Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2.

[Extended Abstract] *

Constantinos Evangelinos and Chris N. Hill

Department of Earth, Atmospheric and Planetary Sciences, MIT
77 Massachusetts Ave.
Cambridge, MA 02139, USA
ce107@mit.edu and cnh@mit.edu

Abstract

In this article we describe the application of HPC standard benchmark tests to Amazon's EC2 cloud computing system, in order to explore the utility of EC2 for modest HPC style applications. Based on these benchmarks we find that the EC2 cloud system is emerging as a credible solution for supporting responsive on-demand, small sized, HPC applications. We illustrate this point with a demonstration of a low-order coupled atmosphere-ocean simulation running in parallel on an EC2 system. This demonstration highlights one significant way in which cloud computing could impact traditional HPC computing paradigms that, together with related ease-of-use technologies (described only briefly in this paper), could shift the manner in which many HPC systems are deployed.

Categories and Subject Descriptors D.2.12 [*Software*]; J.2 [*Physical Sciences and Engineering*]: Earth and atmospheric sciences

General Terms

Keywords

1. Introduction

Traditional parallel scientific high-performance computing (HPC) is dominated by batch processing oriented queue based systems. This operating model has achieved much, but the model can also be a significant impediment to progress. In particular a queue based batch paradigm makes on-demand, responsive and highly customized computational science research hard to undertake. On-demand cloud computing appears to offer an attractive new dimension to HPC, in which virtualized resources can be sequestered, in a form customized to target a specific scenario, at the time and in the manner they are desired. We believe this concept lends itself to more interactive responsive computational science. In this article, in order to better understand the suitability of current cloud computing implementations for HPC, we perform traditional HPC benchmarks on Amazon's EC2 cloud system. Encouraged by the results we then develop a demonstration scenario, running an example coupled atmosphere-ocean model. This parallel scientific simulation scenario illustrates one way in which we envisage cloud computing could begin to have an impact on parallel scientific HPC applications in the near future.

The structure of this article is as follows: in section 2 we briefly discuss the Amazon EC2 infrastructure and the opportunities it offers to a computational scientist. In 2.1 we discuss and benchmark relevant cloud computing resources offered in EC2. In section 3 we discuss the network performance, under the available MPI middleware, that one can achieve on EC2, identifying certain caveats and problems. In section 4 we describe an actual MPMD climate application we ported and run on EC2. Finally, in 5, we conclude with our findings and future plans.

2. Amazon Web Services: Elastic Compute Cloud

In 2006 Amazon announced its Simple Storage Service (S3) its Elastic Compute Cloud (EC2). S3 and EC2 together offer a cloud compute and storage resource that provides the possibility of computing on virtual parallel clusters generated and destroyed on demand. EC2 is based on Linux and

* Paper presented at the CCA-08 in Chicago.

CCA-08 October 22–23, 2008, Chicago, IL

Xen (3) and various O/S images, Amazon Machine Images (AMIs), can be supported.

2.1 EC2 Instances

Currently Amazon EC2 offers five "hardware" instance types with different characteristics (cpu power, memory, disk and addressability) and different pricing. Amazon provides a basic measure of an EC2 Compute Unit (1) for compute power: "One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor." These instance types have been classified by Amazon as (i) a set of three *standard* instances called *m1.small*, *m1.large* and *m1.xlarge* (ii) pair of *high_cpu* instances called *c1.medium* and *c1.xlarge*.

We instantiated all five types and looked at the actual hardware that was provided. It is apparent from the type of cpus encountered that all Amazon EC2 systems we were assigned were dual socket multicore cpus. In the case of the standard instance type, a "half of a core" behavior is enforced by allowing cpu utilization of the single processor virtual machine to be at most 50%, a limitation which system utilities such as "top" easily show. For benchmarking one can, therefore, only meaningfully speak of wallclock and not of cpu time on these instances, even in the case of serial codes. Ensuring that no daemons or other processes are contending for cpu time is, therefore, critical for all measurements to be of any use. For cpu intensive applications that spend most of their time with a full pipeline this 50% utilization limitation means that one instance is essentially equivalent to a downrated 1.3 GHz Opteron processor (hence the definition of an EC2 Compute Unit). The same cannot necessarily be said for memory bandwidth or memory latency (important quantities for many HPC applications). By virtue of running in a virtual machine environment, we also cannot know at any given moment whether the other cores in our actual physical system (but not in our O/S image) are being used by ourselves or another user. Such co-habitation of the physical system leads to contention for the processor socket bandwidth (for the standard instance type), and the main memory bandwidth (for the medium instance types). The only exceptions to this rule are the xlarge instances that essentially run a single Xen domU virtual machine on top of the base Xen dom0 one for the whole two socket node. The high-cpu instance types employ quad core core processors: while these Core2 architecture based processors are very fast, their effective memory bandwidth is further reduced as 4 processor cores share the same socket pins to the northbridge path to main memory (which is in turn shared with the other socket's four cores).

Within each of the families of instances, pricing is proportional to the number of EC2 Compute Units but compute power wise (based on Amazon's figures) the 32-bit High-CPU instance appears to be more cost-effective. For our work we opted to initially concentrate our tests on the 32-bit images only that are the cheapest to use (at $0.1 and $0.2 per cpu hour respectively).

System	1 thread	N threads	per core B/W
m1.small	5.4GB/s	n/a	5.4GB/s
c1.medium	3.6GB/s	5.3GB/s	2.6GB/s
Opteron 1.4 GHz	2.8GB/s	5.6GB/s	2.8G/s

Table 1. STREAM Triad bandwidth (B/W) per core on Amazon EC2

System	Class A	Class W	EP (A)	EP (W)
m1.small	132	149	6.66	6.73
c1.medium	312	357	15.59	15.04
ratio	2.36	2.40	2.34	2.23

Table 2. Geometric means of serial NPB3.3 (excluding EP) and EP results. Units are in Mop/s or Million Random Numbers Generated per second

2.2 Test with standard HPC benchmarks

We set out to use some basic benchmarks of performance potential for a general class of scientific applications. Memory bandwidth was tested using the STREAM benchmark (21) employing the of breed 32-bit executables available for each processor type. The results shown in Table 2.2 show surprisingly high bandwidth for the standard instance type (significantly better than an actual first generation 1.4GHz Opteron system - thanks to using DDR2 memory). The High-CPU medium instance delivers bandwidth similar to what one would expect out of such an architecture but better than what one would expect from two cores sharing the same socket's pins to main memory. This leads us to hypothesize that in fact the hardware mapping of the virtual to physical processors picks 2 cores from different sockets for this type of instance.

To test computational performance on a wide set of model applications and kernels we employed the serial version of the NAS Parallel Benchmarks (NPB) (2) (v. 3.3). We used both the workstation class (W) and the smallest of the parallel classes (A). (More than 10 years since it was defined, class A now fits very comfortably inside any of our instances or available workstation systems). To level the playing field we compiled everything with the system compiler and optimization flags with a target of the lowest common denominator (the Opteron in the standard instance). We see in Table 2.2 that the geometric mean of (BT, CG, FT, IS, LU, MG, SP, UA - we excluded DC) as well as the value of the EP tested are between 2.2 and 2.4 times faster on the High-CPU instances. This is less than half of Amazon's EC2 relative compute unit rating for this machine, but still appears to be worth the 2x price.

	to /tmp		over NFS	
System	Write	Read	Write	Read
m1.small	167.59	1626.20	40.36	72.73
c1.medium	283.52	1882.17	42.10	62.79
using both cores, separate file targets, total performance				
c1.medium	264.44	2399.00	41.27	63.38

Table 3. IOR read and write bandwidth (128KB requests, 1MB blocksize, 100 segments) to local disk and to a remote NFS drive. Units are in $MB/s = 10^6 Bytes/s$.

	to /tmp		over NFS	
System	Class A	Class W	Class A	Class W
m1.small	17MB/s	16MB/s	15MB/s	14MB/s
c1.medium	29MB/s	29MB/s	26MB/s	30MB/s
using both cores, performance per core				
c1.medium	28MB/s	28MB/s	21MB/s	26MB/s

Table 4. serial NAS NPB 3.3 BT-IO; Fortran I/O to local disk and to a remote NFS drive, rounded to 2 sig.dig.

2.2.1 I/O performance

To test the I/O subsystem performance we built the latest version of the IOR benchmark (17) in POSIX mode and tested large write and read requests on both the local /tmp disk as well as the remotely mounted home directory (where the home disk was on a standard "c1.small" instance). The results (after an fsync to negate any O/S filesystem cache effects), which present performance expectations when memory is being fully utilized, can be seen in Table 2.2.1. They show an appreciable difference between the write and the read performance of the standard and the High-CPU instances to/from local disk. NFS performance is essentially the same in all cases and utilizing both cores increases read performance and decreases write performance (in both cases by a small amount). In fact, while the read performance from local disk appears to be close between the two instance types, most measurements (IOR reports the best ones) were in the range of 800MB/s for the standard one.

In order to further explore what this means with respect to actual scientific applications (that do not have such nice I/O patterns) we employed the BT-IO I/O benchmark out of the MPI version of the NPB (compiled as the serial ones previously with dummy MPI routines, set for 1 processor). We run the tests both in the local /tmp and over NFS to see the difference which for this test, appears to be negligible.As seen in Table 2.2.1, the I/O performance of the *c1.medium* instances is about double that of the standard instances and this remains close to true (with a drop per core of about 20%) even when both processes on the virtual node are doing I/O.

Summarizing, it appears that the High-CPU medium instance is not five times faster than the standard instance for the benchmarks we tested. It is in fact more deficient in terms of memory bandwidth (that being a side effect of the architectural difference between the Opteron and Core2 cpus) and between 2-2.5 times faster on the NAS Parallel benchmarks. Local disk I/O performance also tends to be higher, by an approximate factor of $1.5 - 2.0$. Given that this instance type costs twice as much, it appears that it is cost-effective provided a code is not overly memory bandwidth dependent.

3. MPI on EC2

To use EC2 for anything other than bag-of-tasks type of scientific applications, one would need to use some parallel middleware. Most parallel scientific applications today employ MPI (25) and therefore we built and installed several MPI implementations and tested them for performance and correctness. One major caveat for using EC2 is that we have to use a "free" MPI implementation so that our dynamically generated clusters are tied to a given license. We tested the most recent versions of LAM (7.1.2) (5), MPICH2 (1.0.7) (13), OpenMPI (1.2.6 and an alpha version of 1.3 - 1.3a1r19110) (11) and GridMPI (2.1.1) (12).

Of all the MPI implementations, the only one not to function at all was OpenMPI 1.2.6 that refused to work outside a single box. This was traced to a check that the OpenMPI code does to ensure all nodes are in the same subnet and can be assumed to have routes to each other. The way EC2 is setup, even when one asks for multiple node instantiations in one attempt, there is no guarantee that all instances booted will be in the same subnet and in fact the vast majority of the time they are not. We had to build an alpha version of OpenMPI to circumvent this problem. We initially built MPICH2 using the newer, high performance "Nemesis" (4) CH3 device but after witnessing bad performance we rebuilt it for the basic sockets configuration.

We summarize in Table 3 the basic performance parameters of the different MPI runtimes as measure by the corresponding OMB-3.1 benchmarks (18). Both OpenMPI and MPICH2-nemesis exhibited absolutely horrible latency (which was very wildy varying in the $300 - 600\mu s$ range for small message sizes, see Figure 3). Their asymptotic bandwidths were also very poor. No explanation for this behavior is available to us at this time. GridMPI, LAM and MPICH2-sock on the other hand exhibited similar (and smoother) performance characteristics. in all cases a local, gigabit ethernet based, cluster had almost 100% higher asymptotic bandwidth and less than half of the small message latency.

The results show the Xen virtualization layer at best gives us network latencies in the range of $80\mu s$ and bi-directional bandwidths in the range of $80MB/s$. This is actually rather respectable latency-wise (many of the first Beowulf clusters had latencies above $100\mu s$). However, more recent generation gigabit ethernet clusters achieve latencies of $\approx 30\mu$s. The fact that EC2 only offers "availability zones" and not the option of getting the same subnet coupled with the fact that there is no way to know whether instances are sharing the same node (and network bandwidth to/from it) further intro-

System	latency	uni-bw	bi-bw
LAM	81.20μs	57.85MB/s	81.98MB/s
GridMPI	83.46μs	54.60MB/s	77.07MB/s
MPICH2 nemesis	300μs	15.72MB/s	26.08MB/s
MPICH2 sock	85.87μs	58.49MB/s	83.42MB/s
OpenMPI	300μs	16.44MB/s	17.99MB/s
LAM/ACES	35.83μs	117.64MB/s	198.59MB/s

Table 5. Basic MPI communication parameters (zero size latency, asymptotic uni- and bi-directional bandwidth) on a cluster of standard instances. For comparison we include values from a local physical cluster (ACES) running LAM 7.1.1.

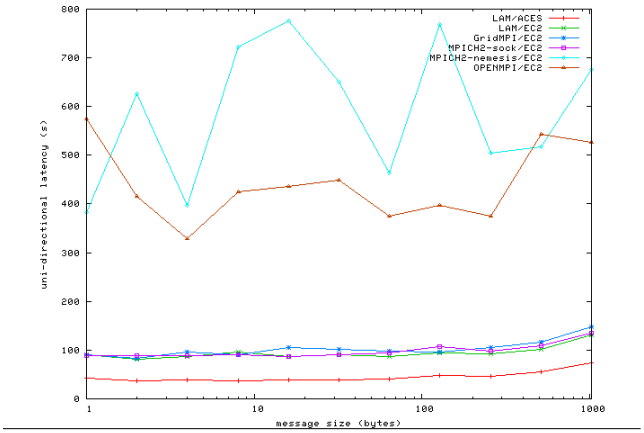

Figure 1. Uni-directional latency comparison

duces performance inconsistencies due to external interference. Some of these could possibly be ameliorated or even done away with by using a Xen-aware MPI implementation (16)alongside appropriately modified Xen domUs. That way shared memory can be used between instances on the same physical host and faster pathways to the network card can be employed. Such efforts however would require Amazon to be involved directly as they control the Xen domUs.

4. Application demonstration

Encouraged by the performance results describe above we implemented a small-scale HPC application configuration. The application we chose is a coupled atmosphere-ocean configuration of the MIT General Circulation Model (MITgcm)(14; 20; 19). Our goal in this exercise was (i) understand the steps required to create an on-demand EC2 based cluster that could run an application typically run on more traditional HPC platforms (ii) demonstrate the software stack needed to take advantage of the sort of interactive, on-demand capabilities such a resource could provide.

4.1 EC2 clusters on demand

In order to use EC2 in an automated way we needed to create and destroy our clusters on demand. At present, there is no standard infrastructure in place targeting an HPC cluster so we had to roll our own using the EC2 API tools and adapting some early work by other practitioners (22). In the absence of cluster AMI combos for EC2 we created our own cluster distribution AMI, based on Fedora Core 7 which we customized for our needs. This allows us to take advantage of the great variety of Earth Science tools and libraries the Fedora Community has been making part of its package tree.

4.2 Optimized Compilers

While modern Linux distributions come with pretty capable versions (v. 4.1+) of the GNU compiler collection, including GFortran which is necessary for our climate codes, there are another two high performance compilers that can be deployed without licensing issues for academics and may perform better, namely Open64 (7) and Sun Studio 12. In fact in the case of the serial NPB results shown in Table 2.2 the latter provides an 11.5% performance boost for the geometric mean of the tests (up to 25% for MG). The only disadvantage to using multiple compilers is that the MPI runtime needs to be rebuilt for the new compiler every time - we have, therefore, been using the modules management system (10) to switch between different software groups.

4.3 Queuing System

While the very nature of an on demand cluster is such that one can use it interactively for running parallel jobs, there is some merit in providing some rudimentary queuing system to allow a set of jobs to be run without the need and wasted time for setting up and tearing down new clusters. Both Torque and GridEngine can be setup in such a way that installation is automated and we have experimented with both.

4.4 Input/Output and Filesystems

The most straightforward way of providing a shared diskspace for parallel processes running on an EC2 cluster on demand is to export home (and other if necessary) directory space over NFS to all nodes. This is a very basic configuration that is always desirable as it enables ease of parallel operation (no need to propagate executables to all nodes etc.) although the performance limitations of NFS would come into play when the number of participating nodes becomes very large. Hence our EC2 instance configuration scripts always export the home directories of the head node to all other nodes.

Depending on the I/O pattern of the parallel program, it might be advantageous to either include the head node in the nodes participating in parallel execution or exclude it to allow all virtual CPU resources to be dedicated to the NFS server threads. Specifically we have seen that the best way to handle the “traditional” I/O through the root process case (N-to-1) is to make that process run on the master node and do local I/O. The other extreme of every process doing its own I/O (N-to-N) appears to be much better served by excluding the master node to avoid competition for the virtual cpu

between NFS and compute threads/processes. However in practice, for large enough process counts the process load on the master node rises to a very large number and makes it essentially unusable. In that case one might want to invest in a different (but more expensive) instance type for the master, with higher I/O performance and more processor cores to handle the NFS load.

At further cost one can experiment with PVFS2 (24) or GlusterFS (6) on demand parallel filesystems by adding extra nodes to host them. They are easily implemented and server/client processes but in practice and under significant I/O stress both the server as well as the client processes can grab a full processor core. In that sense they are counter indicated for use with a standard instance type and when used with a higher instance type one should try and reserve a core per node for the parallel filesystem client. GlusterFS does offer a Non Uniform File Access (NUFA) policy that places entire files on the disk connected directly to the machine the process writing the file is running on - this scenario offers the best opportunities for high performance for the N-to-N I/O strategy. While it is most certainly not a high performance option, we found the use of SSHFS (15) to crossmount remote and EC2 filespace invaluable for easier setup of code and visualization of results.

4.5 Security Issues

It has to be noted that use of EC2 on demand clusters can introduce further security concerns - a more traditional physical cluster has hardened login nodes that are exposed to the outside world and the rest of the nodes (compute and storage) are usually behind a firewall on their own private subnet. EC2 offers the use of security groups to control firewall settings per instantiation of a node. If one's cluster is expected to have a significant (ie. days as opposed to hours) lifetime, care in assigning open ports becomes far more important and all rules should be as restrictive (in terms of allowed IP numbers) as possible. General rules that open ranges of ports to the whole of the outside world are tempting as they make using the cluster easier but may invite trouble. In fact there is also the possibility of an attack originating from within EC2 as well (another user) but that is considered to be unlikely.

4.6 Interactive Usage with an HPC code

We've experimented with two major ways one would use an EC2 cluster on demand for running scientific parallel codes: The most straightforward approach is to setup a cluster, use it (interactively/batch) and then terminate it. It is possible however, as describe reported in (23), to setup a cluster and connect its nodes as additional nodes to an existing cluster under a queuing system. In this paper we focus on the first, interactive, approach. To exploit this approach we have enhanced an automated validating GUI generation tool Legend (8; 9) we developed, to include the setup of an on demand EC2 cluster to the configuration of the build and runtime parameters of our climate, atmosphere and ocean science codes. A screenshot of a panel from this system is shown in figure 4.6. This approach allows a researcher that does not have any understanding of how EC2 works to setup a cluster, build and run a code within the selected cluster image through a validating user friendly interface. Legend can run either on the master EC2 node or locally, using SSHFS to share files with the EC2 nodes. Using this approach we have built and run a standard cubed sphere (approximately 2.8°) resolution coupled ocean-atmosphere model on both a local cluster gigabit ethernet based cluster with dual socket 2.8GHz Pentium4 (Xeon) processors and an equivalent size standard instance based EC2 cluster. The times per timestep observed were approximately 2 secs per timestep on both clusters showing that the cheapest EC2 cluster can do equally well with our local cluster. At this performance, and resolution an EC2 cloud with twelve processors can be used, on demand, to simulate five years of atmosphere-ocean behavior in a week of wall-clock time.

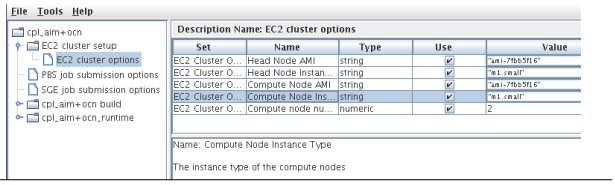

Figure 2. Interface panel provided by Legend that allows a pre-configured virtual cluster image (AMI) to be requested. The AMI contains all the software required to configure, run and diagnose a simulation.

5. Conclusion and Future Research

We described a combination of (i) an EC2 computing cloud on-demand cluster system with adequate performance and (ii) a custom AMI software image designed to make the system use as easy as possible This combination offers a compelling case for cloud computing. Performance is below the level seen at dedicated, supercomputer centers, however, performance is comparable with low-cost cluster systems. Significant performance deficiency arises from messaging performance where latencies and bandwidths are between one and two orders of magnitude inferior to big computer center facilities. Nevertheless, the results are encouraging. It is possible to envisage cloud systems more closely targeted to HPC applications, that feature a specialized interconnect such as Myrinet or Infiniband (with appropriate optimizations MPI and OS virtualization layers). In conjunction with smart usability tools, such as the Legend tool used here, this could be a very powerful addition to the HPC landscape in the very near future.

References

[1] Amazon Inc. Amazon Web Services EC2 site. http://aws.amazon.com/ec2, 2008.

[2] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. Technical report, The International Journal of Supercomputer Applications, 1991.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[4] Darius Buntinas, Guillaume Mercier, and William Gropp. Design and evaluation of nemesis, a scalable, low-latency, message-passing communication subsystem. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 521–530, Washington, DC, USA, 2006. IEEE Computer Society.

[5] Greg Burns, Raja Daoud, and James Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[6] GlusterFS Developers. The Gluster web site. http://www.gluster.org, 2008.

[7] Open64 Developers. The Open64 web site. http://sourceforge.net/projects/open64, 2008.

[8] C. Evangelinos, R.C. Chang P. Lermusiaux, S. Geiger, and N.M. Patrikalakis. Web-enabled configuration and control of legacy codes: An application to ocean modeling. *Ocean Modleing*, 13(3-4):197–220, 2006.

[9] C. Evangelions and C. Hill. A schema-based paradigm for facile description and control of a multi-component parallel, coupled atmosphere-ocean model. In *Proceedings of the 2007 symposium on Component and framework technology in high-performance scientific computing*, pages 83–92, 2007.

[10] John L. Furlani and Peter W. Osel. Abstract yourself with modules. In *LISA '96: Proceedings of the 10th USENIX conference on System administration*, pages 193–204, Berkeley, CA, USA, 1996. USENIX Association.

[11] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[12] GridMPI Group. Grid MPI website. http://www.gridmpi.org, 2008.

[13] William Gropp. Mpich2: A new start for mpi implementations. In *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, page 7, London, UK, 2002. Springer-Verlag.

[14] C. Hill and J. Marshall. Application of a Parallel Navier-Stokes Model to Ocean Circulation. In *Proceedings of Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers*, pages 545–552, 1995.

[15] Matthew E. Hoskins. Sshfs: super easy file access over ssh. *Linux J.*, 2006(146):4, 2006.

[16] W. Huang, M. Koop, Q. Gao, and D.K. Panda. Virtual machine aware communication libraries for high performance computing. In *Proceedings of Supercomputing 2007, Reno, Nevada, November, 2007*, 2007.

[17] Lawrence Livermore National Lab. The IOR benchmark web site. http://sourceforge.net/projects/ior-sio, 2008.

[18] Jiuxing Liu, Balasubramanian Chandrasekaran, Jiesheng Wu, Weihang Jiang, Sushmitha Kini, Weikuan Yu, Darius Buntinas, Peter Wyckoff, and D K. Panda. Performance comparison of mpi implementations over infiniband, myrinet and quadrics. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 58, Washington, DC, USA, 2003. IEEE Computer Society.

[19] J. Marshall, A. Adcroft, C. Hill, L. Perelman, and C. Heisey. Hydrostatic, quasi-hydrostatic and nonhydrostatic ocean modeling. *J. Geophys. Res.*, 102, C3:5,753–5,766, 1997b.

[20] J. Marshall, C. Hill, L. Perelman, and A. Adcroft. Hydrostatic, quasi-hydrostatic and nonhydrostatic ocean modeling. *J. Geophys. Res.*, 102, C3:5,733–5,752, 1997a.

[21] J. McCalpin. The STREAM benchmark web site. http://www.cs.virginia.edu/stream/, 2005.

[22] J. Murty. *Programming Amazon Web Services*. O'Reilly Press, 2008.

[23] Hedeby Project. The Hedeby Project web site. http://hedeby.sunsource.org, 2008.

[24] Robert Ross and Robert Latham. Pvfs: a parallel file system. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 34, New York, NY, USA, 2006. ACM.

[25] Marc Snir, Steve W. Otto, David W. Walker, Jack Dongarra, and Steven Huss-Lederman. *MPI: The Complete Reference*. MIT Press, Cambridge, MA, USA, 1995.