BIOINFORMATICS APPLICATIONS NOTE

PAR: a PARallel and distributed job crusher

François Berenger1,∗, Camille Coti2 and Kam Y. J. Zhang1

1Zhang Initiative Research Unit, Advanced Science Institute, RIKEN, 2-1 Hirosawa, Wako, Saitama 351-0198, Japan and 2HPC Group, Department of Mathematics, Iowa State University, Ames, IA 50011-2251, USA

ABSTRACT

Summary: Bioinformatics are tackling increasingly computation-intensive tasks. In the meantime, workstations are shifting towards multi-core architectures and even massively multi-core may be the norm soon. Bag-of-Tasks (BoT) applications are commonly encountered in bioinformatics. They consist of a large number of independent computation-intensive tasks. This note introduces PAR, a scalable, dynamic, parallel and distributed execution engine for Bag-of-Tasks. PAR is aimed at multi-core architectures and small clusters. Accelerations obtained thanks to PAR on two different applications are shown.

Availability: PAR is released under the GNU General Public License version three and can be freely downloaded (http://download.savannah.gnu.org/releases/par/partgz).

Contact: berenger@riken.jp

Received on July 20, 2010; revised on September 7, 2010; accepted on September 19, 2010

1 INTRODUCTION

Bioinformatics are significant high-performance computing users, in particular for simulations of biologic phenomena. On the other hand, the available hardware is not only getting faster but also much more parallelized (Intel publicly reported working on 80 cores prototype chips in 2007). In this context, most bioinformatics could benefit from an easy-to-use software to harness such computing power.

The focus of this note is Bag-of-Tasks (BoT) applications execution. As the name suggests, BoT applications can be seen as a bag, filled with tasks to do, each being independent from all the others. A middle-ware for BoT applications is called a job crusher. It has to consist of at least a server component connected to a set of clients.

This note introduces PAR, a parallel and distributed job crusher working in pull mode and inspired by desktop grid platforms. Workers join the computation and can be added dynamically at run-time; the server delivers tasks to workers available at a given moment. PAR is actually a transposition of some concepts and features from previous distributed middle-ware to small HPC clusters and multi-core workstations.

This article is organized as follows: Section 2 presents an overview of related projects and technologies used in bioinformatics. Section 3 presents two examples using PAR to illustrate scalability. The last section lists upcoming enhancements.

∗To whom correspondence should be addressed.
Workers were started soon after the server, but could have joined the use of the parallel hardware with minimum effort required on the workers and the master is very small, and this allows for an effective remarkbly well. The overhead due to communications between shared by all threads). We can see that the application scales parallel version is attributed to Python's problem with multi-thread case and 15.54 in the distributed one. Lower performance of the speedup obtained by PAR can be as high as 14.01 in the parallel PAR runs only in user space.

Prior to timing experiments, needed programs and data were copied to each machine by the user. During experiments, PAR was started in server mode with a list of commands to execute. Workers were started soon after the server, but could have joined the computation later if we were not interested in the shortest completion time. The Unix 'time' command was used and averaged over two trials to measure the real time spent by PAR to complete all tasks. Unlike previous job crushers, PAR server's life cycle is only tied time. The Unix 'time' command was used and averaged over two trials to measure the real time spent by PAR to complete all tasks.

Results are shown in Figure 1. The first bar is the real time elapsed when not using PAR. The second bar is the time spent when using PAR in parallel mode, and the following bars are durations in distributed mode. On a CPU-intensive task and when using 16 CPUs, the speedup obtained by PAR can be as high as 14.01 in the parallel case and 15.54 in the distributed one. Lower performance of the parallel version is attributed to Python's problem with multi-thread applications (the Python interpreter uses a global lock mechanism shared by all threads). We can see that the application scales remarkably well. The overhead due to communications between workers and the master is very small, and this allows for an effective use of the parallel hardware with minimum effort required on the user's side.

4 FURTHER DEVELOPMENTS

PAR can be used on network of Unix-like workstations. It can take advantage of a Network shared File System (NFS). However, because of poor NFS performances, data-intensive tasks should be computed on top of a Distributed File System (DFS). As DFS are still rare even within clusters, we envisage to plug in such a functionality into PAR. A prototype has been implemented but is still in experimental stage.

PAR should integrate fault-tolerance policies, in order to be used safely even with more workers over longer periods, and with minimal overhead.

Furthermore, compression could be added to speedup communications. Encryption would be similarly easy to add and would allow PAR to be used over untrusted networks.

Finally, features can be added for large-scale experiments. For example, requesting groups of jobs instead of one at a time would lower the load on the server part. Allowing PAR to run both as a server and as a client would allow it to be deployed in layers, which could be used to connect several clusters together and increase scalability. Requests and contributions from users are also considered.

ACKNOWLEDGEMENTS

We thank all the PAR users, especially early ones like Rojan Shrestha for providing feedback and useful feature requests. We wish to thank RIKEN, Japan, for an allocation of computing resources on the RIKEN Integrated Cluster of Clusters (RICC) system.

Funding: "Initiative Research Unit" program from RIKEN, Japan. Conflict of Interest: none declared.

REFERENCES


Borga,A.L. et al. (2009) Folding@home: lessons from eight years of volunteer distributed computing.


