protocol (invalidate and object message).

Now as an example, suppose that we had a system with a traditional sequential consistency algorithm like configuration 2 (the best on average). If we chose response time as the metric in our CCL algorithm, our RSCA will take only 0.47% of speedup in relation with the configuration 2. However, if we chose number of messages as a metric for the CCL, our RSCA reach 24.40% of speedup. Moreover, in this case, the configuration 2 is worse than configuration 1 that is the worst on average.

In this example, we considered that there weren’t reconfiguration overheads, neither wrong workloads classification. In our case, the reconfiguration overhead is only the spent time for the CCL to switch to the best configuration for the selected workload. The classification of workloads and CCL updates can be done in idle cycles.

6. Conclusions

In this paper, we proposed, developed, implemented (in a simulation tool) and analyzed the performance of RSCA. As general conclusions, we can highlight that on average, the performance of RSCA was 36% better than the other traditional sequential consistency algorithms for all tested workloads. One of the most important results was to show that depending on the selected metric (number of messages), the best algorithm on average (Conf2) may be worse than the worst algorithm on average (Conf1). So, the use of a reconfigurable algorithm may largely improve the system’s performance.

The main contributions of this paper are: the proposal, implementation and performance analysis of RSCA. As future works we can highlight: the inclusion of new blocks in RSCA; an adaptive CCL; study on how to classify the workload found in real applications into sub-workloads etc.

7. References