Improving the Performance of Actor Model Runtime Environments on Multicore and Manycore Platforms

Emilio Francesquini, Alfredo Goldman, Jean-François Méhaut
emilio@ime.usp.br
Institute of Mathematics and Statistics – University of São Paulo
Grenoble Informatics Laboratory – University of Grenoble

Abstract

The actor model is present in many systems that demand substantial computing resources which are often provided by multicore and multiprocessor platforms such as non-uniform memory access architectures (NUMA) and manycore processors. Yet, no mainstream actor model runtime environment (RE) currently in use takes into account the hierarchical memory characteristics of these platforms. These REs essentially assume a flat-memory space therefore not performing as well as they could. In this paper we present our proposal to improve the performance of these systems. Using knowledge about the general characteristics of actor-based applications and the underlying platform, we propose techniques spanning from memory management to scheduling and load-balancing. Based on previous work, we present our design guidelines for the RE adaptation to the Kalray-MPPA-256 manycore processor.

Introduction and Motivation

Symmetric multiprocessors (SMP) is the customary architecture used for multiprocessor machines. However, as the number of processors (and cores per processor) increases, the common bus might represent a bottleneck to the overall system performance. Typical solutions involve the use of hierarchical memory platforms such as non-uniform memory access (NUMA) architectures and network-on-chip (NoC) many-core processors.

These platforms often present several difficulties for the application developer such as concurrency control, cache coherence control, and topology-aware memory and processes placement and migration. The actor model represents a simple yet powerful solution to this problem, willingly taking the application developer away from the architectural idiosyncrasies of the machine. Thus, the actor model runtime environment (RE) becomes the responsible for the efficient use of the underlying architecture. However, to the best of our knowledge, no current mainstream actor RE takes into account these platform characteristics, essentially assuming a flat-memory space platform with constant communication costs.

Figure 1: A simplified view of the STRAB4 Multicore NUMA Platform

The actor RE works as an additional layer between the operating system and the application and, as such, it has supplementary information about the application behavior. This information can be employed to make better scheduling and load-balancing decisions and thus improve the overall performance of the system.

Acknowledgments

The authors would like to thank CAPES for funding this research under project CAPES/Coimbra 660/10.

References


Figure 2: Number of L2 cache misses per 1/10 of a second caused by RM

Figure 3: Curves of L3 cache misses per 1/10 of a second caused by RM

Figure 4: Number of L2 cache misses per 1/10 of a second caused by RM

A Hierarchical Solution

Based on the knowledge of the platforms and on the analysis of the actor-based applications, our proposal for improving the performance on these systems can be divided in the following two complementing aspects:

Affinity Group Maintenance

- Communication graph and affinity groups are extremely dynamic.
- Trying to maintain an on-line representation could impose an important overhead to the runtime environment.
- We propose, therefore, to use hints provided by the application developer.
- The application developer usually has good insights into the behavior of the application.
- We provide the application developer the means to indicate which actors are hubs.

The actor RE always should be aware of the changes in the communication graph.

Possible candidates for migration are evaluated taking into consideration:
- The actor RE knows the current position of the actor.
- The actor RE is aware of the affinity groups and could consider suggesting a migration to一处 closer node.
- Two load-balancing strategies: One is to use a local node spread initial placement policy for the regular actors.
- We apply a spread initial placement policy for these actors.
- A hub’s affinity group is mainly composed by the actors it is spawning.
- We apply a spread initial placement policy for the regular actors.
- This keeps actors near to their hubs and promotes a good initial distribution among the schedulers.

Load-Balancing

Load-balancing mechanisms deal with actor migration between schedulers. These mechanisms are central to our approach:

- Load-balancing aims to keep every processing unit busy and at the same time provide a fair share of processor time to each actor.
- To keep actors affinitized, the load-balancing tries to keep actors and their affinity group close together.
- We use after good trade-offs, in terms of performance, between these two aspects of the execution.
- We believe that an event-based scheduler with multiple queues is the most appropriate.
- Provides soft-affinity.
- Scheduling interpretation for manycores, such as MPPA-256, where there is no global memory space or cache-coherence.

Possible candidates for migration are evaluated taking into consideration the topological distance, specific to the underlying platform, between the scheduling node and the destination scheduler.

NUMA Approach

- We evaluate the performance impact of our proposal using the Erlang VM as the base for our modifications.
- Using a set of standard benchmarks, we evaluated the performance of the modified VM on the machine depicted in Figure 1.

Table: Obtained Speedups

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Short Initial</th>
<th>Intermediate</th>
<th>Short Initial</th>
<th>Intermediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>1.18</td>
<td>1.12</td>
<td>1.00</td>
<td>1.14</td>
</tr>
<tr>
<td>big</td>
<td>0.93</td>
<td>0.92</td>
<td>0.95</td>
<td>0.93</td>
</tr>
<tr>
<td>db</td>
<td>1.27</td>
<td>1.14</td>
<td>1.27</td>
<td>1.12</td>
</tr>
<tr>
<td>dst_int</td>
<td>2.50</td>
<td>1.35</td>
<td>2.08</td>
<td>1.23</td>
</tr>
<tr>
<td>malloc</td>
<td>1.04</td>
<td>1.09</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>timer, wheel</td>
<td>2.13</td>
<td>1.89</td>
<td>2.12</td>
<td>1.88</td>
</tr>
</tbody>
</table>

Figure 5: Obtained execution time of the benchmarks

Figure 6: Normalized execution time of the benchmarks

Manycore Approach

- Some platforms, such as Tiberia’s T364/PRO, can make applications run under the impression that it is an SMP machine.
- Some work on the scalability of the VM running on the node has been conducted [3].
- To the best of our knowledge, no adaptations to the specific characteristics of manycore platforms exist on mainstream actor runtime environments.

The MPPA-256 Platform

Scheduling

- Event-Based: Hard limit of 256 total threads
- Aggressive Hot-Offloading: Limit of 2MB of memory per cluster with no virtual memory support
- Hybrid Queue: Multiple queues however only one queue per cluster
- No need for intra-cluster load-balancing
- Simplifies the code and decreases local memory consumption
- Initial Placement Policy
- Bins: Spread among clusters
- Regular: Allocate on the same cluster

Load-Balancing

- Impossible to run an actor outside its home node.
- Any migration, therefore, implies hop relocation.
- Since migration is expensive, an implementation threshold to set off actor migrations might need empirical evaluation.
- Hierarchical load-balancing and work-stealing are still desirable in this platform.
- Actor Load Balancing: Try to achieve the delivery of messages
- Proxy Clusters with local caches
- Global list of locations kept by the RE Node