# HPC CLOUD APPLIED TO LATTICE OPTIMIZATION [1]

Changchun Sun, Hiroshi Nishimura, Susan James,
Kai Song, Krishna Muriki, Yong Qin

ALS and IT Divisions
Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CA 94720

April, 2011

[1]Work supported by U.S. DoE Contract No. DE-AC02-05CH11231

## DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

# HPC CLOUD APPLIED TO LATTICE OPTIMIZATION *

Changchun Sun †, Hiroshi Nishimura, Susan James, Kai Song, Krishna Muriki, Yong Qin
Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, CA 94720, USA

*Abstract*

As Cloud services gain in popularity for enterprise use, vendors are now turning their focus towards providing cloud services suitable for scientific computing. Recently, Amazon Elastic Compute Cloud (EC2) introduced the new Cluster Compute Instances (CCI), a new instance type specifically designed for High Performance Computing (HPC) applications. At Berkeley Lab, the physicists at the Advanced Light Source (ALS) have been running Lattice Optimization on a local cluster, but the queue wait time and the flexibility to request compute resources when needed are not ideal for rapid development work. To explore alternatives, for the first time we investigate running the Lattice Optimization application on Amazon's new CCI to demonstrate the feasibility and trade-offs of using public cloud services for science.

## INTRODUCTION

### *Lattice Optimization*

Lattice Optimization is a challenging aspect of storage ring design. It is a multi-objective and multi-variable optimization problem, *i.e.*, simultaneously evaluating multiple objectives, such as the emittance, optics function, dynamics aperture and beam life time. The variables used for the optimization are usually the magnetic settings, *e.g.*, quadrupole and sextupole strengths. Two methods, brute force scan and Genetic Algorithms, have been successfully developed and applied by storage ring designers to optimize lattice [1]. Both of these methods heavily rely on the computing capability. Fortunately, High Performance Computing (HPC) allows us to find optimum lattice solutions in a reasonable time. At Advanced Light Source (ALS) [2], we recently developed a genetic optimization code, GeneticTracy, on top of Tracy++ [3] to search for alternating high and low beta lattice for the ALS future upgrades [4]. To search for the optimal solutions of the lattice, a large number of population and generation sizes are required. GeneticTracy implements Message Passing Interface (MPI) with master-slave model to achieve parallelization.

### *Cloud Offerings*

Public Cloud offerings have increased in scope and availability over the past couple of years; however, it has not been widely adopted by the scientific computing community despite growing interest. Jackson, Ramakrishnan, *et al.* [5, 6, 7], have done some previous research on defining the requirements to adopt Cloud for average HPC usage model, as well as some performance evaluation on the Amazon Elastic Compute Cloud (EC2) platform [8]. Their work was developed on the standard instances, which is not the latest Cluster Compute Instances(CCI) that our study is based on. This instance type was introduced by Amazon to be their public cloud offering for High Performance Computing. CCI is different from the original Amazon EC2 instances mainly because it has a predefined architecture and hardware specification and the hardware is not shared with other Amazon EC2 instances. Thus we can launch multiple instances to form a cluster that is similar to a typical small to mid-range HPC cluster. Amazons Placement Group, a logical entity, facilitates the launching of multiple CCI instances into a cluster of instances with a low latency and high bandwidth interconnect. Amazon also provides a re-sizable compute capacity with this instance type so that one can add and remove instances to the cluster as needed. This flexibility is one of the advantages of Amazon EC2.

## CLUSTER CONFIGURATIONS

### *Building a HPC cluster in Amazon EC2*

In this study, we launched CCI instances using the Amazon EC2 command line API or Amazon Web Services (AWS) Management Console. One can store data on the root device, which is the local instance store of these compute instances which will be accessible only from that instance or on Amazons Elastic Block Storage (EBS) volume, such that data can be persistent while independent on the lifetime of the instances. EBS volumes can be created and attached to instances by using EC2 API or AWS console. There is no limit to how many CCI instances that can be launched. To be able to run a traditional HPC application such as the Lattice Optimization code, we needed to create an environment similar to traditional HPC cluster platforms using these independently instantiated compute instances and EBS storage volumes. We also needed to create a master compute instance which acts as a login node or head node to our virtual HPC cluster, and all the remaining instances act as compute or worker nodes on which jobs are executed. We attached an EBS volume which contains all the application software stack, source code, and input/output data files only to the head node and then mounted that volume over NFS on the compute instances such that they are accessible to all the instances. Using the private interface addresses of all the compute nodes we can build a machine file with which MPI parallel jobs can be easily launched on the virtual cluster. Jack-

* Work supported by U.S. DoE Contract No. DE-AC02-05CH11231
† CCSun@lbl.gov

son, Ramakrishnan, *et al.* [5, 6, 7] have developed a set of scripts using Python to deploy virtual clusters on Amazon EC2. In order to automate the model described above, we developed our scripts based on this work.

We launched the CCI instances from scratch every time we needed them, and terminated them after we finished our runs in order to avoid paying for the instances while they are not in use. Amazon provides a start/stop feature for their instances, but we opted not to use them. Although this feature might look attractive, the private interface address of the restarted instances will be different thus forcing us to remake all changes which are based on the private interface address. In addition, if we used the placement group feature of the Amazon cluster compute instances during initial configuration of the cluster, trying to stop and start these instances might not work because of the capacity limitation within that placement group. The placement group is tied to a deployment rack in Amazons data center and instances are physically tied to that rack. If some other instances start running in this location while our instance is stopped we cannot restart our stopped instance as our instances stay tied to their racks for their lifetime.

### *Clusters*

All results presented in this paper were collected from multiple-user production environments. Table 1 listed four different clusters that we used to compare performance and cost. Except Amazon EC2 cluster, the other three run CentOS Linux 5.5 provisioned by Perceus [9] on bare metal. CentOS 5.4 Linux is run on Amazon EC2 virtual cluster to make a fair comparison. However since we were not able to access the hyper-threading configuration on Amazon public cloud, it is not only running in a virtual environment, but also running with hyper-threading enabled which can reduce performance depending on application. For the interconnect, Amazon EC2 is equipped with 10Gb Ethernet, LRC has 4X DDR Infiniband with 3:1 blocking factor, Mako has 4X QDR non-blocking Infiniband, and LR2 has 4X QDR 3:1 blocking Infiniband. Among all four clusters, Amazon EC2 and Mako use Intel Nehalem, and LR2 uses Intel Westmere processor technology. Only LRC uses the last generation Intel Harpertown technology.

Table 1: System Configurations

| | EC2 | LRC | Mako | LR2 |
|---|---|---|---|---|
| CPU Arch. | X5570 | E5430 | E5530 | X5650 |
| CPU Freq.[1] | 2.93 | 2.66 | 2.40 | 2.66 |
| Cache | 8 MB | 12 MB | 8 MB | 12 MB |
| HT | On | Off | Off | Off |
| Interconnect[2] | 10 | 20 | 40 | 40 |
| Virtualization | On | Off | Off | Off |
| Cores/Node | 16 | 8 | 8 | 12 |
| Memory/Node | 23 GB | 16 GB | 24 GB | 24 GB |

[1]. GHz
[2]. Gb/s

## PERFORMANCE ANALYSIS

### *GeneticTracy*

Lattice Optimization problem with a population size of 50K was run over 1000 generations on all the aforementioned local shared clusters and the virtual cluster within Amazon public cloud. Table 2 lists the total time taken by the optimization on various clusters. GeneticTracy is a typical embarrassingly parallel code, and it does not require too much memory, thus the performance is tightly coupled with the CPU architecture and frequency but not the interconnect and memory subsystems. As a result, it is well suited for the Amazon EC2 platform as well as the standard instances. Running on EC2 achieves the flexibility that other shared resources are not able to to provide. For instance, one of the advantages of EC2 is HPC on demand, which provides a mean to require CPU hours when needed. This is extremely useful for researchers to blast large numbers of jobs to search for optimal parameters at the trial stage, or to achieve high throughput during computation. By applying on-demand HPC method, one does not have to wait for queues to be available from a shared resource, or be restricted by the job limit that is typically enforced on most shared resources to ensure a fair use. Running on a public cloud also saves the cost of maintaining a local cluster.

Table 2: GeneticTracy Runtime on various clusters

| | EC2 | LRC | Mako | LR2 |
|---|---|---|---|---|
| Time (mins) | 679 | 857 | 724 | 566 |

### *General Performance*

During this research, we systematically evaluated performances of CPU, memory, interconnect subsystems, as well as the overall system performance and the performance of the GeneticTracy application. However, since GeneticTracy is only a trivial parallel application, it does not reveal the Amazon EC2 performance for other types of scientific computing programs, especially those heavily parallelized applications which take advantage of modern high speed interconnects, *e.g.*, MPI. Here we demonstrate the overall performance with one of the most commonly used benchmarks – High-Performance Linpack (HPL) [10] for all these four clusters. HPL is a benchmark that solves a random dense system of linear equations on a wide variety of distributed memory machines. It is adopted by TOP500 [11] supercomputing site as the primary benchmark for acceptance. Due to the nature of this benchmark, it provides the capability to stress CPU, memory, and interconnect subsystems, and evaluate the system performance as a whole. Figure 1 shows $R_{max}/R_{peak}$ *vs* number of nodes used, in which $R_{max}$ is the maximal Linpack perfor-

mance achieved, and $R_{peak}$ is the theoretical peak performance, thus $R_{max}/R_{peak}$ provides a way to estimate how efficient the system is to achieve the peak performance at various configurations. From this figure we can clearly see that LRC, Mako and LR2 are all consistent on this efficiency up to 16 nodes, which suggests that the performance degradation is not significant when the number of nodes is increased. Nonetheless, Amazon EC2 drops considerably from 87% on one node (8 processors) to 67% on 16 nodes (128 processors). The major reason for this is the suboptimal virtual 10Gb Ethernet interconnect. We were able to reach close to 400 Mbps of uni- and bi-directional network bandwidth. Compared to Infiniband used on all other local clusters, this result is expected and reasonable. We also notice that the memory subsystem can only achieve 50∼60% of theoretical memory bandwidth. This also limits the HPL performance at large N value. Although the performance efficiency of CCI instances is better than that of the original EC2 standard instances (23∼27%) [5], we see that Amazon EC2 CCI suffers from performance degradation for large scale parallel MPI applications, especially network or memory bound applications. Nevertheless, the CCI offering is a great improvement, which has made the Amazon public cloud even more attractive, especially for those who require access to small to mid-range HPC clusters.

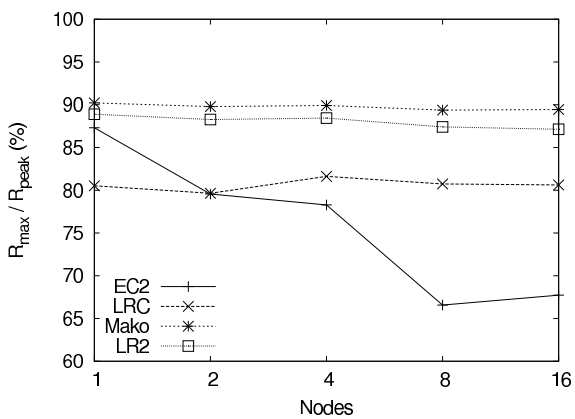


Figure 1: HPL $R_{max}/R_{peak}$ *vs* number of nodes

## SUMMARY

Cloud computing has attracted a lot of attention recently. Amazon also provides HPC cloud as a pay-as-you-go usage model. The flexibility gives users the choice of saving cost by not maintaining small to medium sizes of in-house clusters, as well as provides the high throughput capability. The performance on the recently introduced CCI has also been greatly increased to meet the growing needs from the scientific computing community, which makes it a good candidate to researchers seeking on-demand computing capacity. However, as demonstrated in this paper, EC2 may not work well for large scale parallel applications, which represent a large number of scientific HPC applications.

## ACKNOWLEDGEMENTS

This research used various resources provided by Laboratory Research Computing (LRC) [12], which is managed by the LBNL IT Division; as well as resources from the University of California Shared Research Computing Services (ShaRCS) Cluster [13], which is managed by the LBNL IT Division and SDSC for the University of California, Office of the President. The authors would also like to thank David Robin and Jerry Kekos from ALS, Gary Jung from IT Division for their encouragement and support, as well as Andrew Paoli and Miles Ward from Amazon for their assistance and help.

## REFERENCES

[1] C. Sun *et al.*, these proceedings, TUODN4.

[2] "1-2 GeV Synchrotron Radiation Source, Conceptual Design Report," LBL PUB-5172 Rev. LBL, 1986.

[3] H. Nishimura, "Goemon, A C++ Library for Accelerator Modeling and Analysis", Proceedings of PAC 2001, Chicago, June 2001.

[4] C. Sun *et al.*, these proceedings, WEP031.

[5] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman and N. Wrigh, "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud", 2nd IEEE International Conference on Cloud Computing Technology and Science, December 2010, Indianapolis, IN.

[6] K. Jackson, L. Ramakrishnan, K. Runge and R. Thomas, "Seeking Supernovae in the Clouds: A Performance Study", ACM Science Cloud 2010, June 2010, Chicago, IL.

[7] L. Ramakrishnan, K. Jackson, S. Canon, S. Cholia and J. Shalf, "Defining Future Platform Requirements for e-Science Cloud", ACM Symposium on Cloud Computing, June 2010, Indianapolis, IN.

[8] Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2`.

[9] Provision Enterprise Resourses and Clusters Enabling Uniform Systems (Perceus), `http://www.perceus.org`.

[10] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, `http://www.netlib.org/benchmark/hpl/`.

[11] TOP500, `http://www.top500.org`.

[12] Laboratory Research Computing (LRC), `http://lrc.lbl.gov`.

[13] UC Shared Research Computing Services (ShaRCS), `http://srcs.ucop.edu`.