Dynamic Load Balancing for hybrid applications

Marta Garcia Gasulla, Julita Corbalan and Jesus Labarta
Barcelona Supercomputing Center and Universitat Politecnica de Catalunya
marta.garcia@bsc.es, julita.corbalan@bsc.es, jesus.labarta@bsc.es

*Abstract-**The DLB (Dynamic***

DLB relies on the usage of hybrid programming models and exploits the malleability of the second level of parallelism to redistribute computation power across processes.

I. Introduction

In parallel computing, the loss of efficiency is an issue that concerns both system administrators and parallel programmers. The growth in number of computing units that clusters experienced the last years has helped speeding up applications but has worsened some problems that affect the efficient use of the computational power.

One of the problems that has deteriorated with this growth is load balance. Although it is a concern that has been targeted since the beginning of parallel programming, there is not a universal solution.

In this paper we will talk about the Load Balancing Library, DLB, and a balancing algorithm, LeWI, that can improve the performance of hybrid applications. DLB can load balance an application at runtime without modifying nor analyzing the application.

In a previous work [1] we showed the potential of DLB and LeWI when executed with MPI+OpenMP applications.

In this paper we are showing the results of porting DLB to OmpSs. And how integrating some features of DLB in the runtime the performance can be improved.

II. Dynamic Load Balancing Library (DLB)

The Library

The Dynamic Load Balancing (DLB) is a shared library that helps load balance applications with two levels of parallelism. The current version provides support for:

- MPI+OpenMP
- MPI+OmpSs

The aim of DLB is to balance the MPI level using the malleability of the inner parallel level. One of its main properties is that the load balancing will be done at runtime without analyzing nor modifying the application previously. The algorithm that has showed better performance results is LeWI (Lend When Idle) [1]. And this is the algorithm that we are going to explain in the following section and use for the performance evaluation.

LeWI Algorithm

The philosophy of LeWI is based on the fact that when an MPI process is waiting in an MPI blocking call none of its threads is doing useful work. Therefore, we have one or several CPUs that are not being used. LeWI aims to use these CPUs to speedup other MPI processes running in the same node. The usual behavior of an MPI application is that if a process is blocked in an MPI call it is waiting for one or several other processes to finish. Speeding up processes that are more loaded helps to load balance the application and speedup the whole application.

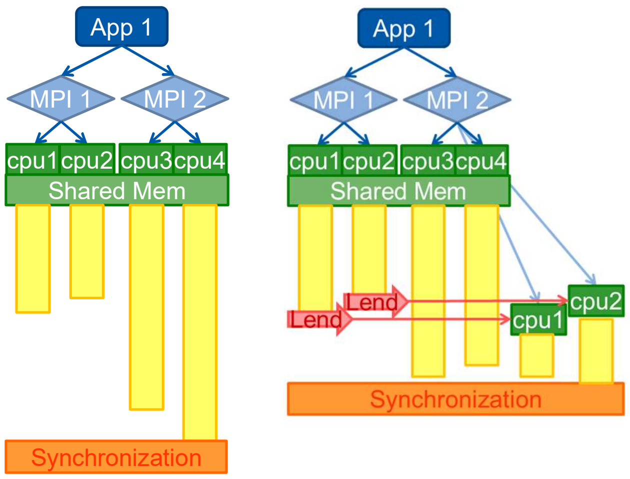

Fig. 1. LeWI Algorithm behavior: Original Application vs. Application load balanced with LeWI.

In Fig. 1 we can see the behavior of the LeWI algorithm when balancing an unbalanced application. On the left shows an unbalanced hybrid application with 2 MPI processes and 2 threads per process. In this example MPI process 2 is more loaded than MPI process 1 and this makes that MPI process 1 must wait in an MPI communication for some time.

At the right we can see the behavior of the same application when executed with the LeWI algorithm. When an MPI process reaches a blocking MPI call it will lend its CPUs to the other MPI processes running in the same node. With the lent CPUs the more loaded MPI processes will be able to finish its computation faster and the MPI process 1 will be less time waiting in the MPI call. The use of the computational resources will be better and the application will perform better.

The first version of LeWI did not use mapping of threads to cpus, it adjusted the total number of running threads. But with the porting of DLB to OmpSs this offers us the posibility of mapping each thread to a cpu and lending a specific cpu, avoiding a temporal oversuscription.

III. Performance Evaluation

The experiments have been executed on Marenostrum3. Marenostrum3 is based on Intel SandyBridge processors. Its compute nodes are IBM iDataPlex dx360 M4 X servers with two 8-core Intel Xeon processors (E5-2670) per node and 32 GB of shared memory. They also include a hard drive of 500Gb and an MPI network card Mellanox ConnectX-3 Dual Port QDR/FDR10 Mezz Card. For management and GPFS they have two Gigabit Ethernet network cards.

We have executed the BT-MZ a benchmark from the NAS-Multizone benchmark suite. BT-MZ has been executed in one node of Marenostrum (16 cpus) with different configurations of MPI processes and threads.

And Lulesh a mini-app representative of simplified 3D Lagrangian hydrodynamics on an unstructured mesh. Lulesh has a parameter that can be changed to increase or decrease the amoount of imbalance present in the execution. A low value means a good load balance and a high value means more imbalance. Lulesh has been executed in 4 nodes of Marenostrum (64 cpus).

For each execution we can see four different series:

- **Binding**: the original execution of the application without load balancing executed with mapping of threads to cpus.
- **No Binding:** the original execution of the application without binding of threads to cpus.
- **No Binding + LeWI:** Execution with LeWI and without binding of threads.
- **Binding + Mask:** Execution with LeWI and with mapping of threads to cpus.

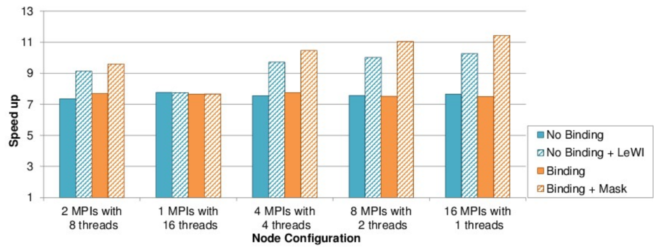

Fig. 2. Speed up obtained by BT-MZ with and without LeWI

In Fig. 2 we can see the speed up obtained by the different executions, when using the load balancing algorithm LeWI we can improve the speed up of the application. But the gain can be higher using a mapping of threads to cpus.

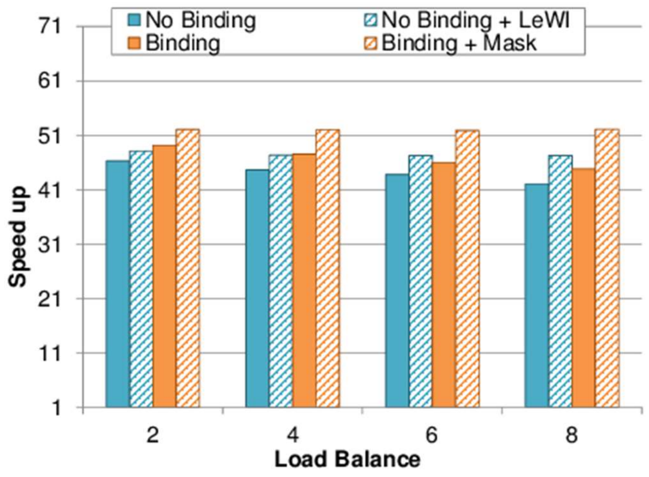

Fig.3. Speed up obtained by Lulesh with and without LeWI

Fig. 3 shows the speed up of Lulesh with a different amount of load imbalance. We can see how the speed up of Lulesh decreases as the amount of load imbalance increases, but when using LeWI the performance is better and maintained independently of the amount of imbalance.

We can se also that the performance when using a mapping of threads to cpus is better when using dynamic load balancing than when not mapping threads to cpus.

IV. Conclusions

In this paper we have presented a load balancing algorithm, LeWI, that has been implemented within a dynamic library, Dynamic Load Balancing (DLB).

The DLB library allows us to balance applications with two levels of parallelism without modifying the application or studying the imbalance it presents. The current version of the library can balance hybrid MPI+OpenMP and MPI+OmpSs applications.

We have shown the relevance of binding of threads to cpus. And how the support from the runtime can help load balance applications.

References

[1]M. Garcia, J.Corbalan, J.Labarta, “LeWI: A Runtime Balancing Algorithm for Nested Parallelism” International Conference on Parallel Processing, ICPP 2009.
[2]I. Karlin, J. Keasler, R. Neely. LULESH 2.0 Updates and Changes. August 2013, pages 1-9