Supercomputing Frontiers and Innovations

2015, Vol. 2, No. 3

Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

Editorial Board

Contents

Foreword to the Special Issue of International Journal of Supercomputing Frontiers and Innovations

The first, inaugural conference "Supercomputing Frontiers 2015" took place in Singapore on March 17–20, 2015. The objective was to celebrate our soon to be launched Singapore National Supercomputing Centre, which will, for the first time, deliver Petascale computational resources to Singaporean scientists, academics, students and industrial researchers.

Additionally, we wished to bring some of the most highly recognised Supercomputing authorities to our community of users — to expose our community to the past achievements and to the visionary ideas of those exceptional trendsetters. The conference was designed to explore global trends and innovations in high performance computing in convolution of the following important areas:

1. Supercomputing applications in domains of critical impact and especially those requiring computer resources approaching Exascale;
2. Big Data science merging with Supercomputing with associated issues of I/O, high bandwidth of networking, storage, workflows and real time processing;
3. Architectural complexity of Exascale systems with special focus on supercomputing interconnects, interconnect topologies and routing, and interplay of interconnect topologies with algorithmic communication patterns for both numerically intensive computations and Big Data; and
4. Any other topics that push the boundaries of Supercomputing to Exascale and beyond.

The conference has gathered more than 390 attendees. There were 50 speakers from around the globe including Australia, Brazil, Canada, China, France, Germany, Japan, Korea, Poland, Russia, Switzerland, Singapore and the United States. The conference featured outstanding keynote speakers including J. Dongarra, A. Gara, J. Gustafson, R. Harrison, S. Klasky, S. Matsuoka, T. Sterling, and R. Stevens.

All conference speakers were invited to submit their work to be considered for publication in the International Journal of Supercomputing Frontiers and Innovation (IJSFI). Fifteen papers were pre-selected by the Conference organisers, and submitted to the IJSFI editors for the final round of rigorous peer review by the independent journal referees. Eventually, out of all submitted papers, seven best papers are included in this special issue of the journal. We hope you enjoy it.

"Supercomputing Frontiers 2015" was a very succesful event, and subsequently, encouraged by a positive feedback and favourable recommendations, we decided to turn it into an annual event.

"Supercomputing Frontiers 2016" will be held on March 14–18, 2016, in Singapore again. We promise a very interesting program with many outstanding speakers. We welcome your participation.

Marek Michalewicz
Yuefan Deng

DOI: 10.14529/jsfi150301

Data Exploration at the Exascale

Hank Childs[1]

In situ processing — i.e., coupling visualization routines to a simulation code to generate images in real-time — is predicted to be the dominant form for visualization on upcoming supercomputers. Unfortunately, traditional *in situ* techniques are largely incongruent with exploratory visualization, which is an important activity to enable understanding of simulation data. In response, a new paradigm is emerging: data is transformed and massively reduced *in situ* and then the resulting form is explored *post hoc*. The fundamental tension in this approach is between the extent of the data reduction and the loss in integrity in the resulting data. However, new opportunities, in terms of increased access to data, may blunt this tension and allow for both sufficient data reduction and also more accurate analysis. With this paper, we describe the trends behind "data exploration at the exascale" and also summarize some recent results that confirmed that this new paradigm can produce superior results compared to the traditional one.

Keywords: scientific visualization, high-performance computing, Lagrangian flow analysis.

Introduction

This paper describes the fundamental challenges behind "data exploration at the exascale," the strategy behind the proposed solution, and some recent evidence that supports the merits of this strategy. It is organized as follows:

- Section 1 provides background. Specifically, Section 1.1 describes the high-performance computing trends that will compel the usage of *in situ* processing and Section 1.2 describes the importance of data exploration and why the traditional approach for this exploration is incongruent with *in situ* processing.
- Section 2 gives an overview of the new paradigm for achieving data exploration with *in situ.*
- Section 3 describes a success story using this new paradigm. One of the main lessons from this example is that increased access to data can lead to more accurate analysis and also reduced storage costs.

1. Background

1.1. In Situ

The justification for *in situ* [6] is discussed extensively in the *Report for the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization* [2]: the ability to generate data is going up much faster than the ability to store it, with the limitations in storage being both in I/O bandwidth and in power costs due to data movement. This summary presented here focuses mostly on the I/O costs, as the the I/O subsystem is undergoing a significant change on upcoming supercomputers.

As supercomputers get ever larger, the cost of achieving sufficient I/O bandwidth is, unsurprisingly, increasing. But supercomputing architects have been experimenting with a new approach to decrease this cost. Where the typical approach has a simulation write data directly to a parallel file system (i.e., "spinning disk"), the new approach introduces an additional participant, solid state drives (SSDs) and has the simulation write data to the SSDs instead. The

[1]University of Oregon, Eugene, USA

simulation can then immediately resume, while, concurrently, the data is copied from the SSDs to the file system, shielding the simulation from slow parallel file system performance. Although the SSDs introduce a new cost, they lessen the importance of I/O bandwidth, allowing for the SSDs to be coupled with a slower (and less expensive) parallel file system, providing an overall cost reduction.

To applications, this I/O configuration appears to have two distinct bandwidth characteristics. On write, the bandwidth appears to be good, since it is be accelerated by SSDs. On read, however, the bandwidth will be poor, since the reads are backed by a slower parallel file system and the presence of SSDs can not accelerate this activity.

The write performance on exascale machines, relative to data size, is expected to be comparable to that of petascale machines (taking into accounts SSDs). But the read performance will be at least one order of magnitude less. Further, as shown in [7], I/O is already the bottleneck on massive data sets. As a result, the I/O bottleneck will be even more extreme at the exascale for visualization programs that attempt to load data at its full resolution.

As a result of these trends, *in situ* processing has become increasingly popular with many successful usages in recent years [8, 12, 14, 17, 20]. Further, an additional advantage of *in situ* processing is that it can access all of the simulation data, which has never previously been possible with *post hoc* analysis. Phrased another way, where supercomputing trends are leading simulations to store data less often, *in situ* processing allows for dramatic increases in temporal frequency, equal to that accessible in the simulation code itself.

1.2. Data Exploration

Bergeron argued in [4] that visualization and analysis usage falls into three categories: descriptive, analytical, and exploratory. Bergeron defined descriptive visualization as useful "when the phenomena represented in the data is known, but the user needs to present a clear visual verification of this phenomenon (usually to others)." He described analytical visualization (or directed search) as "the process we follow when we know what we are looking for in the data." Finally, he defined exploratory visualization (or undirected search) as the process we follow when "we do not know what we are looking for; visualization may help us understand the nature of the data by demonstrating patterns in that data."

Descriptive and analytical use cases can often benefit from a priori knowledge, making them ideal for in situ processing. But exploratory visualization can not benefit from a priori knowledge: it is for when "we do not know what we are looking for."

Exploratory analysis is an iterative process. An analyst forms a hypothesis, poses a question to analysis software, interprets the result, and then forms new hypotheses and/or additional questions. The analyst is the part of this loop and his/her decision making process (i.e. forming questions and hypotheses and interpreting results) is the part of the total time to do the exploration. The time spent by the analyst varies greatly: it is sometimes seconds or minutes, but it is more frequently hours, days, or weeks, and it is not uncommon for an analyst to study a simulation for months. Time scales beyond seconds are clearly not a match for in situ processing, since the exascale machine is such an expensive resource to "hold hostage." But exploratory analysis is too important to marginalize when doing exascale computing, as this category is the one responsible for new scientific insights: it directly leads to "new science."

2. New Paradigm: In Situ Reduction and Post Hoc Exploration

The new paradigm resembles the traditional *post hoc* model, in that the simulation writes data to disk and stand-alone programs visualize this data by reading it from disk. However, the new paradigm introduces a key new step to this model: *it substantially reduces the data using in situ processing before writing it to disk* (see fig. 1). With enough reduction, the amount of data to store for *post hoc* processing can become tractable, although actual sizes that are "tractable" will depend on the details of each individual supercomputer.

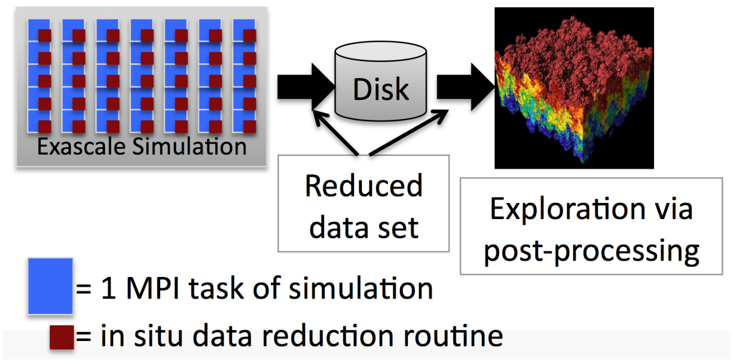

Exascale Simulation
Disk
Reduced data set
Exploration via post-processing
= 1 MPI task of simulation
= in situ data reduction routine

Figure 1. The new paradigm for exploring exascale simulation data via *in situ* transformation and reduction and *post hoc* analysis

Of course, the goals of data reduction and data integrity are in tension. Thinking of a simple compression scheme, too much reduction can sacrifice data integrity, while requiring high data integrity often leaves opportunities for only minimal reduction. So our community must perform significant research to find techniques that balance these tensions. Further, we must constrain ourselves to only considering reduction operators that are viable in an exascale setting.

This new paradigm will represent a significant change for users. Users often distrust any reduction in data; many users believe the integrity of their data can only be preserved if it is displayed or analyzed at its full and native resolution. But this desire is not realistic for exascale computing. I/O and power limitations will restrict how much data can be read in and how much can be stored for subsequent analysis. Given these limitations, users will not be able to continue with "business as usual." This new paradigm is responsive to the fundamental issues, but, ultimately, users will need to accept tradeoffs and guide how decisions are made. Further, significant research is needed to enable users to make informed decisions, e.g., "this level of data integrity comes at the cost of this much time, storage, and power."

More and more research has been devoted to this new paradigm in recent years [11, 15, 16, 18, 19]. A particularly noteworthy research result in this space is ParaView Cinema [3]. With this work, the *in situ* reduction comes from extracting many explorable images, and the *post hoc* exploration is on these images, often in forms that feel interactive for users.

In the following section, we present another research result following this new paradigm, specifically targeting flow visualization. This research result is somewhat different from the other results described previously, in that it makes use of the opportunity provided by *in situ* processing to access more data than ever before, enabling it to create more accurate answers than are possible with a strictly *post hoc* approach.

3. Lagrangian Flow

Doing flow analysis with Lagrangian flow is a relatively new concept for visualization. So, this section begins with an overview of the traditional method for flow analysis (Eulerian flow) in Section 3.1, for the sake of comparison. Section 3.2 then describes the new, Lagrangian method, and Section 3.3 describes results, contrasting them with the traditional method.

3.1. Traditional Method

Particle advection — calculating the trajectory a massless particle follows in a flow field — is foundational for many flow visualization and analysis techniques. McLouglin et al. recently surveyed the state of the art in flow visualization [13], and the large majority of techniques they described, such as line integral convolution [5], finite-time Lyapunov exponents [9], and streamsurfaces [10], depended on advection. Advection assumes access to a vector field, i.e., a continuous function over a four-dimensional domain. If x is the spatial location of a point and t is a time, then the vector field v maps the tuple (x, t) to its velocity as $v(x, t)$.

Advection constructs integral curves, which are continuous functions tangential to the vector field. Each integral curve is called a pathline, and it encodes the trajectory of a single mass-less particle. The path of an integral curve I is the solution to an ordinary differential equation, and is represented as:

$$\frac{d}{dt}I(t) = v(I(t), t) \tag{1}$$

where $I(t_0) = x_0$, for a seed point at time t_0 and location x_0.

For some approaches, visualization techniques focus on the special case of stationary flows which vector fields do not vary over time ("steady state"). With this research, the focus was on the general case: transient flows, where the vector fields are time-varying ("unsteady state").

The traditional method for calculating particle trajectories is not particularly well-suited to exploratory analysis. With *post hoc* analysis, simulations write time slices of data to disk and then this time slice data is explored afterwards. But solving the advection equation requires evaluating the velocity field at many temporal locations. Oftentimes, the necessary time locations are not the ones saved out, so the visualization program instead does a temporal interpolation. This temporal interpolation introduces an error, making the particle follow the wrong trajectory. Further, the increased access provided by *in situ* processing cannot be leveraged by this model when doing data exploration — since the required particles are not known ahead of time, the necessary velocity evaluations cannot be performed, and so the only data that can be used is the time slice data stored for traditional *post hoc* processing.

3.2. Lagrangian Method

Fluid mechanics considers two frames of reference for an observer watching a flow field: Eulerian and Lagrangian. With the Eulerian frame of reference, the observer is at the fixed position and observes flow going by. This is the traditional frame of reference for visualization (i.e., Section 3.1). With the Lagrangian frame of reference, the observer is attached to a particle and moves through space and time. The concept of the Lagrangian frame of reference can be applied to visualization by taking a basis of known trajectories (Lagrangian flows), and then interpolating new particle trajectories from this basis.

Agranovsky et al. [1] explored the Lagrangian approach in the context of *in situ* reduction and *post hoc* exploration (i.e., the new paradigm described in Section 2). The *in situ* transformation and reduction operator placed “basis” particles in the Eulerian vector field and calculated their corresponding trajectories. The storage costs were proportional to the number of particles, so storage reductions could be achieved by limiting the number of these particles. Critically, unlike the traditional/Eulerian method, the Lagrangian method made use of all spatio-temporal data, specifically when calculating the trajectories that their “basis” particles followed. As a result, the spatio-temporal data was encoded into the trajectories, and so subsequent exploration — which happened by interpolating between trajectories — was able to make use of the spatio-temporal data.

3.3. Experiments

Here, we describe experiments comparing Lagrangian and Eulerian techniques. The results presented extend the previous study done by Agranovsky et al.

Three data sets were considered:

- Arnold-Beltrami-Childress (ABC): A three-dimensional analytic vector field from dynamical systems theory, on a regular grid of dimensions $256 \times 256 \times 256$ with 3000 time steps.
- Double Gyre: A common two-dimensional benchmark of two counter-rotating gyres with perturbations over time, on a regular grid of dimensions 512×256 with 3000 time steps.
- Jet: A three-dimensional simulation of a high-speed jet entering a medium at rest, on a regular grid of dimensions $260 \times 520 \times 260$ with 2000 time steps.

Although the frequency a simulation saves state can vary based on many factors, our experiments made the simplifying assumption that a simulation would save at regular intervals, i.e., “every N^{th} cycle.” We then considered six different scenarios for how often the simulation code saved state: 10, 20, 30, 40, 50, and 60 cycles. We refer to the rate a simulation saves its data as the “storage frequency.”

For a given data set and a given storage frequency, we calculated the following information:

- Lagrangian basis trajectories. Particles were placed at even spatial intervals and allowed to advect for the duration of the storage frequency. The resulting displacement (from start to end) was then saved.
- Eulerian time slices, i.e., traditional vector field information at the current time slice.
- Baseline particles. Particles were placed in the flow and their trajectory was calculated. These particles, although calculated in the same way as the Lagrangian basis trajectories, were kept separate, to serve as a baseline.

Then we wanted to compare error between the Lagrangian and Eulerian techniques against the baseline particles. We defined an error metric, which was set to be the difference between the calculated end position (whether Lagrangian or Eulerian) versus the actual end position for that baseline particle. The distances were normalized by the scale of the mesh into units of cells of sizes.

Fig. 2 contains the results of the study. While error increases for both methods as the storage frequency gets larger, the Lagrangian technique is consistently more accurate than its Eulerian counterpart. Further, the Lagrangian technique is still more accurate when reducing the number of basis flows used, meaning that the technique can be both more accurate and take less storage compared to the traditional Eulerian approach.

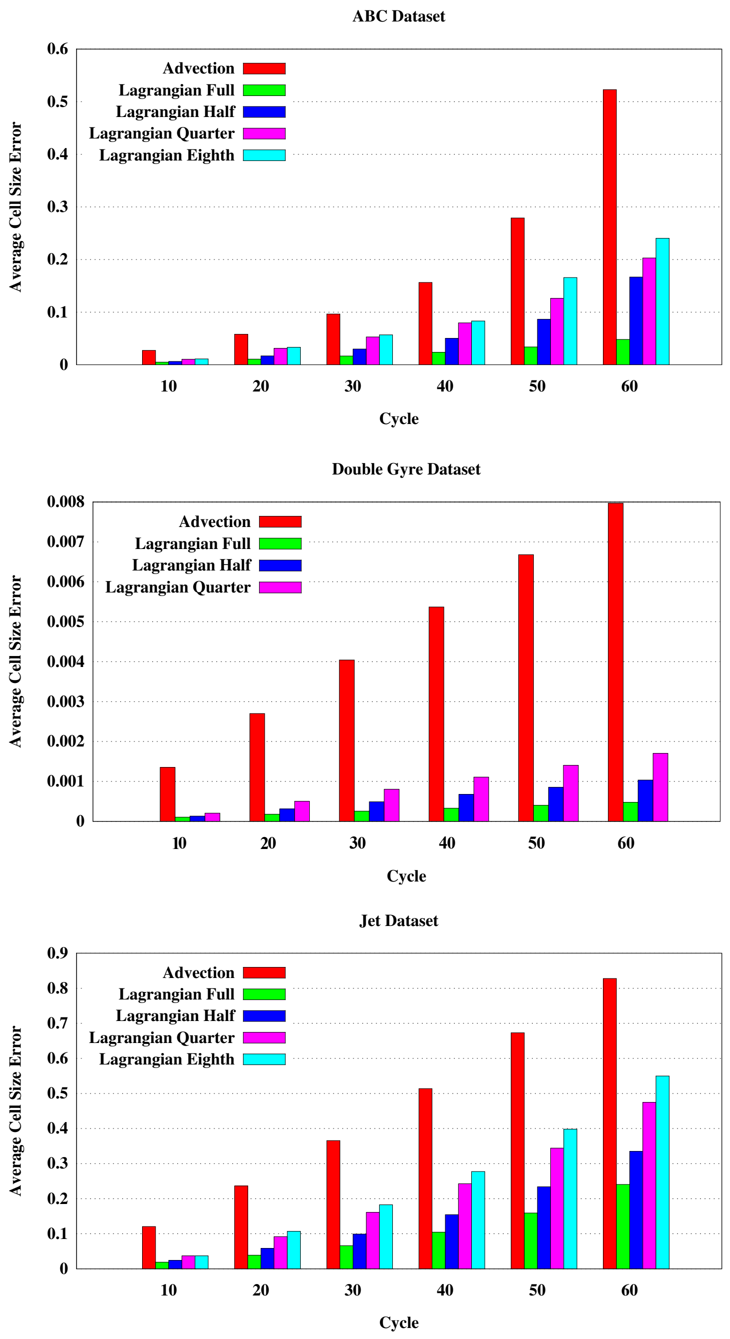


Figure 2. Comparison of Eulerian and Lagrangian techniques

The study varies over three factors: data set, storage frequency, and the number of Lagrangian basis flows. The graphs are organized by data set, and then grouped left-to-right by storage frequency. Traditional Eulerian advection is colored red. When the number of Lagrangian basis flows takes the same storage as the Eulerian method does for saving time slices, then we denote this as "Lagrangian Full" and color the results green. When there are half as many basis flows, and so the storage costs are half that of the Eulerian method, then we denote this "Lagrangian Half" and color the results blue. One-quarter and one-eighth variants are purple and cyan, respectively. In all cases, the results show the average error in the end position over a set of baseline particles, meaning that bigger numbers are worse. This error is normalized by the size of a cell in each data set's mesh.

Summary

The new paradigm of transforming and reducing simulation data *in situ* and then exploring data *post hoc* has received increased attention for the research community in recent years. This paradigm appears to be responsive to the fundamental drivers in high-performance computing, and has the potential to retain the important use case of data exploration, which is often the activity that realizes the value of a simulation. Further, the access to increased temporal resolution creates the opportunity to do better analysis than was previously possible. The Lagrangian technique described in this paper shows that the benefits from incorporating increased temporal resolution can be substantial. For this example, the traditional method was unable to take advantage of increased spatio-temporal data, but the new method was — and the increased access led to superior results.

Hank Childs is grateful for the support of the U.S. Department of Energy Early Career Award, Contract No. DE-FG02-13ER26150, Program Manager Lucy Nowell. Further, this work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

1. Alexy Agranovsky, David Camp, Christoph Garth, E. Wes Bethel, Kenneth I. Joy, and Hank Childs. Improved Post Hoc Flow Analysis Via Lagrangian Representations. In *Proceedings of the IEEE Symposium on Large Data Visualization and Analysis (LDAV)*, pages 67–75, Paris, France, November 2014. DOI: 10.1109/ldav.2014.7013206.

2. Sean Ahern, Arie Shoshani, Kwan-Liu Ma, Alok Choudhary, Terence Critchlow, Scott Klasky, Valerio Pascucci, Jim Ahrens, E. Wes Bethel, Hank Childs, Jian Huang, Kenneth I. Joy, Quincey Koziol, Jay Lofstead, Jeremy Meredith, Ken Moreland, George Ostrouchov, Mike Papka, Venkat Vishwanath, Matthew Wold, Nick Wright, and K. John Wu. Scientific Discovery at the Exascale: Report for the DOE ASCR Workshop on Exascale Data Management, Analysis, and Visualization, July 2011.

3. James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H. Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press. DOI: 10.1109/sc.2014.40.

4. David M. Butler, James C. Almond, R. Daniel Bergeron, Ken W. Brodlie, and Robert B. Haber. Visualization reference models. In *Proceedings of the 4th conference on Visualization '93*, VIS '93, pages 337–342, Washington, DC, USA, 1993. IEEE Computer Society.

5. Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 263–270, New York, NY, USA, 1993. ACM. DOI: 10.1145/166117.166151.

6. Hank Childs, Kwan-Liu Ma, Hongfeng Yu, Brad Whitlock, Jeremy Meredith, Jean Favre, Scott Klasky, Norbert Podhorszki, Karsten Schwan, Matthew Wolf, Manish Parashar, and Fan Zhang. In Situ Processing. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pages 171–198. October 2012. DOI: 10.1201/b12985-12.

7. Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther Weber, and E. Wes Bethel. Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications (CG&A)*, 30(3):22–31, May/June 2010. DOI: 10.1109/mcg.2010.51.

8. Nathan Fabian, Kenneth Moreland, David Thompson, Andrew Bauer, Pat Marion, Berk Geveci, Michel Rasquin, and Kenneth Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89–96. IEEE, 2011. DOI: 10.1109/ldav.2011.6092322.

9. G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248 – 277, 2001. DOI: 10.1016/s0167-2789(00)00199-8.

10. J. P M Hultquist. Constructing stream surfaces in steady 3d vector fields. In *Visualization, 1992. Visualization '92, Proceedings., IEEE Conference on*, pages 171–178, Oct 1992. DOI: 10.1109/visual.1992.235211.

11. Henry Lehmann and Bernhard Jung. In-situ multi-resolution and temporal data compression for visual exploration of large-scale scientific simulations. In Hank Childs, Renato Pajarola, and Venkatram Vishwanath, editors, *4th IEEE Symposium on Large Data Analysis and Visualization, LDAV 2014, Paris, France, November 9-10, 2014*, pages 51–58. IEEE, 2014. DOI: 10.1109/ldav.2014.7013204.

12. Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th international workshop on Challenges of large applications in distributed enviro nments*, CLADE '08, pages 15–24, New York, NY, USA, 2008. ACM. DOI: 10.1145/1383529.1383533.

13. Tony McLoughlin, Robert S. Laramee, Ronald Peikert, Frits H. Post, and Min Chen. Over Two Decades of Integration-Based, Geometric Flow Visualization. In *EuroGraphics 2009 - State of the Art Reports*, pages 73–92, April 2009.

14. Kenneth Moreland, Ron Oldfield, Pat Marion, Sebastien Jourdain, Norbert Podhorszki, Venkatram Vishwanath, Nathan Fabian, Ciprian Docan, Manish Parashar, Mark Hereld, et al. Examples of in transit visualization. In *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, pages 1–6. ACM, 2011. DOI: 10.1145/2110205.2110207.

15. A. Tikhonova, C. Correa, and Kwan-Liu Ma. Visualization by Proxy: A Novel Framework for Deferred Interaction with Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1551–1559, 2010. DOI: 10.1109/tvcg.2010.215.

16. A. Tikhonova, Hongfeng Yu, C. Correa, and Kwan-Liu Ma. A Preview and Exploratory Technique for Large Scale Scientific Simulations. In *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization*, pages 111–120, 2011.

17. V. Vishwanath, M. Hereld, and M.E. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9–14, 2011. DOI: 10.1109/ldav.2011.6092178.

18. Chaoli Wang, Hongfeng Yu, and Kwan-Liu Ma. Importance-Driven Time-Varying Data Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1547–1554, 2008. DOI: 10.1109/tvcg.2008.140.

19. Chaoli Wang, Hongfeng Yu, and Kwan-Liu Ma. Application-Driven Compression for Visualizing Large-Scale Time-Varying Data. *IEEE Computer Graphics and Applications*, 30(1):59–69, January/February 2010. DOI: 10.1109/mcg.2010.3.

20. Brad Whitlock, Jean M Favre, and Jeremy S Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics conference on Parallel Graphics and Visualization*, pages 101–109. Eurographics Association, 2011.

Received July 9, 2015.

DOI: 10.14529/jsfi150302

InfiniCloud: Leveraging the Global InfiniCortex Fabric and OpenStack Cloud for Borderless High Performance Computing of Genomic Data

Kenneth Ban[1,2]***, Jakub Chrzeszczyk***[3]***, Andrew Howard***[3]***, Dongyang Li***[3]***, Tin Wee Tan***[2,4]

InfiniCloud is a geographically distributed, high performance InfiniBand HPC Cloud which aims to enable borderless processing of genomic data as the part of the InfiniCortex project. This paper provides a high-level technical overview of the architecture of InfiniCloud and how it can be used for high performance computation of genomic data in geographically distant sites by encapsulation of workflows/applications in Virtual Machines (VM) coupled with on-the-fly configuration of clusters and high speed transfer of data via long range InfiniBand.

Keywords: Genomics, Cloud-Computing, InfiniBand, Trans-continental, Virtualization, SR-IOV, OpenStack, HPC.

Introduction

The advent of big data has driven the need for flexible high performance computing platforms in order to analyze large amounts of data using user defined reproducible workflows, particularly in the emerging field of genomics and healthcare informatics. These workflows typically require a specific stack of applications with their operating system specific dependencies, which can be different for each pipeline and can frequently change over time as updates are released. In addition to the heterogenous nature of applications, such workflows demand high CPU performance paired with large memory capability as well as a high-performance interconnect for analysis of large genomic/healthcare datasets [9, 11].

In response to this growing need for high performance and flexible computing for analysis of large datasets [8], A*CRC and NCI teams collaborated to define a new cloud computing platform called InfiniCloud, which combines high performance cloud computing powered by OpenStack [6] with the high speed InfiniBand network architecture. This platform was optimized to provide high performance computing with minimal overhead within virtual instances, coupled with native InfiniBand protocol to provide high speed interconnect and transfer of data between the instances.

In cloud computing, resources are presented in a form of virtual machines (VMs). VMs are an abstraction layer which allows hardware resources of a physical system to be presented as number self-contained pools of virtual CPU cores, RAM, storage and network bandwidth that are used to run isolated operating system instances. These resources can be dedicated or shared, depending on performance requirements of the applications running in the cloud environment. The operating image can be created, customized, and versioned by users to ensure that the computing environment is reproducible and flexible. This is particularly important in the field of genomics, where processing pipelines are highly interconnected and can be dependent on a specific version of operating system, kernel, libraries and application binaries. This level of

[1]Institute of Molecular and Cell Biology, A*STAR, Singapore
[2]Dept. of Biochemistry, Yong Loo Lin School of Medicine, National University of Singapore, Singapore
[3]National Computational Infrastructure - The Australian National University, Canberra, Australia
[4]A*STAR Computational Resource Centre (A*CRC), A*STAR, Singapore

flexibility is typically difficult to achieve on a traditional High Performance Computing cluster running multi-purpose system images.

Despite its advantages for reproducible and flexible computing, one major bottleneck in traditional cloud computing platforms is the inefficient and slow transfer of large datasets, commonly encountered in genomic analysis. To address this, we extended the InfiniCloud platform to address the need for efficient high speed transfers by leveraging on the long range Obsidian Longbow E100 InfiniBand extenders, enabling unprecedented high speed transfer of large datasets and VM images across trans-Pacific distances between two geographically distant InfiniCloud platforms in Singapore and Canberra. This capability enables borderless high performance cloud computing by high speed transfer of large datasets together with workflows/applications encapsulated in VMs. The workflows/applications in VMs can be parallelized in virtual instances by the on-the-fly setup of cluster compute nodes, thus opening the door for scaling up reproducible computing environments beyond any one single HPC cloud computing site.

We envision that the InfiniCloud platform combined with long range InfiniBand as part of a global fabric (InfiniCortex) [14] will enable seamless distributed high performance computing amongst geographically distant InfiniCloud nodes, breaking down barriers to meet the challenge of big data computing.

1. InfiniCloud Platform

The InfiniCloud platform was developed on the NCI and A*CRC hardware and is based on OpenStack cloud computing software stack with custom modifications.

1.1. Hardware Components

Currently, InfiniCloud consists of two sites: one located at the NCI (National Computational Infrastructure), in Canberra, Australia and the second at A*CRC, Singapore (fig. 1). The total count of compute cores available is 264, supporting 3TB of memory and a local storage capacity of 15TB (SSD and HDD). All instances are connected to the shared 56Gbit FDR IB fabric.

Figure 1. InfiniCloud sites (left: NCI, Canberra, right: A*CRC, Singapore)

1.1.1. Server Specifications

The overall design of each site is similar, utilizing a common InfiniBand interconnect. The server configurations are detailed in tab. 1 and tab. 2.

Table 1. NCI InfiniCloud configuration

Servers	10x Fujitsu PRIMERGY CX250
CPU	Intel Xeon E5-2650
Memory	256GB
Interconnect	Mellanox FDR
Local storage	1x Intel DCS3500 or 3x Intel DCS3500

Table 2. A*CRC InfiniCloud hardware configuration

Servers	6x SGI C1104-GP1
CPU	Xeon E5-2680
Memory	128GB
Interconnect	Mellanox FDR
Local storage	3x Intel DCS3500 or Micron M600

1.1.2. Local Area Network Components (each site)

The core of InfiniCloud is a global InfiniBand interconnect, which consists of a local FDR switch at each site to connect the local compute nodes, combined with an Obsidian Strategics Longbow E100 range extender connecting the AU and SG InfiniCloud network fabrics (tab. 3 and fig. 2).

Table 3. Network configuration

Switching	FDR IB
Range extender	Obsidian Strategics Longbow E100
Subnet manager	OpenSM (active:AU; standby: SG)

1.1.3. Global Area Network

To enable the global InfiniBand connection, the A*CRC and NCI teams worked closely with AARNet (AU), SingAREN (SG) and Pacific NorthWest GigaPop (PNWGP) in Seattle (USA) to secure a dedicated 10Gbit/s layer 2 link between Canberra and Singapore using spare fibre capacity. Due to the network topology connecting Australia (with the majority of the bandwidth provided to the more densely populated East Coast of Australia), the link was routed via the longer eastern path, crossing the Pacific Ocean twice through PNWGP in Seattle with an RTT of 305ms. In contrast, while exhibiting better delay characteristics more direct western path through Western Australia, Indian Ocean and Guam has limited capacity and is only capable of providing a 1Gbit/s connection (fig. 3).

1.2. InfiniCloud Installation and Configuration

All InfiniCloud systems run the following operating system, drivers and applications stack (tab. 4). Clusters consist of one dedicated management node, one dedicated controller node and

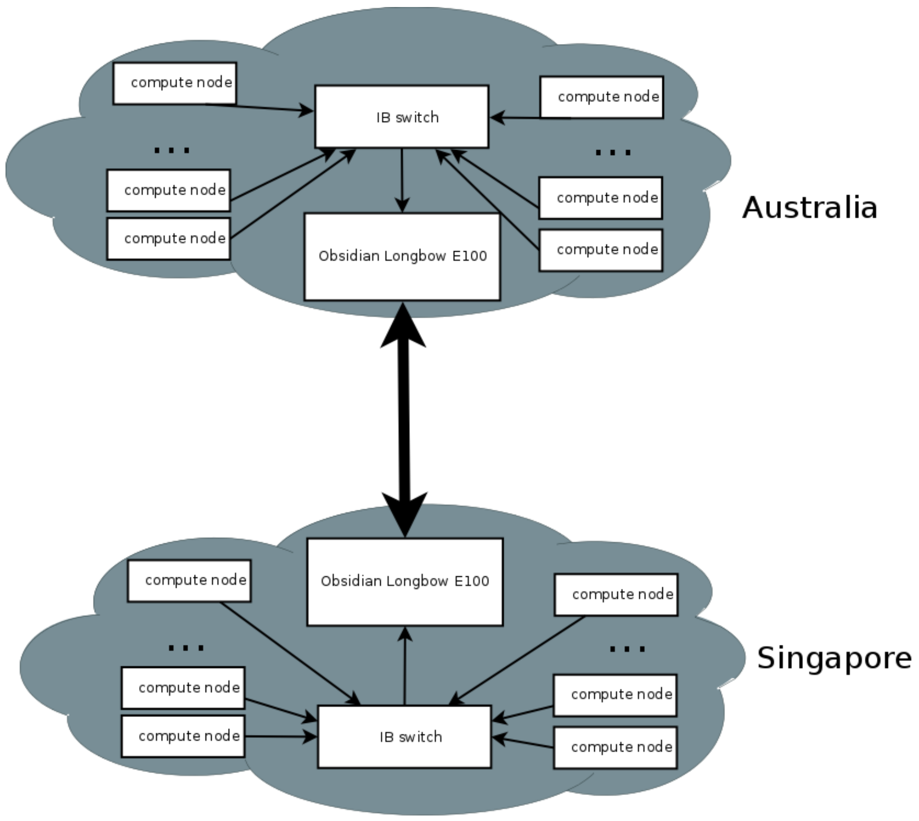


Figure 2. InfiniCloud Network Topology

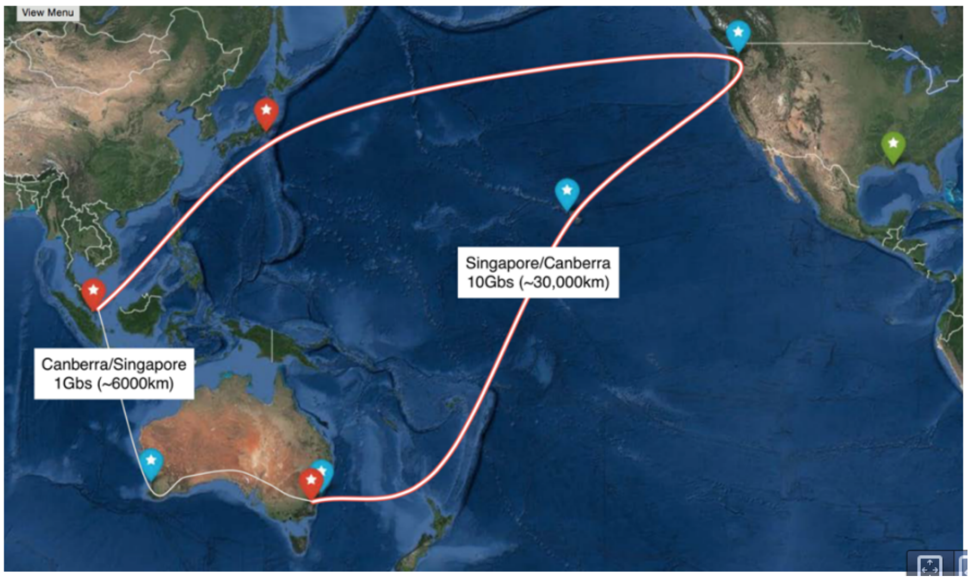


Figure 3. InfiniCloud wide-area networking

Table 4. InfiniCloud software stack

Operating System	CentOS 6.6 x86_64
InfiniBand drivers	Mellanox OFED 2.4
OpenStack version	Icehouse + InfiniCloud specific patches

a variable number of compute nodes (ranging from 4-8). All node classes are integrated to form a fully featured HPC Cloud.

The management node is used for bare metal provisioning and cluster-wide configuration. The controller node provides API, CLI and GUI access to the cloud and is responsible for managing all the core areas of cluster operation: identity management, scheduling, VM image storage, network management and providing an orchestration layer. Compute nodes provide CPU, RAM, storage and high performance SR-IOV networking [12] to the virtual instances. SR-IOV networking support is a requirement for enabling InfiniBand capability in virtual instances.

Building the InfiniCloud cluster required a high degree of customization in order to enable native InfiniBand capability in virtual instances, as well as to provide access to the global InfiniBand network connecting Australia and Singapore. Tab. 5 and fig. 4 show the list of these modifications: (i) A custom virtual interface module adds support for SR-IOV virtual function networking in the nova-compute component; (ii) an embedded switch module implements linking virtual functions to guests and enforces network access restrictions; and (iii) a custom DHCP server package adds InfiniBand support. On top of this, OpenStack out-of-tree patches were necessary in order to force the use of a single partition key, as required by the global InfiniBand fabric. After installing the additional modules and patches, compute nodes are configured to directly connect the HCA to the upstream network, bypassing the layer 2 agent traditionally present on OpenStack compute nodes, as this functionality is now provided by the embedded switch.

Table 5. InfiniCloud OpenStack Customizations

Neutron Server	enable SR-IOV and native IB capability
Neutron Networker	enable EoIPoIB support
Nova Compute	enable SR-IOV and native IB capability
Neutron Agent	enable SR-IOV and native IB capability
DHCP	enable IPoIB support
eswitchd	enable InfiniCortex global IB connectivity

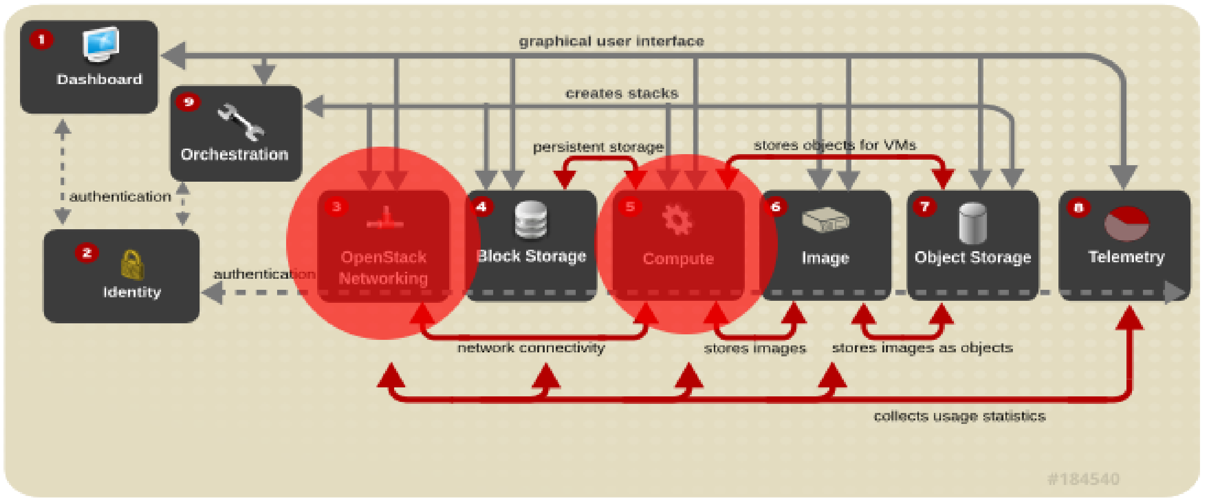


Figure 4. Overview of OpenStack components with customizations highlighted in red. Image adapted from access.redhat.com

2. InfiniCloud InfiniBand Capabilities

After cloud provisioning is complete and all the customizations required for global InfiniBand communications are in place, the system has the ability to provide virtual instances on demand, connected over InfiniBand with full ability to communicate with remote instances using RDMA over a trans-Pacific 10Gbit/s network.

2.1. High bandwidth capability — local connectivity

A high bandwidth capability within a cluster allows for the efficient transfer of data to compute nodes. Listing 1 demonstrates high bandwidth capability (∼6 GB/sec) between 2 virtual instances, close to the line rate on the FDR interconnect:

Listing 1. Local interconnect performance between a pair of virtual machines hosted in Singapore

```
---------------------------------------------------------------------------------
                    RDMA_Write BW Test
 Dual-port       : OFF          Device         : mlx4_0
 Number of qps   : 1            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 TX depth        : 128
 CQ Moderation   : 100
 Mtu             : 2048[B]
 Link type       : IB
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------
 local address: LID 0x05 QPN 0x0a5e PSN 0x90c425 RKey 0xb8011700 VAddr (...)
 remote address: LID 0x1a QPN 0x0cac PSN 0x94503d RKey 0x7001182b VAddr (...)
---------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]    MsgRate[Mpps]
 65536      5000             5984.52            5976.36             0.095622
---------------------------------------------------------------------------------
```

2.2. High bandwidth capability — global connectivity

Integral to the data transfer component is the use of the Obsidian Strategics dsync+ utility [1] which utilizes the RDMA (Remote Direct Memory Access) capabilities to provide long range InfiniBand RDMA transfers between InfiniBand-connected virtual instances. This high performance transfer capability uncouples the need for the data to be located close to the compute nodes, enabling the computing of data to scale beyond a single site.

As a proof-of-concept test of native InfiniBand transfers over long distances, we tested the processing of a large genomic dataset [7] that could be accelerated using large memory compute resources not readily available locally. Listing 2 shows the transfer of 381 GB of genomic data in under 9 minutes from NCI (Canberra, Australia) to A*CRC (Singapore) via the 10G link going through Seattle (∼30,000 km) using the dsync+ utility. In contrast, rsync transfer using TCP/IP over the same 10G link took 3 hours [10].

Listing 2. Global interconnect performance between a pair of virtual machines hosted in Singapore and Australia

```
[root@test01 ~]# dsync --direct-io --option Xfer::RDMA::Buffer-Size=5368709120 \
--option Xfer::RDMA::IO-Block-Size=10485760 \
192.168.200.144:/scratch/kuba/reference_dset/ /scratch/kuba-test/
Finished generating remote file list. 40 files, 3 directores, 381GB.
Finished checking local files. Need to get 40 files, 381GB.
Transfer xfer-ib-rdma network usage 3050B in 0s (10.0kB/s)
Transfer xfer-ib-rdma network usage 381GB in 8m19s (764MB/s)
Done. Transferred 381GB in 8m27s (752MB/s)
```

The remarkable performance observed with long range InfiniBand RDMA provides a significant improvement (∼20 fold) over standard TCP/IP protocols.

3. Using InfiniCloud for Parallelized Workflows in Genomic Analysis

The InfiniCloud platform provides a high performance cloud computing environment for flexible workflows, coupled with unprecedented high speed transfer of big data sets over large geographical distances. A key application that takes advantage of these high performance characteristics is the analysis of genomic sequences which has seen an exponential growth in demand with the advent of next generation sequencing technologies.

The rapid development of next generation of sequencing technologies has dramatically reduced the cost of sequencing genomes [13]. Previously, it took ∼USD $2.7 billion and 10 years to sequence one human genome, but currently the cost has dropped several orders of magnitude to ∼USD $1,000 per genome with the introduction of platforms such as the Illumina HiSeq X sequencer. This drop in cost coupled with the ability to sequence a complete human genome in a few days has driven the adoption of genomic sequencing in research labs as well as hospitals.

Although the cost and speed of sequencing has dramatically improved, the transfer and computation of the genomic data remains a bottleneck in translating that data into the insights needed for improving patient care [10]. Typically, sequencers are not colocated with the compute resources and require the transfer of data in a timely manner. For example, a single Illumina HiSeq X can sequence 32 whole human genomes a week, resulting in ∼6 TB of genomic data. Such volume of data would take over six and a half days to transfer on a dedicated 100Mbps TCP/IP network, assuming ideal conditions and 100 % efficiency. The same transfer could be completed in just under three hours, using long distance InfiniBand [10]. This high performance data transfer capability of native InfiniBand transport would provide the scalability to cope with the growth of genomic data, given the increasing adoption of genomic sequencing in clinical and research labs.

In addition, the computational analysis of genomic data for clinical use requires enforcement of reproducibility standards in addition to the data provenance and security guarantees needed to comply with ethical and legal privacy issues. A computational platform for clinical genomics needs to meet the following challenges:

- High speed data transfers from sequencing data stores to the computational platform
- Reproducible and well documented workflows that can be run on different hardware platforms

- Easy provisioning of compute clusters for processing genomic data from multiple samples using parallel workflows
- High CPU and network performance for rapid analysis of large datasets
- Mechanisms for data provenance and security (e.g. using ephemeral containers) for computation at remote sites

3.1. Provisioning of instances and on-the-fly setup of cluster compute nodes for parallel workflows

To address these challenges, we implemented a software stack on top of the InfiniCloud platform that leverages the use of VM instances or containers to encapsulate workflows, together with automated provisioning of virtual instances and the setup of virtual compute clusters for parallelized workflows.

We adapted ElastiCluster [2] for use on InfiniCloud to enable easy provisioning of instances and setup of virtual clusters for parallel workflows (fig. 5). In our configuration, ElastiCluster was used to provision instances and set up a virtual cluster consisting of a frontend node and a user-defined number of compute nodes. To enable cluster computing for parallel workflows, we configured ElastiCluster to install and setup the SGE job scheduler [5], Ganglia monitoring tools [3], and the IPython notebook interface [4]. The package versions used are listed in tab. 6.

Table 6. Cluster computing stack

IPython	Notebook shell (BASH/Python/R); IPython parallel engine	2.4.1
SGE	Grid engine job scheduler	6.2u5
Ganglia	Cluster monitoring (CPU/Memory/Network)	3.1.7

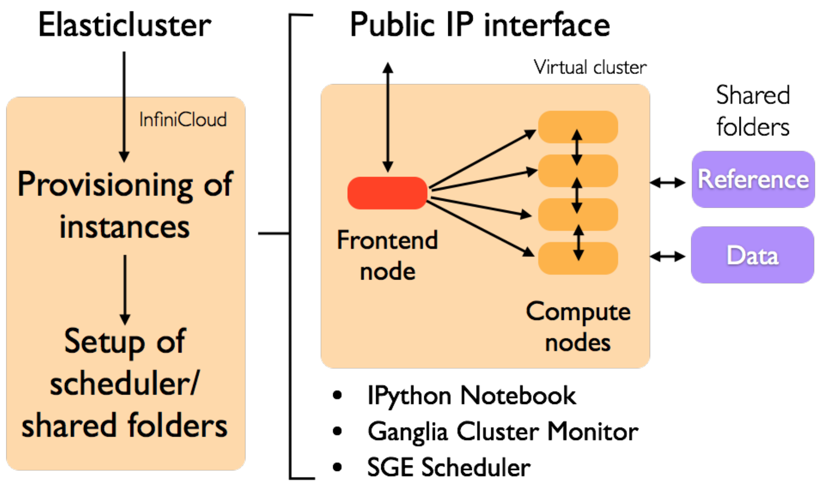
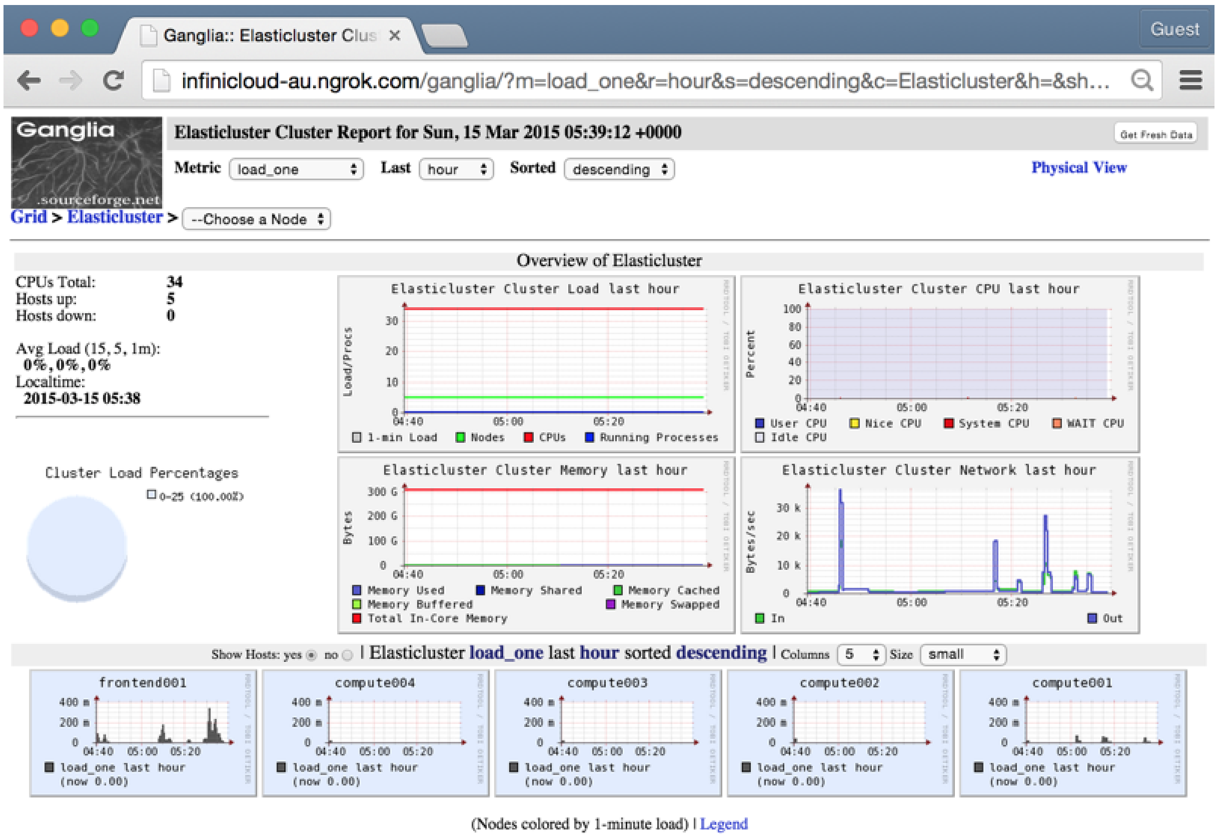


Figure 5. On-the-fly provisioning and setup of virtual clusters for parallelized workflows

In the final configuration, the setup provides SSH access, a web interface for cluster monitoring using Ganglia (fig. 6), and a versatile IPython Web Notebook interface for BASH/Python/R scripting (fig. 7).


Ganglia:: Elasticluster Clus
infinicloud-au.ngrok.com/ganglia/?m=load_one&r=hour&s=descending&c=Elasticluster&h=&sh...
Guest
Get Fresh Data
Physical View
Grid > Elasticluster > --Choose a Node
Overview of Elasticluster
Elasticluster Cluster Load last hour
Elasticluster Cluster CPU last hour
Elasticluster Cluster Memory last hour
Elasticluster Cluster Network last hour
Cluster Load Percentages
0-25 (100.00%)
frontend001
compute004
compute003
compute002
compute001
load_one last hour (now 0.00)
(Nodes colored by 1-minute load) | Legend

Elasticluster Cluster Report for Sun, 15 Mar 2015 05:39:12 +0000

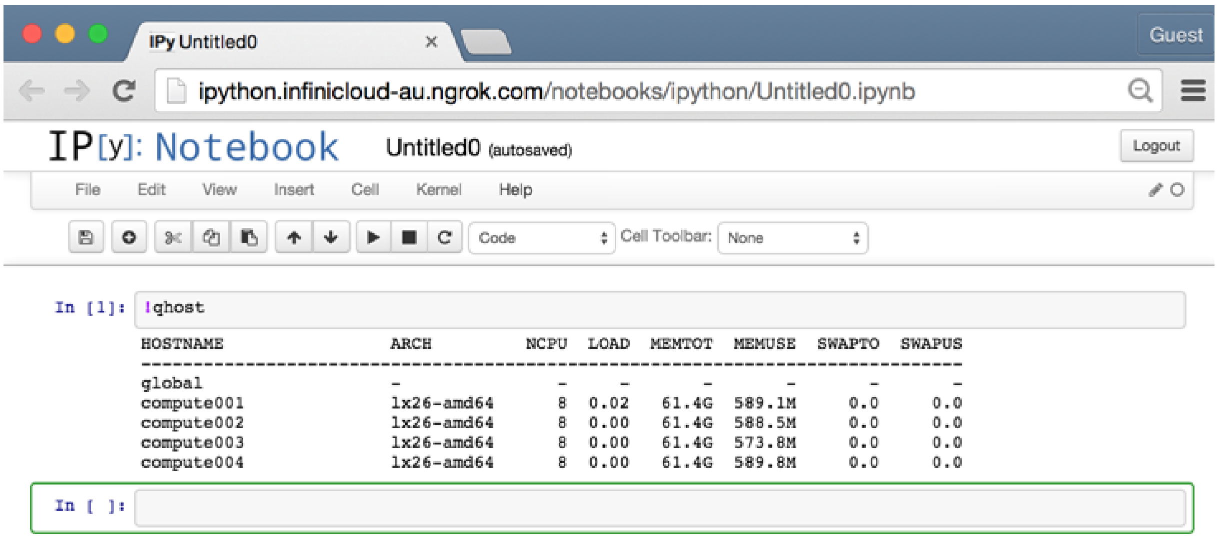
Metric load_one Last hour Sorted descending

CPUs Total: 34
Hosts up: 5
Hosts down: 0

Avg Load (15, 5, 1m):
0%, 0%, 0%
Localtime:
2015-03-15 05:38

Show Hosts: yes no | Elasticluster load_one last hour sorted descending | Columns 5 Size small

Figure 6. Ganglia cluster monitoring



```
In [1]: !qhost
HOSTNAME                ARCH         NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
-------------------------------------------------------------------------------
global                  -               -     -       -       -       -       -
compute001              lx26-amd64      8  0.02   61.4G  589.1M     0.0     0.0
compute002              lx26-amd64      8  0.00   61.4G  588.5M     0.0     0.0
compute003              lx26-amd64      8  0.00   61.4G  573.8M     0.0     0.0
compute004              lx26-amd64      8  0.00   61.4G  589.8M     0.0     0.0
In [ ]:
```

Figure 7. IPython notebook

3.2. Implementation of variant calling genome analysis pipeline

Next, we demonstrate how the on-the-fly provisioning and setup of virtual machines can be used to parallelize a genomic analysis workflow. We chose a clinically relevant workflow, called variant calling, that takes genomic sequences from cancer samples and detects mutations in genes that could be used to determine the prognosis of a patient, or to identify potential chemotherapy drugs that could be used for treatment. As each cancer sample can be analyzed separately, the workflow is amenable to simple asynchronous parallelization without any inter-process communication.

The implementation of genomic workflows typically involves several processing steps that are run using different applications that may vary in complexity of the setup dependencies. The ability to install and run them in a virtual instance allows the different applications to be set up to interoperate properly, then replicated for parallel workflows.

In this workflow, genomic sequences are processed in a pipeline through a series of steps using different applications to identify and annotate mutations (fig. 8). We use the pipeline application to orchestrate the steps in processing and to distribute the processing to the compute nodes using the SGE job scheduler:

1. Genomic sequences from each cancer sample are processed with an aligner — the application that compares the sequences to a human reference genome sequence and identifies the position and alignment of each sequence from the cancer samples.
2. The files from each cancer sample are processed by a variant caller program, which compares the aligned sequences to the human reference genome sequence to identify variations (substitutions, insertions, deletions) in the cancer samples.
3. The variant files from each cancer sample are annotated. A specialized application compares each variation to multiple databases to identify what the potential effects of each mutation have on regions in the genome.

The applications are pre-installed in the VM images together with their dependencies to enable portability between InfiniCloud platforms. The reference datasets required by the aligner, variant caller, and annotation tool are located in a data volume that can be mirrored between InfiniCloud platforms. The genomic dataset is isolated in a separate volume which also stores the results of the analysis (fig. 8); this isolation provides the flexibility for maintaining data provenance and security.

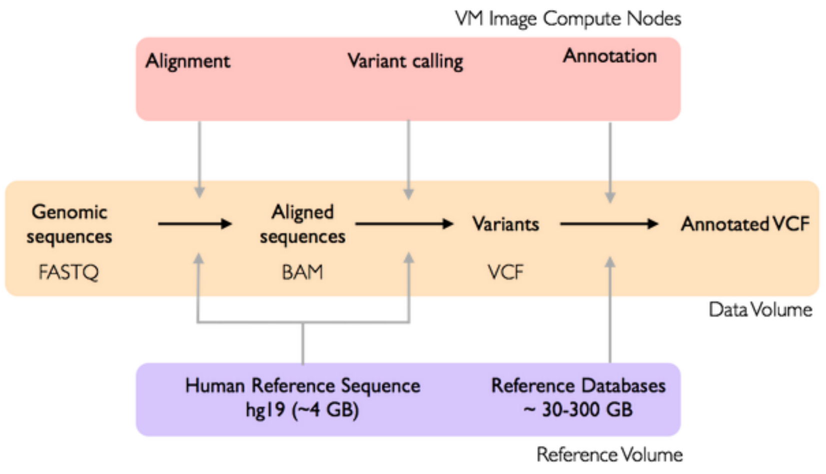


Figure 8. Workflow for variant calling of genomic data from cancer samples

3.3. Demonstration of genome analysis workflow for remote cloud computing

We demonstrate the computation of genomic sequences from multiple cancer samples on the InfiniCloud platform in Canberra, Australia from Singapore by remote provisioning of instances, setup of the cluster, and mounting of reference/data volumes (fig. 9).

- The VM images are mirrored from Singapore to Australia so that both sites have the same application/workflow backends for genomic analysis
- The common reference volume is automatically mirrored from Singapore to Australia and attached to the frontend and compute nodes
- The data volume is synchronized according to a user-defined workflow and attached to the frontend and compute nodes
- Genomic data is transferred from Singapore to Australia for computation
- Results are transferred back from Australia to Singapore
- The data volume on remote site can be deleted in cases where genomic data cannot be stored offsite for data provenance and security reasons

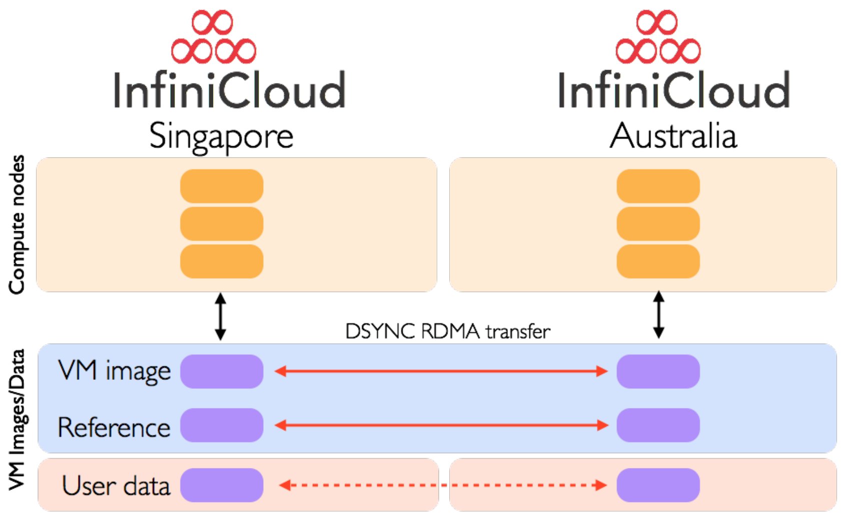


Figure 9. Borderless HPC cloud computing of genomic data across different sites for scalability

For the analysis, the genomic data is first transferred from Singapore into the data volume (Australia) using the dsync+ utility. Here, we achieve a transfer of ~233 GB of data in 5.5 minutes (~696 MB/sec) from Singapore to Australia via Seattle (~30,000 km). The data from multiple cancer samples is then analyzed with the variant calling pipeline. Fig. 10 and fig. 11 show the CPU and network resource utilization during the pipeline run.

As an illustration of the output from the variant calling pipeline, fig. 12 shows mutations detected in a tumour suppressor gene (TP53) in a cancer sample which generally signifies a bad prognosis.

In summary, the high speed data transfer between InfiniCloud platforms can be used to allow scaling beyond a single site to speed up the parallel processing of data in cases where analysis is time-sensitive and/or constrained by local resources. Furthermore, the encapsulation of data and workflows within virtual machines provides one approach to maintain data provenance at the site of origin while harnessing the high performance computational resources at remote sites.

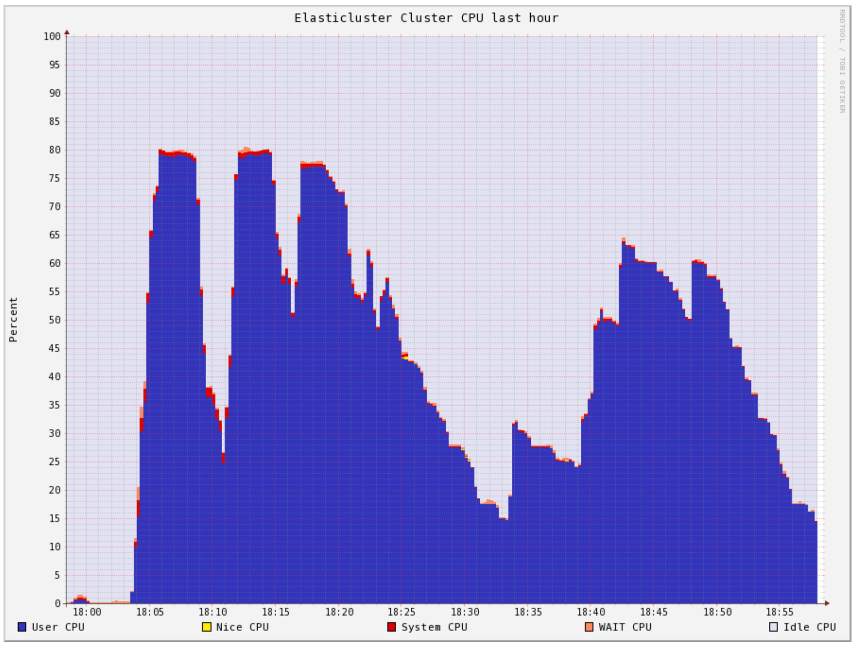


Figure 10. Aggregate CPU load

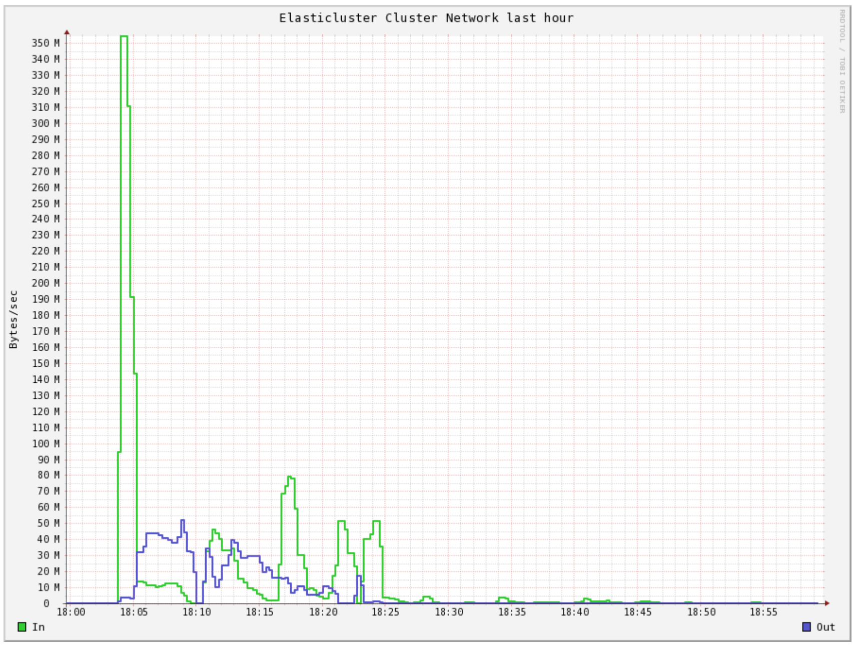
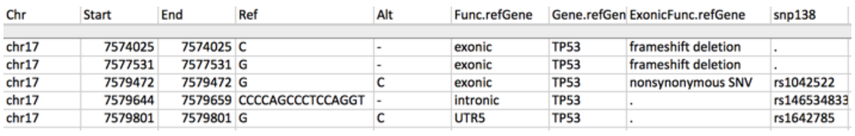


Figure 11. Aggregate network utilization



Chr	Start	End	Ref	Alt	Func.refGene	Gene.refGen	ExonicFunc.refGene	snp138
chr17	7574025	7574025	C	-	exonic	TP53	frameshift deletion	.
chr17	7577531	7577531	G	-	exonic	TP53	frameshift deletion	.
chr17	7579472	7579472	G	C	exonic	TP53	nonsynonymous SNV	rs1042522
chr17	7579644	7579659	CCCCAGCCCTCCAGGT	-	intronic	TP53	.	rs146534833
chr17	7579801	7579801	G	C	UTR5	TP53	.	rs1642785

Figure 12. Mutations detected in the DNA sample

Conclusions

We present a new cloud computing platform called InfiniCloud, which combines high performance cloud computing powered by OpenStack [6] with the high speed/low latency of an InfiniBand network architecture. This platform delivers high performance computing with minimal overhead within virtual instances, coupled with native InfiniBand protocol for high speed interconnect transfer of data between the instances.

In addition, the InfiniCloud platform incorporates long range InfiniBand extension and enables unprecedented high speed transfers of large datasets such as genomic data and VM images across global distances. This capability enables borderless high performance cloud computing that integrates high speed transfer of large datasets together with workflows/applications encapsulated in VMs. This encapsulation allows easy parallelization of virtual instances and on-the-fly instantiation of cluster compute nodes using ElastiCluster.

We envision that the InfiniCloud platform combined with long range InfiniBand as part of the InfiniCortex global InfiniBand fabric [14], will enable seamless distributed cloud-based high performance computing amongst geographically distant InfiniCloud nodes, breaking down borders and illuminating the path to exascale computing to meet the challenge of supporting current and future big data computing needs.

This research was undertaken with the assistance of resources from the National Computational Infrastructure (NCI), which is supported by the Australian Government.

*This work was supported by the A*STAR Computational Resource Centre through the use of its high performance computing facilities.*

References

1. dsync+. `http://www.obsidianresearch.com/products/software/dsync+/index.html`.
2. ElastiCluster. `https://github.com/gc3-uzh-ch/elasticluster`.
3. Ganglia monitoring system. `http://ganglia.sourceforge.net/`.
4. IPython interactive computing. `http://ipython.org/`.
5. Open grid scheduler. `http://gridscheduler.sourceforge.net/`.
6. OpenStack cloud computing platform. `http://www.openstack.org`.
7. Georges A, Li Q, Lian J, O'Meally D, Deakin J, Wang Z, Zhang P, Fujita M, Patel HR, Holleley CE, Zhou Y, Zhang X, Matsubara K, Waters P, Graves JA, Sarre SD, and Zhang G. High-coverage sequencing and annotated assembly of the genome of the Australian dragon lizard Pogona vitticeps. *Gigascience*, 4(45), Sep 2015. DOI: 10.1186/s13742-015-0085-2.
8. Jakub Chrzeszczyk, Muhammad Atif, Joseph Antony, Dongyang Li, Matthew Sanderson, and Allan Williams. Perspectives on implementation of a high performance scientific cloud backed by a 56G high speed interconnect. HPC Advisory Council Event, Singa-

pore, http://www.hpcadvisorycouncil.com/events/2014/singapore-workshop/preso/12_ANU.pdf, November 2014.

9. Marius Hillenbrand, Viktor Mauch, Jan Stoess, Konrad Miller, and Frank Bellosa. Virtual InfiniBand clusters for HPC clouds. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, page 9. ACM, 2012. DOI: 10.1145/2168697.2168706.

10. Andrew Howard. NCI InfiniCloud: Expanding clouds with high speed InfiniBand interconnects. http://www-lk.apan.net/meetings/Fukuoka2015/Sessions/22/NCI_Howard_InfiniCloud_APAN_Fukuoka_TAN.pdf.

11. Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K Panda. A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 125–134. ACM, 2006. DOI: 10.1145/1183401.1183421.

12. Jithin Jose, Mingzhe Li, Xiaoyi Lu, Krishna Chaitanya Kandalla, Mark Daniel Arnold, and Dhabaleswar K Panda. SR-IOV support for virtualization on infiniband clusters: Early experience. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 385–392. IEEE, 2013. DOI: 10.1109/ccgrid.2013.76.

13. E. R. Mardis. A decade's perspective on DNA sequencing technology. *Nature*, 470(7333):198–203, Feb 2011. DOI: 10.1038/nature09796.

14. Tin Wee Tan, Dominic S.H. Chien, Yuefan Deng, Seng Lim, Sing-Wu Liou, Jonathan Low, Marek Michalewicz, Gabriel Noaje, Yves Poppe, and Geok Lian Tan. InfiniCortex: A path to reach Exascale concurrent supercomputing across the globe utilising trans-continental InfiniBand and Galaxy of Supercomputers. Submitted to Supercomputing Frontiers 2015 conference proceedings, Singapore.

Received July 7, 2015.

DOI: 10.14529/jsfi150303

Performance Assessment of InfiniBand HPC Cloud Instances on Intel® Haswell and Intel® Sandy Bridge Architectures

Jonathan Low[1], *Jakub Chrzeszczyk*[2], *Andrew Howard*[2], *Andrzej Chrzeszczyk*[3]

This paper aims to establish a performance baseline of an HPC installation of OpenStack. We created InfiniCloud - a distributed High Performance Cloud hosted on remote nodes of InfiniCortex. InfiniCloud compute nodes use high performance Intel (R) Haswell and Sandy Bridge CPUs, SSD storage and 64-256GB RAM. All computational resources are connected by high performance IB interconnects and are capable of trans-continental IB communication using Obsidian Longbow range extenders.

We benchmark the performance of our test-beds using micro-benchmarks for TCP bandwidth, IB bandwidth and latency, file creation performance, MPI collectives and Linpack. This paper compares different CPU generations across virtual and bare-metal environments.

The results show modest improvements in TCP and IB bandwidth and latency on Haswell; performance being largely dependent on the IB hardware. Virtual overheads were minimal and near-native performance is possible for sufficiently large messages. From the Linpack testing, users can expect the performance in their applications on Haswell-provisioned VMs more than twice. On Haswell hardware, native and virtual performance differences is still significant for MPI collective operations. Finally, our parallel filesystem testing revealed virtual performance coming close to native only for non-sync/fsync file operations.

Keywords: Cloud-Computing, InfiniBand, Trans-continental, Benchmarking, Virtualization, SRIOV, BeeGFS, OpenStack, HPC.

Introduction

Cloud computing offers resources on-demand as an Infrastructure-as-a-Service (IaaS) platform, providing good flexibility in resource allocation and usage that can be easily managed by both end-users and administrators. This brings the benefits of isolated, user-customised software and hardware environments that enable software reproducibility and turn-key solutions and applications regardless of the underlying physical computing hardware. Over the past few years, there has been a shift in utilising such cloud computing services and its associated benefits to address the needs of the HPC scientific community [7].

Since 2009, the National Computational Infrastructure (NCI) in Australia have been providing a cloud computing platform service for compute and I/O-intensive workloads to their big data research community [2]. NCI Cloud services provide computational resources in the form of virtual machines (VM) provisioned by the OpenStack[4] cloud operating system platform. The bare-metal (BM) backend consists of high-spec Intel CPUs, SSDs for storage and started off with 10Gb Ethernet for networking. Encouraged by rapid adoption of the Cloud services, NCI enhanced the interconnect from 10Gb to 56Gb Ethernet using Mellanox hardware together with Single Root IO Virtualisation (SR-IOV) as a first phase. This brings significant performance improvements to traditional HPC applications that typically require a fast interconnect. As the same Mellanox hardware was capable of 56Gb InfiniBand (IB) and SR-IOV, A*CRC and NCI

[1]A*STAR Computational Resource Centre, Singapore
[2]National Computational Infrastructure, ANU, Canberra, Australia
[3]Jan Kochanowski University, Kielce, Poland
[4]OpenStack cloud computing platform - http://www.openstack.org

teams worked together to build InfiniCloud, a native IB OpenStack Cloud prototype which was completed in October 2014 and demonstrated at SC14 in New Orleans, as a part of the InfiniCortex project [11]. In February 2015 A*CRC and NCI teams further enhanced InfiniCloud by addition of six SGI servers located in Singapore. They consist of the latest Intel Haswell CPU models, DDR4 memory, SSD storage and the same Mellanox 56Gb ConnectX-3 hardware. The servers at both NCI and A*CRC can communicate with each other with native InfiniBand through range-extender equipment from Obsidian Strategics. InfiniCloud users can quickly and easily make use of compute resources located at either location, despite the bare-metal hardware residing on a remote site. This allows to distribute the processing of user data, as well as utilizing additional capacity and unique capabilities of hardware located at each site. For example, users can opt for top-performance CPUs in Singapore or larger available memory fat-nodes in Australia.

In this paper we aim to provide an insight into the improved performance that users can expect when moving from the NCI SandyBridge hardware to A*CRC's Haswell servers. Thus we present bandwidth, latency and MPI micro-benchmarks to gauge the VM network performance, storage benchmarks to test the VM storage backend and Linpack benchmarks to gauge real HPC application performance.

The paper is presented as follows. Section 1 explores any past work done and how this paper's work relates to that. Section 2 explains the hardware and software configuration as well as details of the benchmarks performed. Section 3 presents the results obtained, followed by concluding remarks in the last section.

1. Background and Related Work

In the past, virtualised environments incurred significant overheads so that their use for intensive workloads came with significant performance degradation. This started to improve, starting with the introduction of Intel VT to better share resources and improving the performance of CPU, memory virtualisation and more. However, network I/O remained a challenge to obtain near-native performance amongst virtual machines due to the packet processing, switching and CPU interruptions involved. These overheads become very significant when attempting to make use of high speed interconnects that typical HPC workloads require and their associated features such as RDMA that needed to work effectively in virtual environments.

To solve the network I/O problem, the SR-IOV technology was drawn up by the PCI Special Interest Group. This is the hardware-based virtualisation method that allows near-native performance of network interfaces to be realised, where network I/O can bypass the hypervisor to avoid involvement of the CPU. This works for both Ethernet and InfiniBand. Amazon Web Services provide SRIOV-enabled Gigabit Ethernet (GigE) for their C3 instances[5], the feature marketed as "Enhanced Networking" and there have been numerous performance studies for SRIOV-enabled Gigabit Ethernet and InfiniBand usage [3, 6, 8–10].

Today there exists a number of virtual environment installations utilising InfiniBand, ours included. Citing other examples:

- A private cloud platform, "FermiCloud", was used to study SRIOV-enabled, IB-interconnected virtual hosts provisioned using OpenNebula and conducting MPI micro-

[5]Announcing New Amazon EC2 Compute Optimized Instances - http://aws.amazon.com/about-aws/whats-new/2013/11/14/announcing-new-amazon-ec2-compute-optimized-instances

benchmarks and HPL [3]. The hardware used in the study were Intel Westmere CPUs and DDR InfiniBand hardware.

- An in-depth performance study on SRIOV-enabled FDR InfiniBand for virtual clusters examined the behaviour of virtual IB under differing combinations of resource subscriptions, IB progression modes and parallel programming languages [8].
- The San Diego Supercomputing Center will host a Pflop-capable HPC resource with a key aim to

  "Provide a virtualized environment to support development of customized software stacks, virtual environments, and project control of workspaces" [10]

For our exercise, we use the OpenStack cloud operating system to provision resources and test the performance of a SRIOV-enabled InfiniBand virtual cluster on SandyBridge & Haswell CPUs with FDR InfiniBand.

2. Setup

In this section we detail the hardware and software setup and provide details of the benchmarking configuration.

2.1. Setup of NCI SandyBridge and A*CRC Haswell VMs

The hardware details of both bare-metal server types are summarised in tab. 1.

Table 1. Summary of NCI and A*CRC server specifications, provisioned and managed by the OpenStack Icehouse release

	OpenStack (IceHouse) provisioned	
Location	NCI, Australia	A*CRC, Singapore
CPU	2x Intel E5-2650 8-core SandyBridge (SB) Arch.	2x Intel E5-2680v3 12-core Haswell (HW) Arch.
Memory	256GB 1333MHz DDR3	128GB 2133MHz DDR4
Storage	6x 10k RPM Seagate HDD	Intel DC S3500 SSD
Network	Mellanox Connect-X3 FDR	Mellanox Connect-X3 FDR
Operating System	CentOS 6.5	CentOS 6.5
# of compute servers used	4 (2 + 2: BM-BM and VM-VM)	4 (2 + 2: BM-BM and VM-VM)

To compare native and virtual performance, the four servers were used as two pairs, one for BM testing whilst the other was used with one VM instance each. Mellanox OFED[6] drivers v2.4 were used to provide the hardware-based SR-IOV virtualisation of the InfiniBand interface in the form of virtual functions that can be dedicated to particular VM instances. As of March 2015, SR-IOV is a requirement for running InfiniBand in virtual instances on OpenStack. In standalone KVM, non-OpenStack virtualized environments, it is possible to assign the entire HCA to a single virtual machine, enabling InfiniBand connectivity without using SR-IOV. The main

[6]Mellanox OFED - http://www.mellanox.com/page/products_dyn?product_family=26

drawback of such approach is no support for running multiple InfiniBand guests concurrently on a single compute node. For above reasons, this paper focuses on SR-IOV based approach. More information on the direct PCIe passthrough performance compared to SR-IOV can be found from Lockwood's blog [9].

Both resources at NCI and at A*CRC were provisioned by using the OpenStack interface to setup both environments. A major part of the setup effort was for OpenStack to play nicely with the InfiniBand interfaces. To make this possible, three additional modules were installed: A custom virtual interface module adds support for SR-IOV virtual function networking in the nova-compute component. An embedded switch module implements linking virtual functions to guests and enforces network access restrictions. A custom DHCP server adds InfiniBand support. On top of this, a few OpenStack out-of-tree patches were necessary in order to force the use of a single partition key, as required by the InfiniBand range extenders. After installing the additional modules and patches, compute nodes are configured to directly connect the HCA to the upstream network, bypassing the layer two agent traditionally present on OpenStack compute nodes — this functionality is now provided by the embedded switch.

In addition for Haswell servers, CPU passthrough was enforced instead of OpenStack defaulting to the Nehalem CPU architecture. This resulted in a 3-fold speedup in Linpack performance due to the AVX, AVX2 and FMA feature flags present in Haswell over Nehalem.

In addition to the two OpenStack provisioned setups, we utilised an existing non-OpenStack virtual cluster at A*CRC that was already setup using virt-manager and hosts a BeeGFS parallel filesystem [5]. This was used to test the parallel filesystem's performance using both native and virtual metadata & storage target backends on Haswell hardware and was also used for MPI micro-benchmarks.

Each cluster was interconnected to a Mellanox SX-6036 36-port switch and all servers utilised KVM/QEMU as the virtual machine monitor.

2.2. Benchmarking and VM configuration

This subsection details each benchmarking application used in this exercise we used together with the VM configuration for each one where appropriate. To present the possible worst-case scenarios, the highest recorded benchmarks out of several runs on native BM hardware were used whilst the lowest for VMs were recorded. The exceptions are the MPI and storage benchmarks, where we reported the average values.

2.2.1. iperf TCP performance

iperf was used to test the TCP bandwidth available between a pair of nodes. The test was multi-threaded, utilising all cores available on each server (24 on Haswell and 16 on SandyBridge) to saturate the available bandwidth. For the virtual test, each node hosted a single VM with all available cores allocated. The aggregate bandwidth achieved was recorded at the end.

2.2.2. InfiniBand write performance and latency

The *ib_write_bw* and *ib_write_lat*, part of the OFED perftools package, were used to test the RDMA bandwidth performance and latency of the InfiniBand interconnect in both native and virtual instances. The server and VM setup was the same as that for the iperf test. For the bandwidth and latency tests, 64k and 2-byte messages transfers were used respectively.

2.2.3. Linpack

Used to rank supercomputer installations in the Top500, the Linpack benchmark is used to ascertain the performance of a typical HPC application involving computation and communication by solving a dense linear system of equations [4]. Two Linpack benchmark types were used:

- A local Linpack application, namely the Intel Optimized SMP Linpack binary, was used to test on-node performance by solving a problem size with leading dimension of 60k elements. The virtual test used one VM with all cores allocated.
- An MPI-distributed Linpack, namely HPL, was tested on a pair of BM and VM servers. In this case, one MPI process per node was used with multi-threading enabled to utilise all the available cores available within each node. Hence for VM testing, one VM communicated with the other on distinct nodes, giving an idea of the communication performance through the IB interconnect in a virtual setting. A problem size with leading dimension of 120k elements was used together with a blocking size of 168 elements.

2.2.4. MPI Ping-Pong, Alltoall and Barrier microbenchmarks

Using the Intel MPI Benchmarks v4.0.0 package[7], we looked at the performance of message-exchange for a range of message sizes using the Ping-Pong test and the performance of MPI collective operations that involve the synchronisation of many MPI processes on Haswell hardware.

For this benchmark, three non-OpenStack servers were used where each VM was allocated one CPU socket of 12 cores and 64GB memory. The bare-metal test utilised the other, unallocated CPU socket and 64GB of memory available. Tab. 2 summarises the allocated resources.

Table 2. Summary of server specifications of the non-OpenStack A*CRC machines allocated to a VM or BM instance

VM Manager	virt-manager
CPU	Intel E5-2680v3 12-core Haswell (HW) Arch.
Memory	64GB 2133MHz DDR4
Storage	3x 512GB Micron M600 SSD 1x 1TB 10k RPM WD HDD
Network	Mellanox Connect-X3 FDR
Operating System	CentOS 6.5
# of servers	3: Arranged as 3 VM or BM instances, each with the resources stated above

The Ping-Pong test used one MPI process on two distinct nodes whilst all available cores were used for the MPI collectives with 36 MPI processes. The average latency or time to completion was recorded.

[7]Intel MPI Benchmarks - https://software.intel.com/en-us/articles/intel-mpi-benchmarks

2.2.5. Filesystem benchmarking with fs_mark

We used the fs_mark v3.3 benchmark utility[8] for this test to measure the rate of file creation on a given filesystem. This was executed via the Phoronix Test Suite[9] framework and the average result is specified in terms of number of files created per second. We looked at the performance of the BeeGFS parallel filesystem using the three non-OpenStack Haswell nodes each with 3 Micron SSDs. Each SSD is a storage target formatted with the XFS filesystem and a 16GB ext4 partition on one of the SSDs was used as a metadata target. The filesystem client was a spare bare-metal server that executed fs_mark on the filesystem mountpoint. Four benchmark tests were conducted:

- Creating 1000 1MB files using 16 threads with and without the use of sync/fsync.
- Creating 4000 1MB files spread across 30 subdirectories using 16 threads with and without the use of sync/fsync.

For conducting the native BM test, the SSDs were mounted outside the VM and the BeeGFS meta and storage software daemons were running natively whilst the virtual test involved attaching the block devices to the VMs and the BeeGFS daemons running inside the VM. In both cases, raw disk data mode was used i.e. the XFS/ext4 filesystems were readable when mounted outside the VM.

The stripe setting was set to one storage target with a chunksize of 512k bytes. Hence the individual 1M files are assigned to one SSD in a round robin fashion. With three metadata targets, the second test involving 30 subdirectories round-robins each subdirectory to a metadata target.

3. Results

Tab. 3 shows a summary of the benchmarks obtained on OpenStack-provisioned Sandy-Bridge and Haswell hardware and also comparing both native and virtual environments to determine the total virtualisation overheads on both architectures. The filesystem benchmarks are recorded in tab. 4.

Table 3. Summary table of NCI-A*CRC InfiniCloud OpenStack performance benchmarks

Benchmark (units)	SB, native / virtual	HW, native / virtual
iperf (Gbits/s)	43.20 / 39.48	47.08 / 43.18
ib_write_bw (MB/s)	6003.99 / 5901.84	6075.36 / 5963.20
ib_write_lat (μs)	0.94 / 1.43	0.88 / 1.30
Local Linpack (Gflops)	279.15 / 268.41	779.45 / 654.39
MPI Linpack (Gflops)	506.02 / 476.11	1332.41 / 1329.18

3.1. Latency, bandwidth and linpack results on OpenStack InfiniCloud

Fig. 1 illustrates the IB RDMA write latency test and the measured virtualisation overhead for writing a 2-byte chunk of data. The overhead is slightly less on Haswell but both are relatively

[8]The fs_mark benchmark - http://sourceforge.net/projects/fsmark
[9]Phoronix Test Suite - http://www.phoronix-test-suite.com

Table 4. Summary table of parallel filesystem performance on file creation, comparing native and virtual storage backends

FS-Mark Benchmark	Native (files/s)	Virtual (files/s)
1k files	1992	981
1k files, no sync	4335	4168
4k files in 30 subdirs	2039	1172
4k files in 30 subdirs, no sync	3450	3074

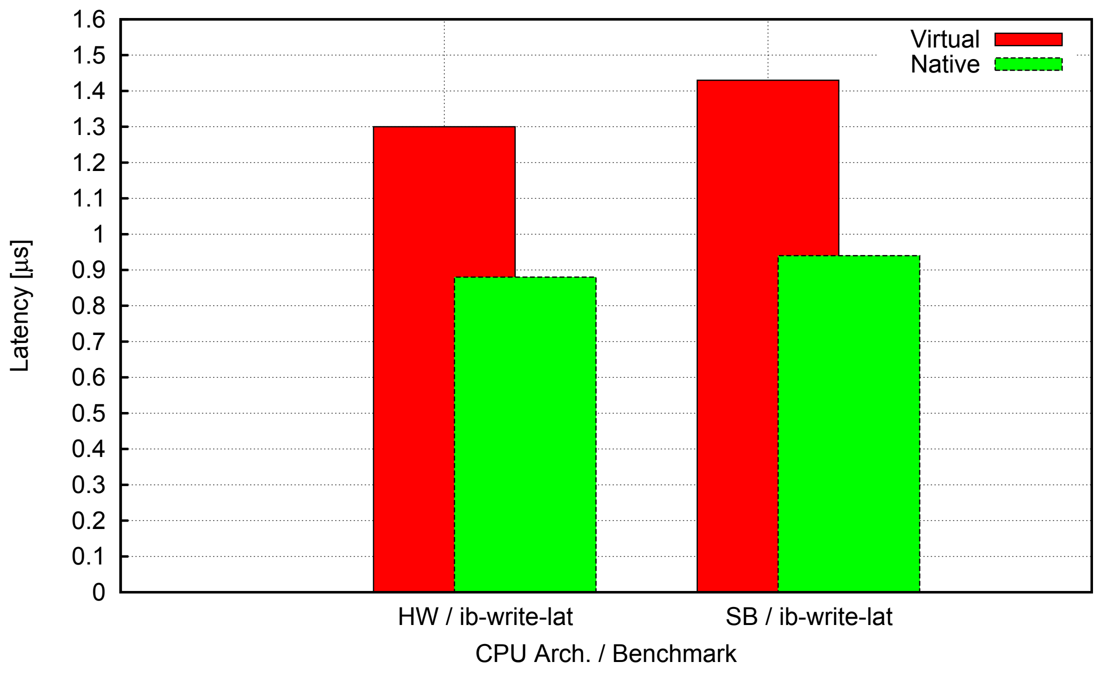


Figure 1. Bargraph showing the IB write latency measurements between native and virtual instances on both Intel architectures for 2-byte messages

significant when comparing native and virtual environments. It has been known from previous works that for small data sizes, the VM latency lags behind native latency [8, 9]. A possible reason is the way small messages are packaged in virtual functions.

When the IB RDMA write bandwidth is tested and shown in fig. 2, we see little overhead as we move to larger message sizes. Haswell is slightly ahead, although the noise encountered whilst executing the benchmark runs means any combination of VM/BM and CPU architecture can win. Since this benchmark is RDMA and should not involve much of the CPU, this shows the performance of the Mellanox interconnect and showing near identical performance between native and virtual instances.

For the TCP iperf test, the overheads for both architectures is greater than that for RDMA due to more involvement of the CPU in processing TCP packets and this illustrates the CPU virtualisation overhead as a result. We see that Haswell pulls ahead due to more processing power over SandyBridge.

For the local Linpack results in fig. 3, the Haswell virtual result shows around 84% performance relative to the native result. We believe that a large fraction of the overhead is due to the CPU virtualisation and this particular benchmark run was taxing the CPU cores. The CPU

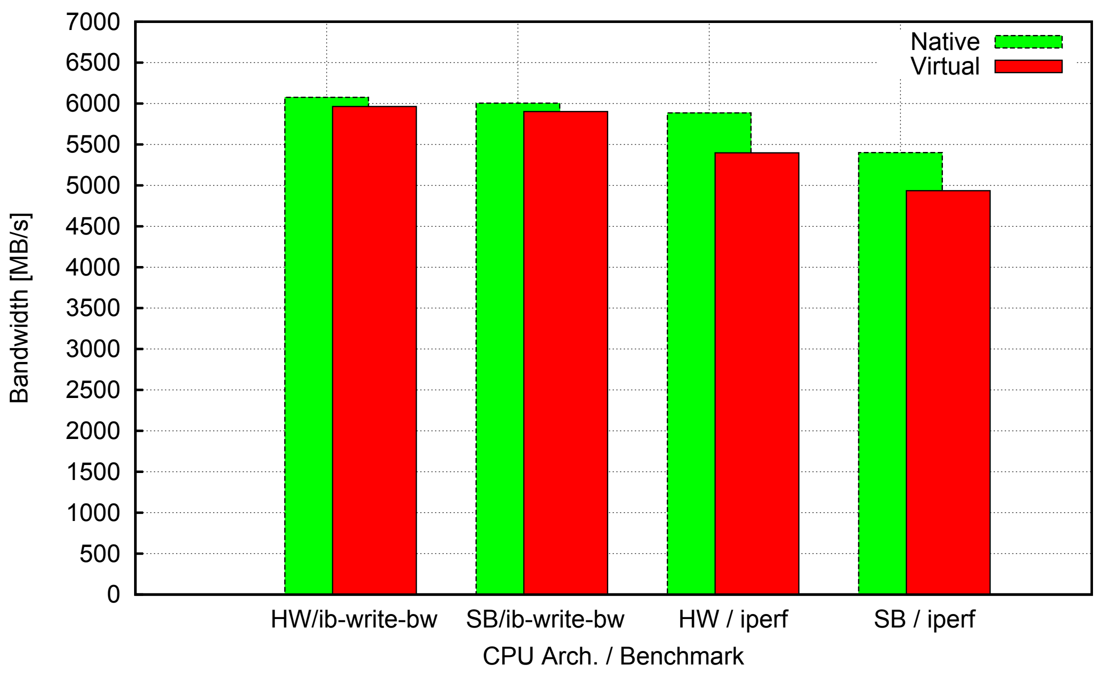


Figure 2. Bargraph showing the IB and TCP bandwidth measurements obtained using the ib_write_bw and iperf micro-benchmark programs respectively on both architectures

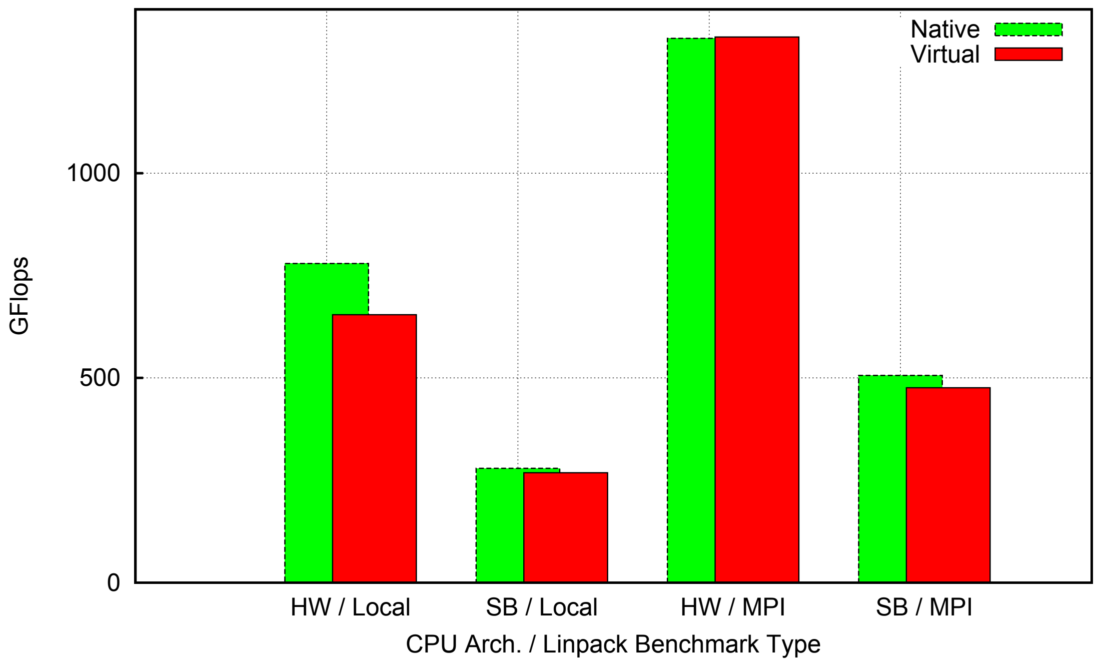


Figure 3. Bargraph showing the performance of both local and MPI Linpack on both CPU architectures

virtualisation overhead was found to be about 10% relative to native performance if no communication is involved [3]. For the other three results, little overhead is shown between virtual and native performance, showing near-identical performance for the communication part and it is likely all three cases were communication-bound. In the example of Haswell with MPI Linpack, more Haswell cores could complete the computational part more quickly hence more time spent communicating. Comparing to the SandyBridge cases, users can expect more than double the performance improvement due to the superior Haswell architecture and more cores available. This should translate into comparable performance for real-world applications on Cloud computing platforms, as long as it is not heavy in collective communications as we illustrate next.

3.2. MPI microbenchmarks

When benchmarking the time taken to send a message back and forth between two processes, there is a difference for small messages until we reach 1M sized messages, shown in fig. 4. This is expected due to the lack of optimisation in the virtual functions packaging small messages, confirmed in previous studies [3], although we do not see any effect of inlining small messages in the native case. But the performance in virtual environments is far better than what is achieved using TCP on InfiniBand through IPoIB mode.

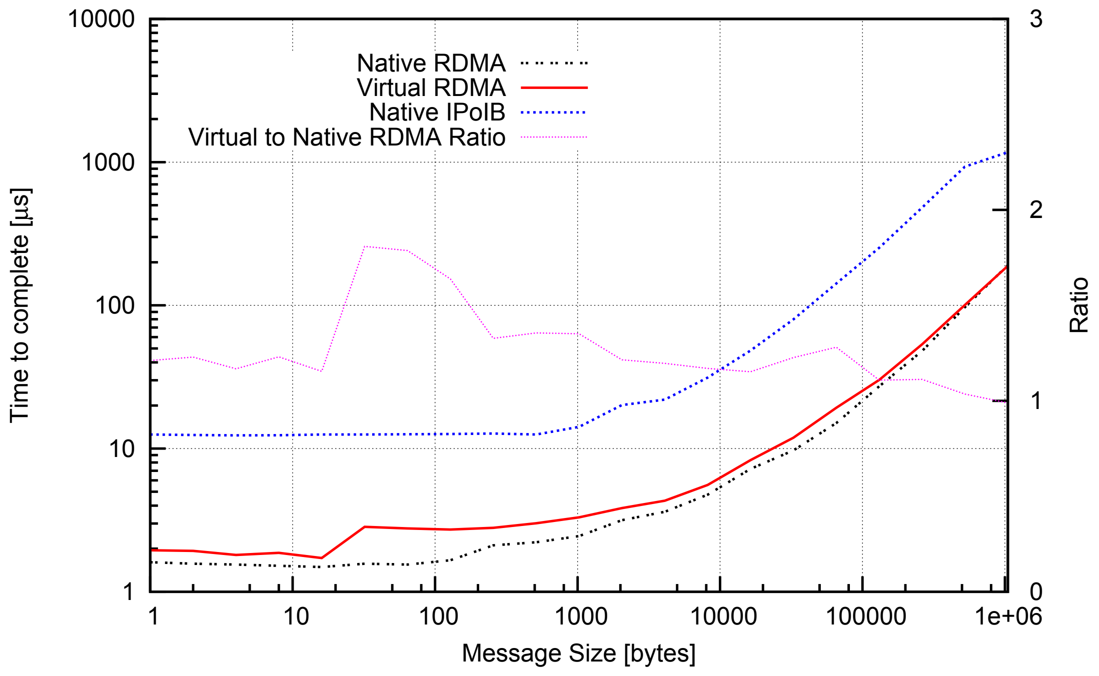


Figure 4. Graph showing a logarithmic plot of the time taken to complete a message pingpong between two MPI processes on distinct nodes against message sizes. The performance ratio between native and virtual RDMA is also shown

When testing MPI collective operations in fig. 5 and 6, we still see inferior performance compared to native mode, even on Haswell hardware. The time for all 36 MPI processes to sync to a barrier is 5.73μs compared to 20.92μs in VMs. For MPI Alltoall in fig. 6, the overheads increase the time by around 2.5 to 3 times, before settling around 1.2 times the native result for larger messages. The sharp jump in the ratio highlights the occurrence of the virtual result jumping to a higher completion time between message sizes 512 and 1024 bytes before the same phenomenon occurring in the native case between 1024 to 2048 bytes. This work confirms similar results from a detailed study on SR-IOV InfiniBand where collective operations are not as optimised on the virtual interfaces [8].

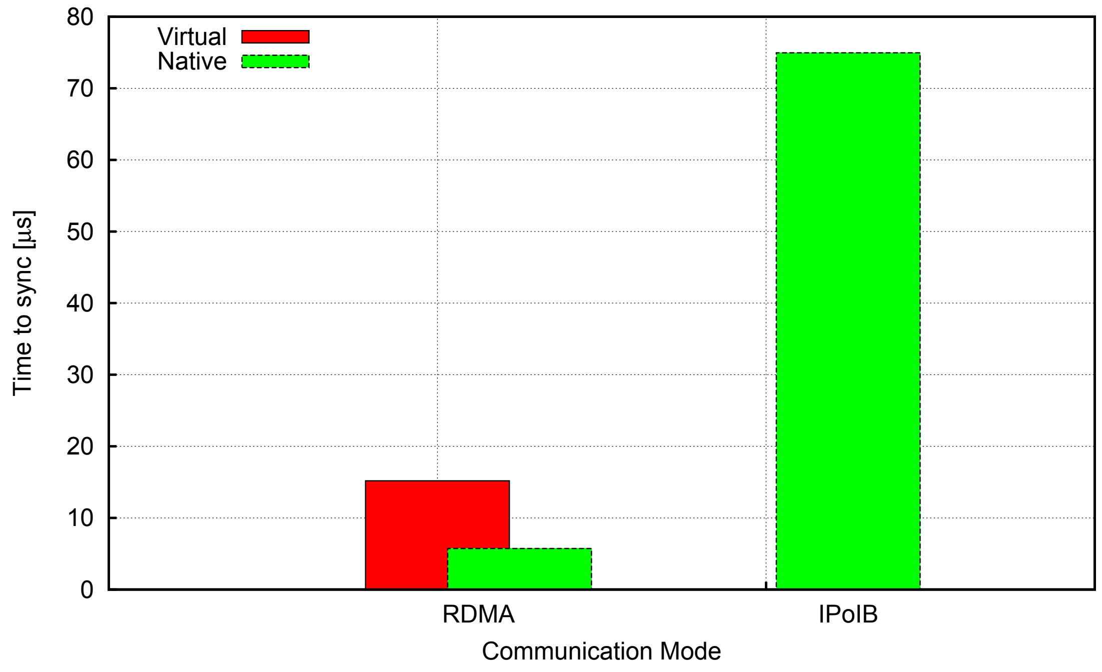


Figure 5. Graph showing the time to synchronise all 36 MPI processes to a collective MPI Barrier

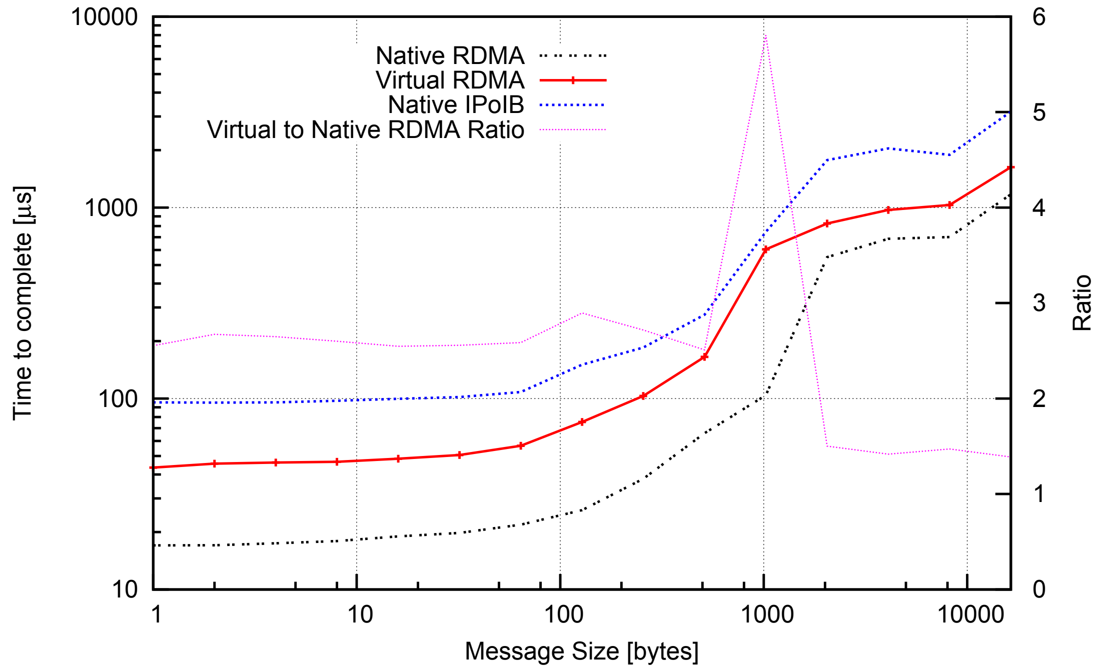


Figure 6. Graph showing a logarithmic plot of time taken to complete an MPI Alltoall collective amongst all 36 MPI processes against message sizes. The performance ratio between native and virtual RDMA is also shown

3.3. Parallel FS performance on native and virtual Haswell servers

Fig. 7 shows the BeeGFS parallel filesystem performance for native and virtual metadata and storage backends. When file syncing is enforced, the difference is about 50% whereas if no syncing is used, the filesystem performance is comparable between native and virtual. We believe that there is further scope for filesystem tuning to improve performance as well as future improvements in the Mellanox virtual functions.

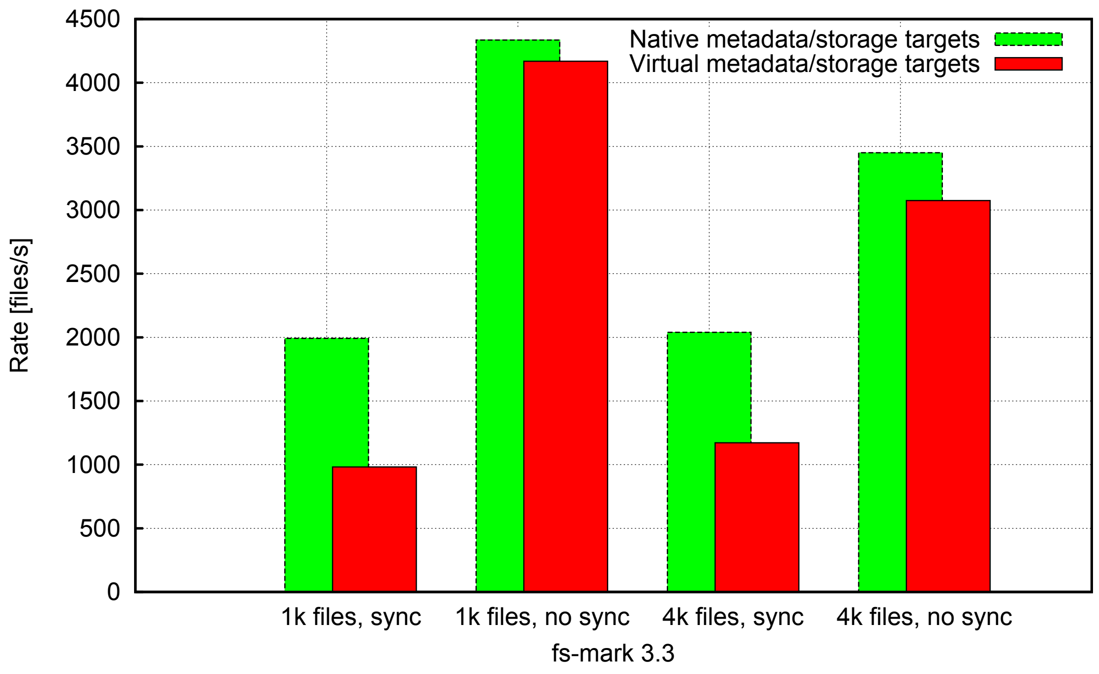


Figure 7. Graph showing the file creation rate of 1MB files, comparing native and virtual storage backends. The parallel filesystem is BeeGFS [5]

Conclusions

We have setup a pair of native and virtual nodes interconnected using SRIOV-enabled FDR InfiniBand and utilising SandyBridge hardware at NCI and Haswell over at A*CRC. These were provisioned using OpenStack with customised patches for InfiniBand and the suite of benchmark tests were conducted to test the network bandwidth, latency and application performance on native and virtual hosts. Later, a three node cluster was utilised to further test the native and virtual performance using MPI microbenchmarks and filesystem benchmarks. In summary, we found that:

- In terms of IB write bandwidth throughput, the difference is negligible for sufficiently large messages on both CPU architectures. For TCP bandwidth, there is an increased CPU virtualisation overheads on both architectures at around 9% with Haswell slightly increasing the throughput due to improved processing power.
- For IB write latency, we see an overhead of around 50 - 60% for 2-byte messages. This confirms previous work that virtual function interfaces are less-optimized for small messages. Haswell does seem to reduce the latency but the effect is minimal.
- For the local and MPI Linpack results, in three cases we see near-native performance. We believe this is due to the particular run not being CPU-bound and that the 168-element block setting ensured sufficiently large messages were exchanged to minimize the virtual overheads. The fourth case may indicate the run was CPU-bound and previous studies confirm a CPU virtualisation overhead of around 10% with no network message exchanges

involved. Clearly, A*CRC's Haswell nodes can offer InfiniCloud end-users a typical speedup of around 2.5 times over NCI's SandyBridge nodes.

- When using MPI collective benchmarks, a significant overhead exists when synchronizing MPI processes and this is still present on Haswell hardware, hence a network hardware limitation and already explored in previous studies.
- When examining parallel filesystem performance using native and virtual backends on Haswell servers and SSD storage, the difference is reduced when not enforcing sync or fsync, otherwise the performance difference is 2 times over for file creation rates. This is a quick look at performance and we believe there is room for filesystem and network parameter tuning.

Despite the overheads are involved, we believe that virtual environments are suitable for typical compute and I/O-intensive workloads whilst providing the benefits of software and resource management that virtualisation can offer. One such example on NCI-A*CRC's InfiniCloud platform is a genetic biological workflow [1] that can immediately take advantage of increased performance now from Haswell servers and from new technology in the future with little or no required adaptation of the software. In the future we would like to see continued effort in overhead reduction, especially for intensive, collective-based communication patterns common in scientific applications using FFTW for example. We believe that technologies such as Docker[10] and Linux Containers are an interesting proposition. Finally, it would be interesting to see how the performance varies on a larger HPC system and not restricted to the small prototype used.

*This work was supported by the A*STAR Computational Resource Centre and National Computational Infrastructure through the use of their high performance computing facilities.*

References

1. Kenneth Ban, Tin Wee Tan, Jakub Chrzeszczyk, Andrew Howard, and Dongyang Li. InfiniCloud: Leveraging Global InfiniCortex Fabric and OpenStack Cloud for Borderless High Performance Computing of Genomic Data and Beyond. Submitted to Supercomputing Frontiers 2015 conference proceedings, Singapore.

2. Jakub Chrzeszczyk, Muhammad Atif, Joseph Antony, Dongyang Li, Matthew Sanderson, and Allan Williams. Perspectives on implementation of a high performance scientific cloud backed by a 56G high speed interconnect. HPC Advisory Council Event, Singapore, `http://www.hpcadvisorycouncil.com/events/2014/singapore-workshop/preso/12_ANU.pdf`, November 2014.

3. Tiago Pais Pitta de Lacerda Ruivo, Gerard Bernabeu Altayo, Gabriele Garzoglio, Steven Timm, Hyun Woo Kim, Seo-Young Noh, and Ioan Raicu. Exploring infiniband hardware virtualization in opennebula towards efficient high-performance computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 943–948. IEEE, 2014. DOI: 10.1109/ccgrid.2014.90.

[10] Docker - https://www.docker.com

4. J Dongarra. Luszczek and A. Petitet (2001):" The LINPACK Benchmark: Past, Present and Future", University of Tennessee. Technical report, mimeo.

5. Jan Heichler. An introduction to BeeGFS. `http://www.beegfs.com/docs/Introduction_to_BeeGFS_by_ThinkParQ.pdf`, November 2014.

6. Marius Hillenbrand, Viktor Mauch, Jan Stoess, Konrad Miller, and Frank Bellosa. Virtual InfiniBand clusters for HPC clouds. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, page 9. ACM, 2012. DOI: 10.1145/2168697.2168706.

7. Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K Panda. A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 125–134. ACM, 2006. DOI: 10.1145/1183401.1183421.

8. Jithin Jose, Mingzhe Li, Xiaoyi Lu, Krishna Chaitanya Kandalla, Mark Daniel Arnold, and Dhabaleswar K Panda. SR-IOV support for virtualization on infiniband clusters: Early experience. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 385–392. IEEE, 2013. DOI: 10.1109/ccgrid.2013.76.

9. Glenn K. Lockwood. High-Performance Virtualization: SR-IOV and InfiniBand. `http://glennklockwood.blogspot.sg/2013/12/high-performance-virtualization-sr-iov_14.html`.

10. Richard Moore, Luca Clementi, Dmitry Mishin, Phil Papadopoulos, Mahidhar Tatineni, and Rick Wagner. Comet: Realizing High-Performance. Virtualized Clusters using SR-IOV Technology. HPC Advisory Council Event, China, `http://www.hpcadvisorycouncil.com/events/2014/china-workshop/preso/3_Moore_Comet.pdf`, November 2014.

11. Tin Wee Tan, Dominic S.H. Chien, Yuefan Deng, Seng Lim, Sing-Wu Liou, Jonathan Low, Marek Michalewicz, Gabriel Noaje, Yves Poppe, and Geok Lian Tan. InfiniCortex: A path to reach Exascale concurrent supercomputing across the globe utilising trans-continental InfiniBand and Galaxy of Supercomputers. Submitted to Supercomputing Frontiers 2015 conference proceedings, Singapore.

Received July 3, 2015.

DOI: 10.14529/jsfi150304

The L-CSC cluster: Optimizing power efficiency to become the greenest supercomputer in the world in the Green500 list of November 2014

David Rohr[1], *Gvozden Nešković*[1], *Volker Lindenstruth*[1,2]

The L-CSC (Lattice Computer for Scientific Computing) is a general purpose compute cluster built with commodity hardware installed at GSI. Its main operational purpose is Lattice QCD (LQCD) calculations for physics simulations. Quantum Chromo Dynamics (QCD) is the physical theory describing the strong force, one of the four known fundamental interactions in the universe. L-CSC leverages a multi-GPU design accommodating the huge demand of LQCD for memory bandwidth. In recent years, heterogeneous clusters with accelerators such as GPUs have become more and more powerful while supercomputers in general have shown enormous increases in power consumption making electricity costs and cooling a significant factor in the total cost of ownership. Using mainly GPUs for processing, L-CSC is very power-efficient, and its architecture was optimized to provide the greatest possible power efficiency. This paper presents the cluster design as well as optimizations to improve the power efficiency. It examines the power measurements performed for the Green500 list of the most power-efficient supercomputers in the world which led to the number 1 position as the greenest supercomputer in November 2014.

Keywords: L-CSC, HPL, Linpack, Green500, GPU, Energy Efficiency, HPC, LQCD.

Introduction

Quantum Chromo Dynamics (QCD) is the physical theory of the strong force, which describes the interaction between quarks and gluons, the fundamental constituents of hadronic matter in the universe. It is a highly nonlinear theory where perturbative methods are only applicable in a small regime. Lattice QCD (LQCD) uses a discretization in a space time grid, and it is the only general *a priory* approach to QCD computations. LQCD requires the inversion of the Dirac operator, which is usually performed by a conjugate gradient algorithm, which involves a sparse matrix-vector-multiplication called $\not{D}$. This $\not{D}$ operator is the computational hotspot of LQCD applications and therefore is responsible for a majority of the runtime of the program. The bottleneck in $\not{D}$ is usually not the compute performance but the memory bandwidth, because sparse matrix-vector-multiplications require many memory loads per compute operation compared to other matrix operations with dense matrices like DGEMM. Hence, for a compute cluster with LQCD as primary focus, a large memory bandwidth is paramount.

Supercomputers are inevitable in today's research. Scientific challenges demand the fastest possible supercomputers, but it is prohibitively expensive to acquire more and more compute power through the use of more and more electricity. In order to use the available resources to the maximum, computers have to become more power-efficient. During the last several years, heterogeneous HPC clusters combining traditional processors with special accelerators such as GPUs or the Xeon Phi have been proven to deliver both superior compute performance and energy efficiency. In an effort to raise awareness for power efficiency, the Green500 list [8] provides a list of supercomputer power efficiencies and presents the "greenest" supercomputers in the world.

[1]Frankfurt Institute for Advanced Studies, Goethe University Frankfurt, Department for High Performance Computing, Ruth-Moufang-Str. 1, 60438 Frankfurt, Germany, rohr@compeng.uni-frankfurt.de
[2]GSI Helmholtz Center for Heavy Ion Research, Planckstraße 1, 64291 Darmstadt, Germany

This paper presents L-CSC (Lattice Computer for Scientific Computing), which is built with commodity hardware and features four high-performance GPUs per compute node. It is organized as follows: Section 1 describes the hardware of the cluster and why it is suited for LQCD. The section outlines the design decisions for good power efficiency. Section 2 illustrates some optimizations we applied to achieve the best efficiency in the Linpack benchmark. Finally, section 3 describes the efforts required to obtain an accurate and reasonable power measurement for the Green500 list and presents the results.

1. The L-CSC cluster

In order to access a broad variety of hardware and reduce acquisition costs, L-CSC is based on off-the-shelf components. Its design follows the LOEWE-CSC and Sanam [7] clusters, which have proven the validity of the commodity hardware approach for GPU accelerated HPC clusters. L-CSC is a general purpose cluster that can run any kind of software, although its main focus is LQCD.

L-CSC continues a trend of increasing performance and memory density of compute nodes as set by its predecessors, LOEWE-CSC and Sanam. Tab. 1 illustrates this trend. The increased memory size enables larger HPC tasks to be executed on a single node and the increased processing power shortens the wall time. Consequently, this reduces the number of nodes and the size of the network in the cluster, which reflects positively on power efficiency and acquisition cost.

Table 1. Comparison of LOEWE-CSC, Sanam and L-CSC nodes (all numbers are aggregate values per compute node)

Component	**LOEWE-CSC**	**Sanam**	**L-CSC**
CPU cores	24	32	40
GPUs	1	4 (2x dual-GPU)	4
System memory	64 GB	128 GB	256 GB
GPU stream processors	1600	7168	11264
GPU memory	1 GB	12 GB	64 GB
GPU peak memory bandwidth	153.6 GB/s	960 GB/s	1280 GB/s
Peak Performance [fp64 GFLOPS]	745.6	3661	10618

The most important criteria for a supercomputer with LQCD-focus are memory bandwidth and memory capacity. Memory bandwidth defines the compute performance and memory capacity defines the maximum lattice size. The performance of $\not{D}$ depends more or less linearly on the memory bandwidth and it is possible to use a large fraction of the theoretically available bandwidth in the application (Bach et al. [2] show more than 100 GFLOPS which translates to about 80% of the peak memory bandwidth with the OpenCL application employed on L-CSC). The demands with respect to memory capacity are a bit more complex. It is mandatory that the lattice fits in GPU memory. If it does fit, no additional memory can be used at all. Hence, memory should not be chosen too large in the first place. For L-QCD calculations, the extent of the time dimension of the lattice is anti-proportional to the temperature. Thermal lattices ($T > 0$) need much less memory than lattices with $T \approx 0$. As a different aspect, the distance of the lattice points can be decreased for better accuracy requiring more memory, but this also slows down the program. Hence, the answer to the question of how much memory is needed depends on the actual problem. A memory of 3 GB is already enough for most thermal lattice sizes ($T > 0$) [7], but has some limitations. By and large, we consider 16 GB of L-CSC's S9150 cards sufficient for almost all lattices.

To make things even more complex, one can distribute the lattice over multiple GPUs or even over different compute nodes. Tests on the Sanam cluster have shown a performance decrease on the order of 20%, when more than one GPU is used. The paradigm for L-CSC is to run most lattices on a single-GPU only, while there is still the possibility of using multiple GPUs for very large ones. Still, multiple GPUs inside a compute node can be fully used in parallel to compute independent lattices. Since LQCD needs a lot of statistic, involving a great deal of lattices, this approach is very efficient.

Overall, the design goal was four GPU boards per node with maximum aggregate GPU memory bandwidth - under the constraint of sufficient memory per GPU. Two GPU types have been chosen: The AMD FirePro S9150 GPU, featuring a capacity of 16 GB and a bandwidth of 320 GB/s. And the AMD FirePro S10000 dual-GPU (i.e. eight GPU chips per node), with a capacity of 2×6 GB (6 GB per GPU chip) and a bandwidth of 2×240 GB/s, thus with a higher aggregate bandwidth than S9150. Besides the higher memory capacity, the S9150 has the additional advantage of being able to reduce the wall time for small jobs compared to the S10000 due to the higher per-GPU-chip bandwidth. This is important for application development and testing, when a quick answer is needed. L-CSC runs all larger lattices on the S9150, and the smaller latices on both S10000 and S9150. Very large lattices can span multiple S9150 cards, having access to 64 GB of GPU memory per node.

L-CSC consists of 160 compute nodes with 48 S10000 GPUs and 592 S9150 GPUs. Each compute node consists of an ASUS ESC4000 G2S/FDR server, two Intel Ivy-Bridge-EP ten-core CPUs, and 256 GB of DDR3-1600 memory. In order to offer more flexibility for general purpose applications on the CPUs in parallel, two CPU models are used: 60 nodes have 3 GHz CPUs for applications with high CPU demands and 90 nodes have 2.2 GHz CPUs. The interconnect is 56 GBit FDR InfiniBand with half bisectional bandwidth and fat-tree topology. Our main OpenCL LQCD application is CL^{2}QCD.[3] It achieves around 135 GFLOPS per S9150 GPU in $\not{D}$, which is the core routing of LQCD, and the aggregate $\not{D}$ performance of the entire cluster is 89.5 TFLOPS [6]. We had optimized it for the Sanam cluster and it performs very well on the new S9150 GPUs of L-CSC without additional modifications. The theoretical peak performance of L-CSC of around 1.7 PFLOPS is in fact much higher than what we achieve in CL^{2}QCD because LQCD is memory bound [6].

2. Optimizing for best power efficiency in Linpack

The Linpack benchmark is the standard benchmark for measuring the performance of supercomputers. The Green500 list presents the most power-efficient supercomputers in the world [8]. Its ranking is determined by the GFLOPS achieved in the Linpack normalized by the average electricity consumption during the Linpack run.

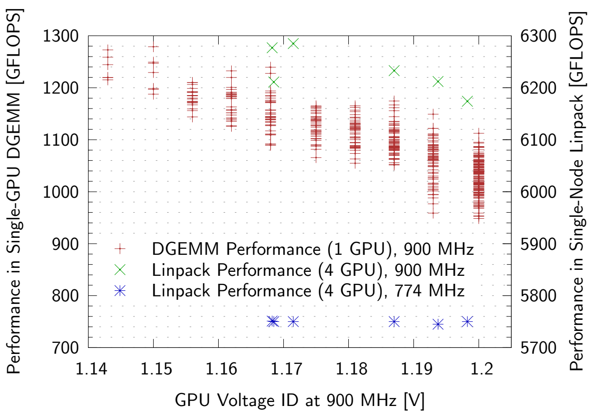
Even though L-CSC consists of commodity hardware, there are no unnecessary components that drain power. The main contributors are the CPUs, GPUs, memory, chipset, network, and remote management. Power consumption of the hard disk with scratch space in each node and of

[3]https://github.com/CL2QCD/cl2qcd

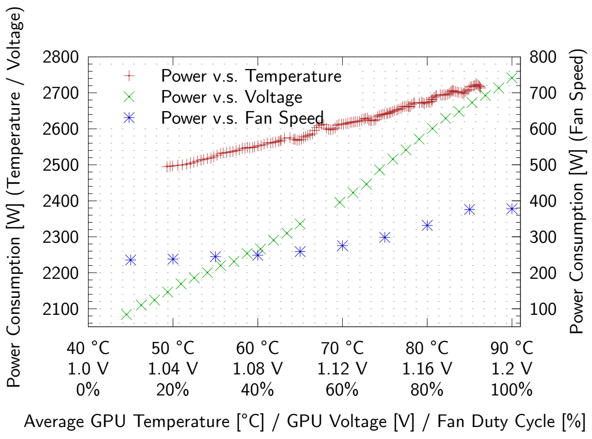
other components are comparatively small, given that each node features four GPUs with 275 W each. Universal Serial Bus (USB) contributes significantly with up to 20 W. L-CSC uses full USB suspend which amounts to the same savings as if USB were switched of completely, so USB does not play a role here.

Some additional optimizations boost L-CSC's power efficiency during the Linpack run for the Green500. An InfiniBand-based network boot allows switching off hard disks, SATA controller, and all Ethernet LAN ports completely. We have investigated the effects of hardware parameters such as fan speed as well as voltage and frequency of GPU and CPU on both power consumption and performance in detail. Fig. 1 shows some of our measurements.

Fig. 1a shows the performance achieved in DGEMM (single-GPU) and HPL (single-node, i. e. quad-GPU) at the stock clocks of 900 MHz and HPL performance at the most efficient clock rate of 774 MHz. The x-axis represents the voltage of the employed GPUs at 900 MHz and it is obvious that the GPUs with higher voltage by trend throttle more and achieve less performance.[3] Fig. 1b shows how the power consumption of the full server varies with GPU temperature, GPU voltage, and FAN Speed.[4]



a) Performance[3]



b) Power Consumption[4]

Figure 1. Performance (a) and power (b) measurements of L-CSC nodes, S9150 GPUs, and system Fans

Due to fluctuations in the manufacturing process every ASIC is a bit different and the vendors account for this by programming individual voltage IDs into their chips. This means that

[3] In the measurements for HPL performance, every measurement point corresponds to one compute node. For each node, we have selected four GPUs of identical voltage ID and plugged these GPUs into the node, such that the GPU voltage on the x-axis defines the voltage of each of the four GPUs in the node. Consider that the x-axis shows the voltage ID of the GPUs at 900 MHz. Running at 774 MHz, the GPUs operate at a lower voltage. For the x-position of the 774 MHz measurements, we still use the voltage ID of the high frequency in order to identify the compute nodes. (774 and 900 MHz measurements of the same compute nodes are shown at the same x-position.)

[4] For these measurements, we always locked all settings to a fixed value (e. g. deactivated power saving features and automatic fan speed adjustments) and used GPU clocks of 774 MHz to avoid throttling. The workload is a continuous DGEMM loop. For the power versus GPU fan speed measurement, we removed all GPUs from the servers to exclude GPU temperature effects and measure only the change in fan power. The power v.s. temperature curve is measured by letting the system heat up over a period of several minutes under load while the measurement is taken. The GPU power consumption measurements for the right plot were performed on an ASUS ESC8000 server, the eight-GPU cousin of the L-CSC servers, to increase the fraction of the GPU power consumption in the total system power consumption.

every individual GPU runs only at the voltage its particular chip needs and especially different GPUs even of the same type operate at different voltages. Today, CPUs and GPUs have a TDP limit and they will throttle their clock frequency under high load if their power consumption would exceed this limit otherwise. Since GPUs operating at different voltages drain different power, the GPUs with lower voltage will hit this limit less frequently and hence operate on average at a higher frequency yielding better performance. Fig. 1a on the left visualizes this aspect in single-GPU DGEMM (matrix-matrix multiplication) and multi-GPU HPL benchmarks. In the DGEMM case at 900 MHz, the GPUs that can operate at the lowest voltage of 1.1425V achieve an average DGEMM performance of 1250 GFLOPS compared to only 950 to 1100 GFLOPS for the slowest GPUs that operate at 1.2V. HPL Performance at 900 MHz varies between 6175 and 6280 GFLOPS. Because multi-node HPL distributes the workload evenly among all compute nodes, the slowest compute nodes dictate the performance. In this case, it is very unfavorable if the compute nodes achieve different performances due to different GPU voltages. Using an exhaustive search in the parameter space of GPU voltage, GPU and CPU frequencies, fan speed settings, and settings for the HPL-GPU benchmark, we have identified the parameter set that we believe delivers the best power efficiency. The optimal GPU frequency is 774 MHz. Fig. 1a (left) shows a completely flat performance profile for this energy-efficient configuration with 774 MHz, i. e. no GPUs throttle and all nodes achieve the same performance.

An interesting observation in this context is that the highest clock rate of 900 MHz does not deliver the highest performance. Due to the throttling, the GPU oscillates between the 900 MHz frequency and lower frequencies. This is less efficient than constant operation at the highest possible frequency that does not throttle. For instance, running with default GPU power management settings on L-CSC, we see higher constant DGEMM performance at a GPU frequency of 820 MHz than with 900 MHz.

The fig. 1b (right) shows the dependency of the power consumption on fan speed, GPU voltage, and GPU temperature. Obviously, the largest contribution by far comes from the GPU voltage. For the final Linpack run, we have used the minimum voltage required for stable operation of all GPUs at the target frequency of 774 MHz. Now, it is clear that one cannot operate at the lowest possible temperature and on low fan speeds at the same time because low fan speeds cause higher temperatures. The power curve for different fan speeds shows a stronger slope for fan speeds above 40% and we have found 40% to be the optimum during the high-load phase of the Linpack benchmark. Toward the end of a Linpack run, the load reduces significantly. We account for this by employing a curve that defines different FAN duty cycles for different load levels / temperatures, which ensures that the FANs always run only at the minimum speed required. This reduces further power consumption.

For running the Linpack benchmark we employed our HPL-GPU[6] [1] implementation of the benchmark, which we have developed and used for the LOEWE-CSC and Sanam clusters before. It provides two operating modes: One optimized for maximum performance, and an alternative mode that sacrifices a small fraction of the performance to reduce the power consumption resulting in better net power efficiency. This alternative efficiency-optimized mode was developed further for L-CSC and has been used for the Green500 result [6].

[6]https://github.com/davidrohr/hpl-gpu/wiki

3. Measuring the power consumption for the Green500 list

The Green500 ranking is determined by the quotient of the achieved performance in the Linpack benchmark divided by the average power consumption during Linpack execution. Due to late installation of the system, only 56 nodes with S9150 GPUs were available for the Linpack benchmark in November 2014, which were connected by three InfiniBand switches in a ring-configuration. We did not repeat the Linpack measurements on the full system which has gone in production operation meanwhile. From scalability tests of HPL-GPU on the Sanam cluster and on subsets of L-CSC [6], we assume the full system would achieve an almost identical power efficiency. The current Green500 measurement methodology revision 1.2 is defined by [3]. Tab. 2 lists three measurement levels defined in this methodology document yielding different accuracies.

Table 2. Measurement levels for Green500 with different accuracy

Level	Components	Measured fraction of system	Duration
1	Only compute nodes	At least $\frac{1}{64}$ of the system	At least 20% of the middle 80% of the run
2	Full cluster with network (network estimated)	At least $\frac{1}{8}$	Full runtime
3	Full cluster with network (network measured)	Full system	Full runtime

Level 1 is provided for facilities without sufficient equipment for higher level measurements. Unfortunately, the level 1 specifications are exploitable such that one can create measurements which show a higher power efficiency than actually achieved [6]. Thus, higher levels are preferred. The L-CSC installation had only one revenue grade power meter available (see [3] for power meter requirements), and it was thus impossible to measure a larger fraction of the system at the accuracy required for level 2 or level 3. Thus, only a level 1 measurement was feasible. All measures were taken to make the result as accurate as possible. Our measurement for L-CSC includes the entire Linpack run and we measured the entire cluster with the network. Due to the lack of more revenue grade power measurement equipment, only two compute nodes could be measured. However, in order to obtain the most accurate result, we have taken additional measures to mitigate the effect of measuring not the full system. Power consumption variability of nodes can be estimated by measuring the efficiency of several individual nodes during single-node Linpack runs, which yielded the following values on seven randomly chosen nodes:

5154.1, 5260.1, 5248.4, 5245.5, 5125.1, 5301.2, 5169.3 [MFLOPS/W].

The results show a relatively small variation of only ±1.2%. In order to deliver the most accurate result, we used nodes with middle power consumption among the nodes we had measured individually before. Hence, the difference to the full level three measurement is small. The only aspect not fulfilling level 3 is the number of measured nodes. With a deviation of less than 1.2% between the nodes, and due to the fact that we have chosen average nodes for the final measument, we assume that our efficiency result is off by less than 1% from a full level three measurement. (Surprisingly for us, the three switches only contribute with 257 W to the power consumption.) In contrast, many other top ranked Green500 systems (e. g. the ExaScaler systems and TSUBAME-KFC currently ranked on places 1, 2, 3, and 5 of the June 2015 list) do not take measures to ensure exact measurements but instead only measure a period with low power consumption according to level 1 [5]. We have shown that in case of L-CSC such a measurement overestimates the real efficiency by up to 30% [6], currently corresponding to the margin of the first over the fourth rank in the Green500 list. This greatly deteriorates the comparability. Accordingly, the newly released power measurement specification 2.0 RC 1 [4] for the Green500 list prohibit such course of action and will lead to better comparability of new measurements.

4. Results and Conclusion

The 56 nodes used for the measurement achieved a Linpack performance of 301.5 TFLOPS expending on average 57.2 kW and yielding an average efficiency of 5271.8 MFLOPS/W with a measurement error of less than 1.2%. With this result, L-CSC was awarded 1st place in the Green500 list of November 2014 as the most power-efficient supercomputer in the world. We have selected lower clocks and voltages to achieve optimal performance. The performance decrease is not very large in applications like Linpack that reach close to the peak performance because under such high load the GPUs cannot maintain the maximum clocks over a long time. Essentially, when the GPU operates at its power limit, the achieved performance depends linearly on the power efficiency and we have seen that a slight decrease in clock speed can even lead to a better performance. The energy efficiency improvements we observe are also applicable to our application. Our LQCD $\not{D}$ kernel in particular is memory bound and little sensitive to frequency. It suffers less than 1.5% performance decrease with the efficiency-optimized settings.

We would like to thank Advanced Micro Devices, Inc. (AMD) and ASUSTeK Computer Inc. (Asus) for their support. A part of this work was funded by HIC for FAIR.

References

1. Matthias Bach, Matthias Kretz, Volker Lindenstruth, and David Rohr. Optimized HPL for AMD GPU and Multi-Core CPU Usage. *Computer Science - Research and Development*, 26(3-4):153–164, 2011. DOI: 10.1007/s00450-011-0161-5.

2. Matthias Bach, Volker Lindenstruth, Christopher Pinke, and Owe Philipsen. Twisted-Mass Lattice QCD using OpenCL. In *Proceedings of Science - 31st International Symposium on Lattice Field Theory - LATTICE 2013.*

3. EEHPC Working Group. Energy Efficient High Performance Computing Power Measurement Methodology v1.2 RC2.

4. EEHPC Working Group. Energy Efficient High Performance Computing Power Measurement Methodology v2.0 RC1, 2015.

5. Toshio Endo, Akira Nukada, and Satoshi Matsuoka. TSUBAME-KFC: Ultra green supercomputing testbed. Presented at International Conference for High Performance Computing, Networking, Storage and Analysis (SuperComputing, SC13), 2013.

6. David Rohr, Matthias Bach, Gvozden Nešković, Volker Lindenstruth, Christopher Pinke, and Owe Philipsen. Lattice-CSC: Optimizing and building an efficient supercomputer for

Lattice-QCD and to achieve first place in Green500. In *Proceedings of the International Supercomputing Conference*, 2015. DOI: 10.1007/978-3-319-20119-1_14.

7. David Rohr, Sebastian Kalcher, Matthias Bach, A. Alaqeeli, H. Alzaid, Dominic Eschweiler, Volker Lindenstruth, A. Sakhar, A. Alharthi, A. Almubarak, I. Alqwaiz, and R. Bin Suliman. An Energy-Efficient Multi-GPU Supercomputer. In *Proceedings of the 16th IEEE International Conference on High Performance Computing and Communications, HPCC 2014, Paris, France. IEEE*, 2014. DOI: 10.1109/hpcc.2014.14.
8. Sushant Sharma, Chung-Hsing Hsu, and Wu-Chun Feng. Making a case for a Green500 list. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, page 343. IEEE, 2006. DOI: 10.1109/ipdps.2006.1639600.

Received June 7, 2015.

DOI: 10.14529/jsfi150305

An Autonomic Performance Environment for Exascale

Kevin A. Huck[1], *Allan Porterfield*[2], *Nick Chaimov*[1], *Hartmut Kaiser*[3], *Allen D. Malony*[1], *Thomas Sterling*[4], *Rob Fowler*[2]

Exascale systems will require new approaches to performance observation, analysis, and runtime decision-making to optimize for performance and efficiency. The standard "first-person" model, in which multiple operating system processes and threads observe themselves and record first-person performance profiles or traces for offline analysis, is not adequate to observe and capture interactions at shared resources in highly concurrent, dynamic systems. Further, it does not support mechanisms for runtime adaptation. Our approach, called APEX (Autonomic Performance Environment for eXascale), provides mechanisms for sharing information among the layers of the software stack, including hardware, operating and runtime systems, and application code, both new and legacy. The performance measurement components share information across layers, merging first-person data sets with information collected by third-person tools observing shared hardware and software states at node- and global-levels. Critically, APEX provides a policy engine designed to guide runtime adaptation mechanisms to make algorithmic changes, re-allocate resources, or change scheduling rules when appropriate conditions occur.

Keywords: ParalleX, HPX, exascale, performance measurement, adaptive runtimes.

Introduction

The transition to extreme-scale computing poses new challenges in performance analysis and optimization because of the anticipated high concurrency and dynamic operation that will be required to make extreme-scale systems operate efficiently. Increasingly heterogeneous hardware, deeper memory hierarchies, reliability concerns, and constraints posed by power limits will contribute to a dynamic environment in which hardware and software performances may vary considerably during the application's execution. Furthermore, emerging exascale programming models will emphasize message-driven computation and finer-grained parallelism, resulting in more asynchronous computation. It is no longer reasonable to expect that a *post-mortem* performance measurement and analysis methodology will suffice to optimize applications in such an environment.

Rather, there is a strong need for runtime performance observation that merges in real time *first-person* (application perspective) with *third-person* (resource perspective) introspection, and for *in situ* performance analytics to identify bottlenecks and their impact on specific sections of code. This information can drive online dynamic feedback and adaptation techniques that can be integrated with an exascale software stack. The goal is to create an autonomic capability in the exascale system that can direct the application performance to more productive execution outcomes. In this paper, we describe our prototype implementation of an *Autonomic Performance Environment for eXascale (APEX)* that is the part of the *OpenX* integrated software stack being developed in the DOE XPRESS project [9] (see Section 1). The APEX prototype supports both introspection and policy-driven adaptation for performance and power optimization objectives. We describe the APEX design and development in Section 2. Section 3 shows several examples that demonstrate the effects of APEX-enabled execution. This focuses on mak-

[1]Performance Research Laboratory, University of Oregon, Eugene, OR 97405, USA
[2]Renaissance Computing Institute, University of North Carolina, Chapel Hill, NC 27517, USA
[3]The STE||AR Group, Louisiana State University, Baton Rouge, LA 70803, USA
[4]Center for Research in Extreme Scale Technologies, Indiana University, Bloomington, IN 47408, USA

ing guided adjustments to thread-scheduling controls for different policy objectives. Section 5 discusses the next steps in our research work.

1. XPRESS Project

The XPRESS project is organized into four major elements: system software, programming models and languages, applications and cross-cutting issues. The HPX-3 runtime system [3, 18, 35] serves as a starting point as programming tools and runtime system target at the beginning of the XPRESS project. This has been complemented by the development of HPX-5, which is being developed to add functionality for fault tolerance and power management, and to provide a robust open-source runtime system. The LXK lightweight kernel operating system based on the advanced Kitten operating system [7, 31] is being developed in response to the new requirements for billion-way concurrency, introspective management of faults and power, and management of a protected and dynamic global virtual name space. It targets projected future directions of system architectures while running efficiently on near term systems. LXK is co-designed with HPX around the centrepiece of the RIOS interface between the runtime and operating system software. This interface will share information in both directions between the two major software layers for performance, reliability and control of power consumption. The OpenX software stack is shown in fig. 1.

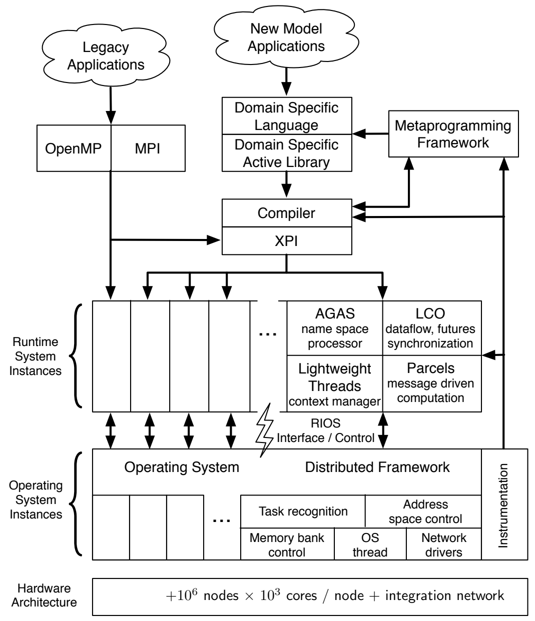


Figure 1. Major components of the OpenX architecture stack. APEX is the cross-cutting *instrumentation* component

Two programming methods are employed to provide early means of conducting application and kernel-driven experiments, as well as to the facilitate ease of programming and portability. In addition to the native programming API provided by HPX-3 and HPX-5 and potentially wrapped by Domain Specific Languages (DSLs), a low-level imperative programming interface, XPI is being developed to expose the semantic constructs comprising the ParalleX execution model [18] embodied in the experimental HPX runtime systems. The project is exploring legacy mitigation to ensure the seamless transition to the OpenX software stack of codes written with the help of MPI or OpenMP. The approach is to develop XPI interfaces for these programming models, thus provide interoperability between software modules in both forms, and provide a path for incrementally extending parallelism within the MPI and OpenMP frameworks. APEX provides performance instrumentation interfaces compatible with XPI, DSL, and legacy codes.

Essential cross-cutting functions include automatic control and introspection, resilience, power management and heterogeneity. Power-management software in combination with anticipated energy-efficient hardware will achieve much greater resource utilization per joule while reducing data movement dramatically, a major source of power consumption through active locality management. APEX represents the initial research prototype for introspection and dynamic control required for the XPRESS project.

2. APEX Design

2.1. Overview

APEX aims to enable autonomic behavior in software by providing the means for applications, runtimes, and operating systems to observe and control their performance. Autonomic behavior requires performance awareness (introspection) and performance control/adaptation. APEX is designed around these two main components. APEX provides introspection from both top-down and bottom-up perspectives, including node-wide resource utilization data, energy consumption, and health information, all accessed in real-time. The introspection results are combined and associated with policy rules in order to provide the feedback control mechanism.

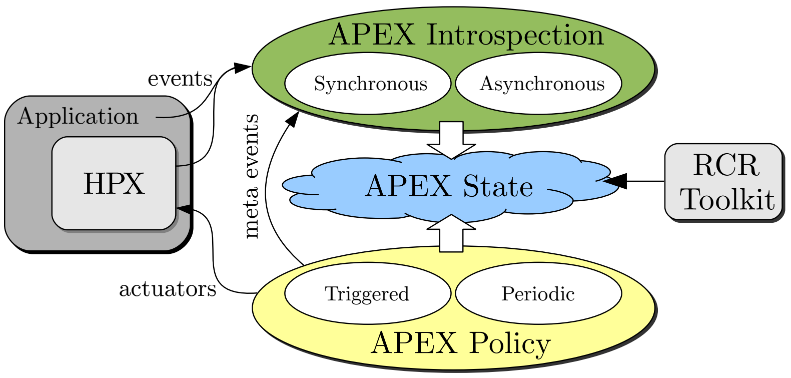


Figure 2. APEX design

2.2. Introspection

APEX collects *top-down* introspection data from a runtime system, library, or high-level application through an event-based *inspector* API. The software to be controlled is instrumented with this event API. APEX recognizes several types of logistic events such as initialization, termination, setting a process rank (*e.g.,* an MPI rank, or HPX locality ID), and creating a new thread. For measurement, APEX has instrumented timer-start and timer-stop calls, as well as sampled counter values (*e.g.,* bytes transferred, queue length, idle rate). These API calls enter APEX as events. Internally, APEX has several event *listeners* that perform actions based on the types of events that are passed in to APEX. Events are either handled by listeners immediately using synchronous code execution or are handled using asynchronous method invocation. For the asynchronous processing, the event is stored internally on a queue for background processing, and the execution control is quickly returned to the code that called the APEX API. Custom events are also available to trigger specific policy engine rules. Further explanation of this behavior is presented in Section 2.4.

Bottom-up introspection data is collected from the operating system and hardware using periodic sampling. These measurements do not use events, but some additional OS threads are spawned to periodically read values directly from available sources. On Unix-like systems, the `/proc` virtual filesystem files provide access to CPU, memory, network, disk, process, and operating system statistics. Resource Centric Reflection (RCR) [21, 22] provides a user-level API to access any counter available through PAPI, PERF_EVENTS, or a hardware instruction. RCRdaemon runs on protection ring 0 and supplies information about hardware resources shared by more than one core (*e.g.,* energy consumption, Last Level Cache events, or memory-controller usage) in a data structure that can be read at user-level. RCRdaemon uses a self-describing hierarchical data structure in a shared memory region to transmit protected counter values in an application-agnostic manner. The power interface reads these values and can be used by any application to acquire power/energy information. RCR calipers can be placed around any code region (up to the entire application) to measure energy used by that region. On Cray systems the access to the protection level 0 is denied, but the Cray PM Counters [23] facility is available. RCRdaemon was therefore modified to get its data from this source. The values were then placed into the same data structure previously used. The user API was unchanged. Updates occur at the same rate as Cray updates `/proc`.

2.3. Event Listeners

As mentioned in Section 2.2, APEX events are processed by event listeners. Each listener is implemented as a C++ class, and as events pass through APEX, each instantiated listener is given access to the event object. The listeners implement handler methods for each event type available in the system. Notable event listeners in APEX include the *Profiling Listener*, the *Concurrency Listener*, the *Policy Engine Listener*, and the *TAU Listener*.

The profiling listener implements timer and counter measurement back-end processing in APEX. The salient events processed by the profiling listener include the `timer_start`, `timer_stop`, and `sample_value` events. When the profiling listener gets a `timer_start` event, it creates a profiler object, generates a timestamp, and returns a handle to the profiler object. When the profiling listener gets a `timer_stop` event, it takes a second timestamp, puts the profiler object in a single-producer-single-consumer (spsc) queue for back-end processing, and returns. Each OS thread in the process has its own spsc queue to avoid contention. Similarly, when the profiling listener gets a `sample_value` event, it creates a profiler object, puts it in the spsc queue for back-end processing, and returns. The profiling listener has a background *consumer* thread that waits for a signal that indicates that data has been pushed onto one of the queues. When the consumer thread has been signalled, it clears all of the spsc queues of pending work by removing a profiler object from the queue and updates the per-thread and per-process statistical profile for the running application. The current executing profile can be queried subsequently at runtime through an introspection API. The optional TAU listener is similar to the profiling listener with the exception that all processing is done synchronously through the TAU measurement library in order to generate a detailed profile or trace for offline, post-mortem performance analysis.

The concurrency listener works as follows. The salient events processed by the profiling listener are the `timer_start` and `timer_stop` events. When the concurrency listener gets a `timer_start` event, it pushes the timer ID onto a thread-specific stack, and returns. When the profiling listener gets a `timer_stop` event, it pops a timer ID off of the thread-specific stack. The concurrency listener also has a background consumer thread that periodically examines the top of each thread's timer stack and builds a histogram reporting the task currently being executed by each thread during that time quantum. At the end of execution, the histograms are written to files on disk and `gnuplot` [37] is used to visualize a concurrency graph of the application. Fig. 3–7 are examples of concurrency graphs. The concurrency listener does not have a role in runtime adaptation and is instantiated only when concurrency graphs are desired.

2.4. The Policy Engine

The most important listener component in APEX is the Policy Engine. The policy engine provides autonomic controls to an application, library, runtime, or operating system using the introspection measurements described in Section 2.2. Policies are rules that decide on outcomes based on the observed state captured by APEX. The rules are encoded as callback functions that are registered with APEX and are either *triggered* or *periodic*. Triggered policies are invoked by an APEX event, whereas periodic policies, by definition, are executed at set intervals. The policy rule functions have access to the APEX API in order to request profile values from any measurement collected by APEX. Using these values to make logical decisions, the functions can change the behavior of the application by whatever means available, such as throttling threads, changing task granularity, or triggering data movement such as mesh refinement or repartitioning. In this way, the policy engine enables runtime adaptation using introspection data, engages actuators across stack layers, and can be used to invoke online auto-tuning support.

2.5. Global Performance Views

Thus far in the discussion performance introspection has been limited to local node observations. No performance information from remote nodes or processes is available implicitly to the local policy functions. However, there are situations in which global performance information is necessary to make runtime adaptation decisions for problems such as load balancing. In those cases, APEX provides a skeleton interface for exchanging local information in a distributed application scenario. The global exchange of local performance data in APEX is similar to that provided by TAUg [16], in which TAU performance data collected by an MPI application was exchanged using MPI functions. Rather than be tied directly to a specific communication infrastructure, APEX provides a skeleton interface to be populated using the distributed communication library used in the application to be controlled. Examples implemented so far include HPX-3, HPX-5 and MPI. The interface that the runtime has to implement includes two functions: `action_apex_get_value()` – each node gets local data to be reduced and performs an optional *put* (if implementing a push model) and `action_apex_reduce()` – each node performs an optional *get* (if implementing a pull model), all remote node data is aggregated at root node, and an optional push broadcasts the aggregated result back out to the non-root nodes. Ideally, puts and gets are performed using one-sided communication such as remote distributed memory accesses (RDMA) or by using a Global Address Space (PGAS or AGAS).

2.6. HPX Integration

APEX is integrated with operating systems, runtime systems, libraries, and applications by instrumenting the code with calls to the APEX introspection API, as well as by registering desired policy functions and global communication. Because both HPX-3 and HPX-5 are task-based runtime systems, we added the instrumentation in the respective task schedulers, placing timer start/stop calls just before and after task functions are executed, taking special care to avoid measuring internal lightweight tasks such as "no-op". `Sample_value()` calls were added to capture internal runtime statistics (*i.e.,* number of yields, steals, spins, *etc.*) and we added other instrumentation for initialization, thread creation and termination. Where applicable, we wrote policy functions and added the code to register the policy functions to perform adaptation of the runtime system. All the examples described in Section 3 modify runtime behavior in the same way, by setting a cap on the maximum number of active worker threads, so we also modified the HPX thread scheduler loop for worker threads to check the cap value and de-activate the worker thread if the number of active threads is greater than the thread cap. Even though we are measuring nearly every task executed by the runtime, our measurements show that the overhead introduced by APEX does not exceed 2% and is usually less than 1%, depending on the granularity of the executed tasks. We believe that this is due to our asynchronous profile-processing combined with the small but sufficient amount of available processing capacity headroom when executing on many-core nodes. Global performance data is exchanged in HPX using the Active Global Address Space (AGAS).

3. Experimental Results

In order to demonstrate the features and capabilities of APEX, we integrated it with two distinct but related runtimes, HPX-3 and HPX-5. We implemented a variety of policy rules, and we present a selection of them here, along with the applications that best demonstrate them. In this section we present the following examples:

- HPX-3 1-D stencil code, runtime optimized for best performance
- HPX-5 Single-source, shortest-path benchmark, runtime optimized for highest throughput
- HPX-5 LULESH kernel, runtime modified to stay under a user-specified power cap
- HPX-3 miniGhost kernel, runtime modified to stay under a user-specified power cap

All of the experiments described below were conducted on Edison, a Cray XC30 system deployed at NERSC [36]. Edison has 5576 nodes with two 12-core Intel "Ivy Bridge" processors operating at 2.4 GHz, with a total of 48 threads per node (24 physical cores w/hyperthreading). The network on Edison is a Cray Aries interconnect with Dragonfly topology, with 23.7 TB/s global bandwidth. As LXK hadn't been integrated with HPX yet, the applications were executed on the Compute Node Linux (CNL) operating system.

3.1. HPX-3 1-D Stencil Code

The 1D stencil code is a simple, iterative heat-diffusion solver using a 3-point stencil, used as an example code for HPX-3, and for which multiple versions are available with different optimizations applied. The simplest version represents the computation for each data point as an individual future, but the performance of this version is extremely poor as the task granularity is far too small. The version with good performance partitions the data into a user-configurable number of equally-sized chunks, with the computation on each chunk being represented as a future. Within a node, performance initially increases with an increasing number of worker threads, but then decreases.

Fig. 3a shows the runtime (blue line) of the 1D stencil code as function of number of worker threads from 1 to 24, which is the number of physical cores available on Edison nodes. It also shows that runtime is highly correlated with the average thread queue length (red line), which is a counter exposed by the HPX-3 runtime representing the number of tasks waiting to execute on worker threads. APEX can query the thread queue length while the program is executing and adjust dynamically the number of worker threads allocated to minimize runtime.

Fig. 3b shows the concurrency graph for the execution of the 1D stencil code run on 100,000,000 elements partitioned into 1000 chunks with 48 worker threads, which is the number of logical cores available on the Edison node with hyperthreading enabled. Actual concurrency is substantially lower, as many tasks wait on dependencies to complete before become eligible to run, and there is a substantial variability in actual concurrency over time. This execution takes 138 seconds to run. Fig. 3c shows the concurrency graph for an execution of the same problem size but with 12 worker threads, which produces the shortest runtime of any number of worker threads. That execution takes 61 seconds to run.

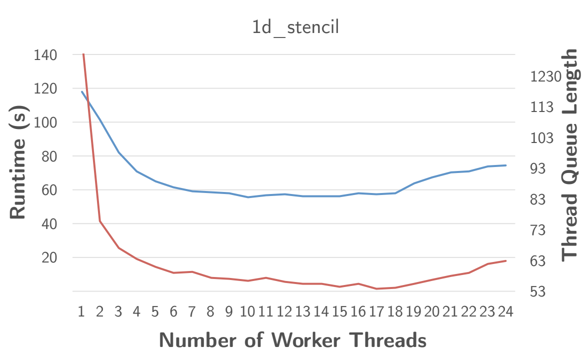
Fig. 3d shows the concurrency graph for the same problem size and an initial number of worker threads of 48, but using discrete hill-climbing search to minimize the average thread queue length. It converges on 13 worker threads (*vs.* the optimal value of 12) and acts in an enough quick way so that the overall runtime is nearly as fast (64 seconds) as starting with the optimal number.

3.2. HPX-5 SSSP benchmark

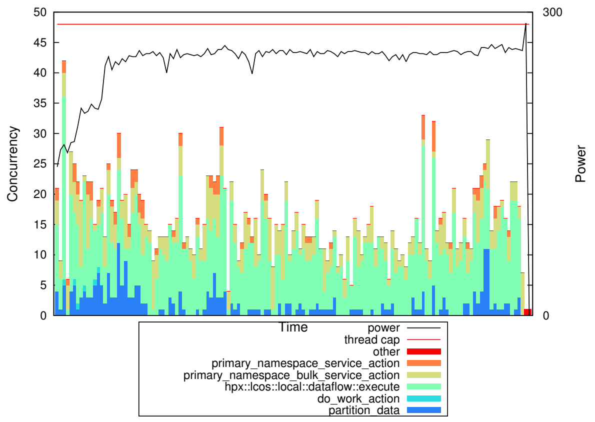
The Single Source, Shortest Path graph search benchmark (SSSP) [5] is a candidate for inclusion in the Graph500 [6] benchmark kernels. Given an initial graph, the SSSP benchmark computation finds the shortest distance from a given starting vertex to every other vertex in the graph. In the HPX-5 implementation, a large graph is loaded and distributed across localities, a point is selected at random, and the shortest path between it and all other points is found. The search runs for a fixed length of time and terminates when the accumulated time performing searches exceeds the specified length of time. Key constraints of the benchmark are that only one initial vertex search is performed at a time, and no memoization between searches is allowed. The dataset used in this example is the Random4-n.10 dataset, executed for 60 seconds worth of timed searches. For this benchmark, the metric of interest is total throughput, not time to completion. The code was run on 10 nodes, using 24 threads per node (no hyperthreading).

The APEX Policy rule used for optimization of SSSP was the maximization of the number of calls to `_handle_queue_action()`, used as proxy for the “throughput” metric. The primary metric for this benchmark is Traversed Edges Per Second (TEPS), and the queue contains vertices to be explored. The policy function adjusts the thread concurrency to maximize throughput, using the *Parallel Rank Order* search strategy provided by the auto-tuning and optimization search framework *Active Harmony* [10]. The initial value for the thread cap was set at 24, with a minimum value of 6. The policy function was registered to execute on a periodic basis (1Hz), adjusting the thread cap to a new value as specified by the optimization search.

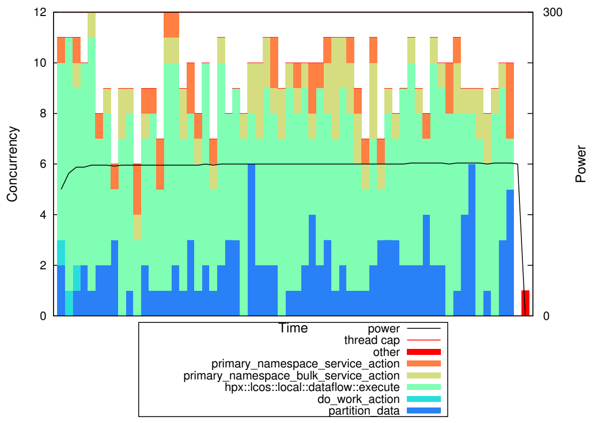
[5] `http://hpx.crest.iu.edu/applications`
[6] `http://www.graph500.org`



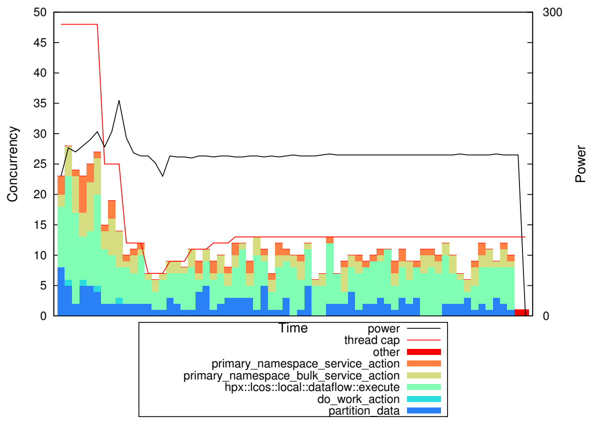
a) 1D stencil strong scaling. This chart shows the correlation between the execution time (blue line) and the queue lengths (red line) when running with different numbers of threads on Edison



b) 1D Stencil unthrottled. This concurrency chart shows a stacked bar chart with the periodic (1 Hz) status of each OS thread. The max number of threads is 48, and the instantaneous power for each sample is the black line



c) 1D Stencil with ideal number of threads. This concurrency chart shows the periodic (1 Hz) status of each OS thread. The number of threads is fixed at 12, and the instantaneous power for each sample is the black line



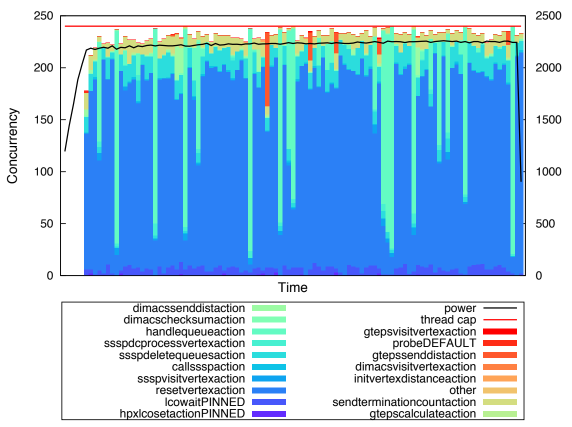
d) 1D Stencil throttled by APEX. This concurrency chart shows the periodic (1 Hz) status of each OS thread. The number of active threads starts at 48, but is throttled while APEX searches for an optimal number of active threads to minimize execution time. The evolving thread cap is the red line, and the instantaneous power draw is the black line

Figure 3. 1D Stencil

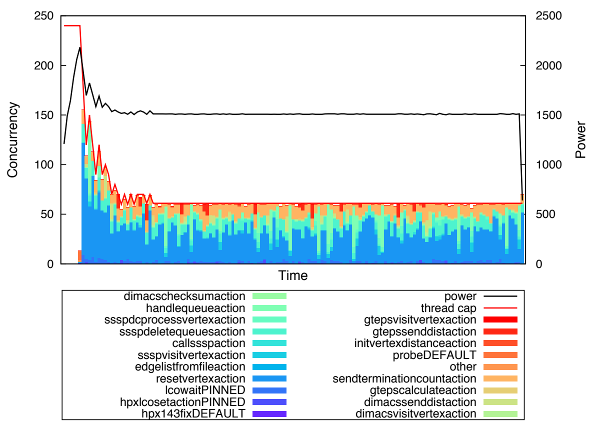
Fig. 4a shows the cumulative concurrency graph across all 10 nodes for the baseline execution. The concurrency charts show a stacked bar chart with the periodic (1Hz) instantaneous status of all threads. The red line indicates the maximum total number of threads (fixed at 240), and the black line is the instantaneous power measurement for each sample. In this run, 1962 searches are performed in 60 seconds. The graph shows that nearly all 240 threads are busy, and power consumption is about 240 W per node.

Fig. 4b shows the cumulative concurrency graph across all 10 nodes for the throttled execution using the policy engine. The total maximum number of threads starts at 240, but is throttled while Active Harmony is searching for an optimal number of active threads to maximize transaction throughput. As in the baseline figure, the evolving thread cap is the red line and the instantaneous power for each sample is the black line. In this execution, 6929 searches were performed in 60 seconds. When the search converges, only 61 (6 threads on 9 nodes, 7 threads on one node) threads are active. As a side-effect, power consumption is much lower, about 150 W per node. Most importantly, the number of searches done in the 60 seconds is several times higher. Fig. 4c shows the correlation between the throughput (total calls to `_handle_queue_action()`) and the evolving thread cap.

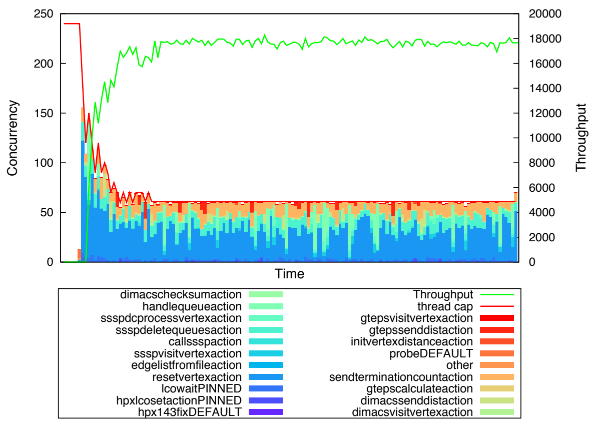
Tab. 4d shows a comparison of key metrics between the baseline and the runtime optimized executions of SSSP. In the throttled execution, the total cycles and instruction counts are reduced, while the number of L2 cache misses increases slightly. Because the graph is distributed, visiting remote vertices requires network communication. The network request causes a worker thread to yield the task waiting on the network request to perform other work, rather than block and wait on the result. The yield process is implemented using locks, so increased requests for the network lead to lock contention in the runtime. The yield algorithm also includes a small amount of "busy work", which explains the reduction in instructions. Essentially, this application implementation appears to be network-bound, so reducing the number of active worker threads decreases the contention for yielded tasks. As it can be seen in the table, the TEPS metrics are increased considerably by throttling, resulting in greater throughput. It is important to note that the problem is not with the runtime, but with the nature of the implementation. Because the graph is distributed, the threads contend while waiting on remote actions.



a) SSSP Baseline



b) SSSP Throttled, with power consumption side effect



c) SSSP Throttled, with throughput metric correlation to thread cap

Metric	Baseline	Throttled
Searches	1962	6929
Cycles	$6.9E+12$	$2.8E+12$
Instructions	$3.2E+12$	$2.0E+12$
L2TCM	$8.0E+9$	$8.6E+9$
IPC	0.45896	0.71626
INS/L2TCM	397.52	235.23
min. TEPS	$7.2E+04$	$9.5E+04$
med. TEPS	$1.4E+05$	$5.0E+05$
max. TEPS	$2.5E+05$	$7.6E+05$

d) Metric comparison between the baseline and throttled executions of SSSP

Figure 4. SSSP Benchmark

3.3. HPX-5 LULESH kernel

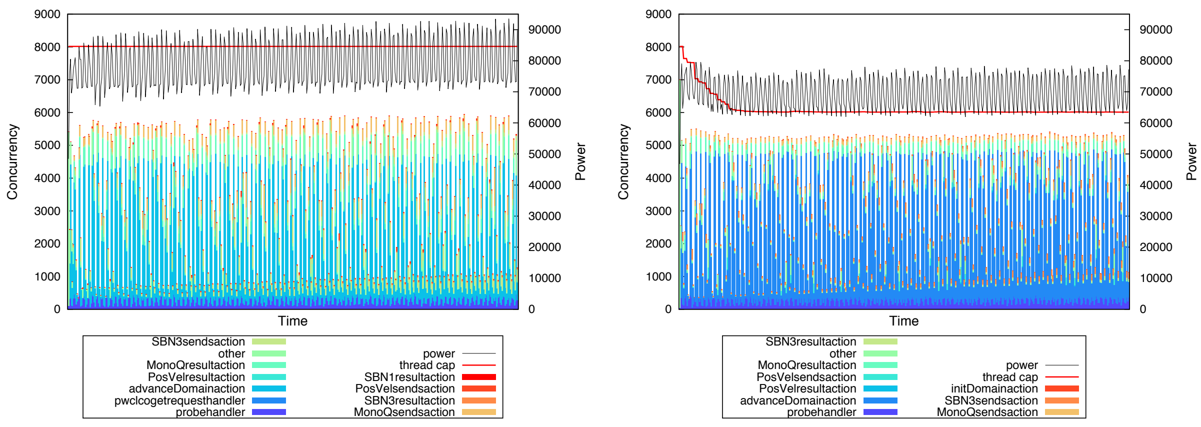
The Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) benchmark is one of the proxy applications for the US Department of Energy co-design efforts for exascale. LULESH is an application from the Lawrence Livermore National Laboratory (LLNL) that is used to model and study hydrodynamics, the motion of materials relative to each other when subject to forces. The HPX-5 LULESH implementation was written by the HPX-5 researchers at Indiana University. Because LULESH is CPU bound in most implementations, it is an interesting test case to demonstrate what happens when executed under a power cap. As it is CPU bound, reducing the power consumption typically involves using fewer threads or slowing down the CPU clock speed, which will affect performance.

For this example, we developed an APEX policy for maintaining power draw within a high/low range. The policy will periodically check the power draw, and if the current power draw is greater than the high power cap, the thread cap will be reduced. If the power draw is lower than the low power cap, the thread cap will be increased. The policy rule is a simple hill-climbing algorithm with hysteresis, using a running average of the last N observations. In our tests, we set $N = 3$. We modified the HPX-5 thread scheduler algorithm to check the thread cap on every iteration of the main worker loop. If a thread is not holding any resources and the number of active workers is greater than the current thread cap, the thread goes into an idle state until signaled to resume work. If the number of active workers is less than the cap, an active worker signals an idle thread to resume working. A quiescent node of Edison draws approximately 40W, whereas a fully loaded node draws as much as 300W. We used a high power cap of 220W and a low cap of 200W. We executed LULESH with 8000 sub-domains, $nx = 64$, for 100 iterations on 334 nodes of Edison (8016 total cores).

Fig. 5a shows the cumulative concurrency graph across all 334 nodes for the baseline execution. The total runtime of the application is 118 seconds. The red line shows the maximum concurrency, 8000 threads (fixed). The black line shows the cumulative power draw across all 334 nodes. The power consumption has peaks around 9.3kW, about 278W per node. The average power draw per node was around 236W. The total energy usage was measured as 9.327MJ (megajoules). The stacked bar chart shows which tasks were executing when APEX sampled them with a 4Hz period.

Fig. 5b shows the cumulative concurrency graph across all 334 nodes for the throttled execution using the policy engine. The key difference between the two executions is that the total energy draw for the throttled execution was only 8.180MJ (approximately 12.3% less) while the execution time was not affected. The red line shows the thread cap as it is modified by the policy. The black line shows the reflected reduction in power draw with some localized fluctuations. The average power draw per node for this run was 207 W. Once the search had converged, this execution used approximately 3/4 of the number of threads, but runtime was unaffected.

Like the SSSP benchmark, the throttled version of LULESH does not yield tasks as much as the original. A sampled TAU profile showed much less time spent in yielding activity – when a worker thread surrenders its task in order to stay busy while waiting on a remote result. Our conclusion is that the assignment of sub-domains to localities in HPX does not maintain spatial locality, but rather assigns them round-robin to distribute the work. The HPX-5 implementation is being rewritten in order to exploit spatial locality and put less pressure on the network.



a) LULESH Baseline **b)** LULESH Throttled

Figure 5. LULESH Benchmark

3.4. HPX-3 miniGhost kernel

MiniGhost [5], developed as part of the Mantevo project [14], is a finite difference miniapp simulating heat diffusion over a three-dimensional domain. The original version uses OpenMP intra-node and MPI inter-node. It has been ported to HPX-3 [2]; this version uses HPX for both intra- and inter-node parallelism. The HPX version provides better performance than the original OpenMP version.

Fig. 6 shows that there are diminishing returns from allocating additional worker threads to MiniGhost. This suggests that we can throttle the application by cutting back on the number of worker threads to reduce energy usage while avoiding substantial performance degradation.

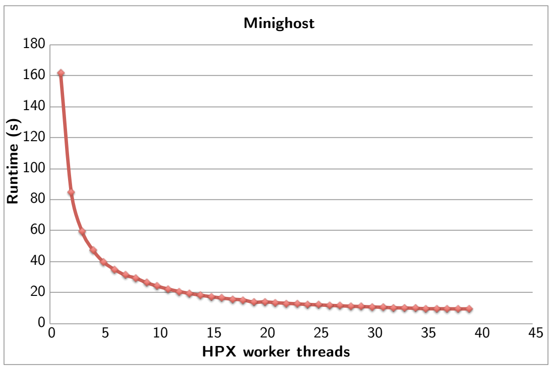
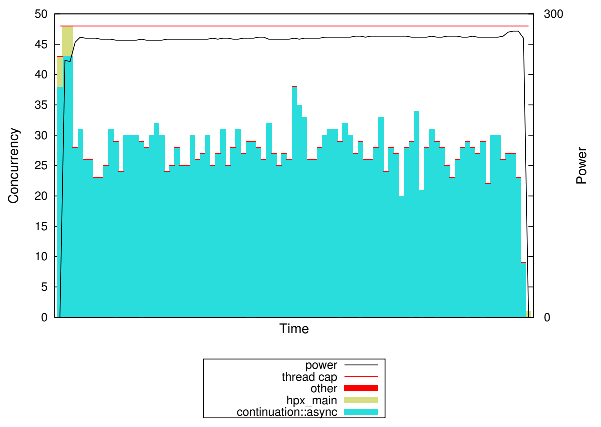
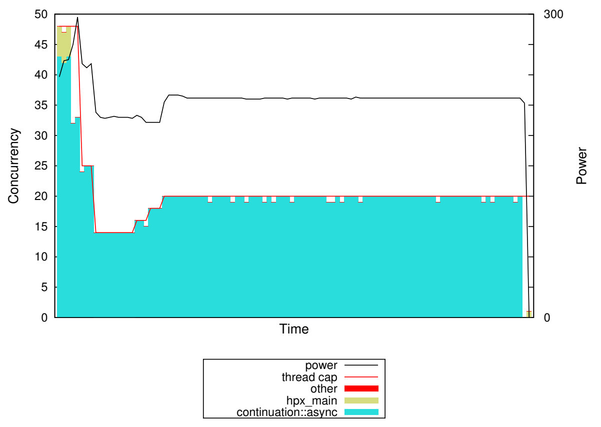


Figure 6. miniGhost strong scaling

Fig. 7a shows the concurrency with 48 worker threads, the number of logical cores on an Edison node. While not all available worker threads are used, the application will often use slightly more than 24 available physical cores. With 48 worker threads, MiniGhost runs in 92 seconds and uses about 275 Watts of power. Fig. 7b shows the concurrency when the initial number of worker threads is set to 48 but the thread cap is dynamically adjusted to keep power at or below 200 Watts. APEX converges on the thread cap of 20, yielding 200 Watts of power usage, a 33% of reduction in power, and the runtime of 103 seconds, 12% of increase in runtime.



a) miniGhost Baseline. This concurrency chart shows a stacked bar chart with the periodic (1Hz) status of each thread. The max number of threads is 48 (red line), and the instantaneous power for each sample is the black line



b) miniGhost Throttled. This concurrency chart shows a stacked bar chart with the periodic (1Hz) status of each thread. The max number of threads starts at 48, but is throttled while APEX searches for an optimal number of active threads to keep under the power cap. The evolving thread cap is the red line and the instantaneous power for each sample is the black line

Figure 7. miniGhost Benchmark

4. Related Work

Several performance tools use measurement for the purposes of offline performance analysis, including TAU [32], HPCToolkit [1], Scalasca [38], Vampir [20], Extrae [27] and others. All are powerful and capable tools in their own right. These tools, however, were designed for offline performance analysis and tuning, focusing on first-person performance measurement of tied tasks on a per-thread (OS thread) basis. New and emerging exascale programming models present technical challenges that the designers of those measurement systems had not considered, such as untied task execution and migration, runtime thread control and execution, third-person observation, and runtime performance tuning. Also, as these tools are inescapably intrusive, they are not designed to be integrated permanently into an application for continuous performance introspection, but rather to be used in an iterative execute-analyze-tune cycle. In contrast, APEX is designed to perform asynchronous first- and third-person measurement for the sole purpose of supporting runtime introspection and performance adaptation.

One of the most active research areas in HPC is to reduce energy consumption while maintaining and even improving performance. For example, Curtis-Maury *et al.* [11] demonstrated the ability to build a runtime-adaptable optimization that both converges on the best performing configuration and reduces power consumption. This result is due to the observation that some parallel applications have diminishing returns with respect to scalability, and additional hardware merely consumes more power without improving performance. Rountree *et al.* [30] demonstrate the use of dynamic voltage scaling to save energy while minimizing impact on the performance. Their *Adagio* approach attempts to scale computation and communication in distributed MPI applications using only local information acquired and applied at runtime in order to eliminate slack at synchronization points. Rountree *et al.* [29] have subsequently explored the inherent variation among processors and the range of effects that placing a hard power cap has on applications with different characteristics.

With respect to runtime thread scheduling, Olivier *et al.* [26] demonstrated that a hierarchical, cache-aware thread scheduler performs better than a flat task scheduling in conjunction with load balancing (via task stealing) within cache and/or NUMA domains. While this is a form of runtime adaptation, it is an approach targeting one issue and does not react to runtime measurements, but rather uses thread affinity and memory hierarchy information at startup. Similarly, Charm++ [19, 39] has mechanisms for distributed dynamic load-balancing based on runtime information. Other researchers have used Charm++ as a platform for developing additional runtime load-balancing strategies [17] both between nodes and within a node using cache/memory hierarchical information. PICS [34] allows runtime adaptivity in Charm++ by allowing the application to register *control points* [12] specifying what effect application parameters have on various categories of performance-effecting properties. For example, the application can register that a variable controlling the size of a subproblem will change the grain size and degree of parallelism. Based on runtime performance measurement, the system selects a property to adjust and adjusts registered control points accordingly.

The OmpSs runtime system has demonstrated the ability to schedule an appropriate kernel implementation based on available heterogeneous hardware choices [13, 28]. In this implementation, DGEMM tasks are scheduled on either CPU or GPU resources depending on the input size, available hardware, and prior performance results.

The Open Tool for Parameter Optimization [8] tunes parameters exposed by the OpenMPI runtime. In OpenMPI, many runtime tasks are delegated to modules, which implement different versions of communication algorithms (such as collectives) and map MPI operations onto lower-level network operations (such as for TCP, InfiniBand, Cray Gemini/Aries, etc.). These modules expose a set of tunable parameters, called MCA parameters, as the result of which a typical installation will have several hundred. OTPO searches for parameters giving the best performance, as measured by latency or bandwidth of network operations.

The AutoTune project [24] is developing the Periscope Tuning Framework, an extension to the earlier Periscope [6] performance analysis and diagnosis tool which allows plugins to provide new functionality. PTF has been used for runtime energy tuning using DVFS and for tuning of MPI runtime parameters [25], and it has been integrated with several parallel pattern libraries to tune parameters such as how many CPU cores and acclelerators to use in heterogenous codes and what scheduling policies to use [4]. APEX differs from PTF in being more deeply integrated with runtimes and in providing tuning capabilities based on a global performance view.

Hoffman *et al.* [15] have developed an interface for diverse applications to report a performance measure in a generic way so that operating systems and runtimes can adapt themselves to optimize application performance. In their *Application Heartbeats* framework, applications signal a “heartbeat” as they make progress in a computation; for example, a video-encoding application could signal a heartbeat each time it processes a frame. The system then tries to optimize the observed “heart rate”. They provide examples of optimizations purely within an application, such as a video encoder switching algorithms and altering parameters to algorithms to meet a target frame rate, and outside applications, such as a computer-vision application that adjusts the number of cores that it uses to find the smallest number of cores necessary to achieve real-time video processing.

5. Conclusion

The quest for exascale brings fundamentally new challenges to performance and productivity. The solutions that will likely usher in the exascale era will require software designers and users to embrace performance heterogeneity and variability. We believe that any successful implementation will have to integrate performance introspection, *in situ* analysis and adaptation in an exascale system stack. The XPRESS project has developed a prototype of APEX integrated with HPX-3 and HPX-5 for use in OpenX. We have demonstrated APEX with several benchmark examples, and we believe that the APEX framework is generally applicable to other X-stack runtime efforts.

There is considerable work that can be done with respect to APEX. In the short term, we would like to conduct more robust application experiments and to explore behavior larger scales on different platforms. As more applications are developed using HPX, we hope to have a greater opportunity to demonstrate the APEX capabilities for runtime adaptation. With that in mind, new applications will present more and better policy (optimization) rules, both for specific applications and to generalize them in the operating system and runtime libraries. In particular, we are interested in possible policy rules that address heterogeneous HPX-3 code that can be executed on GPGPUs, as well as many-core architectures such as the Intel Phi. We plan to develop more policy rules that specifically address the SLOWER design principles of the ParalleX model [33]. We soon will be exploring the multi-objective optimization opportunities available in the development branch of Active Harmony. With that support, we can tune with respect to both performance and energy efficiency, as well as to any other application-specific metrics. Finally, we believe that APEX has applications outside of the XPRESS project, and that it can be successfully integrated into other runtime systems and parallel execution models with controllable parameters, including OpenMP, MPI, and OmpSs. It can serve as a framework for triggering application-specific optimizations such as adaptive mesh refinement, load balancing, and other dynamic behavior.

Support for this work was provided through the X-Stack Software Research and Scientific Discovery through Advanced Computing (SciDAC) programs funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics) under award numbers DE-SC0008638, DE-SC0008704, DE-FG02-11ER26050 and DE-SC0006925. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. DOE's National Nuclear Security Administration under contract DE-AC04-94AL85000. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

1. L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. Hpctoolkit: tools for performance analysis of optimized parallel programs http://hpctoolkit.org. *Concurr. Comput. : Pract. Exper.*, 22:685–701, April 2010. DOI: 10.1002/cpe.v22:6.

2. Vinay C Amatya. *Parallel Processes in HPX: Designing an Infrastructure for Adaptive Resource Management*. PhD thesis, Louisiana State University, 2014.

3. Matthew Anderson, Maciej Brodowicz, Hartmut Kaiser, and Thomas L. Sterling. An Application Driven Analysis of the ParalleX Execution Model. *CoRR*, abs/1109.5201, 2011. http://arxiv.org/abs/1109.5201.

4. Enes Bajrovic, Siegfried Benkner, Jiri Dokulil, and Martin Sandrieser. Autotuning of pattern runtimes for accelerated parallel systems. In *PARCO 2013, September 2013, Munich, Germany*, September 2013.

5. Richard F Barrett, Courtenay T Vaughan, and Michael A Heroux. MiniGhost: a miniapp for exploring boundary exchange strategies using stencil computations in scientific parallel computing. Technical Report SAND2012-10431, 2011.

6. Shajulin Benedict, Ventsislav Petkov, and Michael Gerndt. Periscope: An online-based distributed performance analysis tool. In *Tools for High Performance Computing 2009*, pages 1–16. Springer. DOI: 10.1007/978-3-642-11261-4_1.

7. Ron Brightwell and Kevin Pedretti. An intra-node implementation of OpenSHMEM using virtual address space mapping. In *Fifth Partitioned Global Address Space Conference*, October 2011.

8. Mohamad Chaarawi, Jeffrey M. Squyres, Edgar Gabriel, and Saber Feki. A tool for optimizing runtime parameters of open MPI. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, number 5205 in Lecture Notes in Computer Science, pages 210–217. Springer Berlin Heidelberg. DOI: 10.1007/978-3-540-87475-1_30.

9. Sandia Corporation. eXascale PRogramming Environment and System Software (XPRESS). `http://xstack.sandia.gov/xpress/`, April 2015.

10. Cristian Ţăpuş, I-Hsin Chung, and Jeffrey K. Hollingsworth. Active harmony: Towards automated performance tuning. In *2002 ACM/IEEE Conference on Supercomputing*, SC '02, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

11. Matthew Curtis-Maury, James Dzierwa, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In *20th Annual International Conference on Supercomputing*, ICS '06, pages 157–166, New York, NY, USA, 2006. ACM. DOI: 10.1145/1183401.1183426.

12. Isaac J Dooley. *Intelligent runtime tuning of parallel applications with control points*. PhD thesis, University of Illinois at Urbana-Champaign, 2011.

13. Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesüs Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. OmpSs: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21(02):173–193, 2011. DOI: 10.1142/S0129626411000151.

14. Michael Heroux and Richard Barrett. Mantevo project, 2011.

15. Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments. In *7th International Conference on Autonomic Computing*, ICAC '10, pages 79–88, New York, NY, USA, 2010. ACM. DOI: 10.1145/1809049.1809065.

16. Kevin A. Huck, Allen D. Malony, Sameer Shende, and Alan Morris. TAUg: Runtime global performance data access using MPI. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 4192 of *Lecture Notes in Computer Science*, pages 313–321. Springer Berlin Heidelberg, 2006. DOI: 10.1007/11846802_44.

17. E. Jeannot, E. Meneses, G. Mercier, F. Tessier, and Gengbin Zheng. Communication and topology-aware load balancing in charm++ with treematch. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8, Sept 2013. DOI: 10.1109/CLUSTER.2013.6702666.

18. Hartmut Kaiser, Maciej Brodowicz, and Thomas Sterling. ParalleX: An advanced parallel execution model for scaling-impaired applications. In *Parallel Processing Workshops*, pages 394–401, Los Alamitos, CA, USA, 2009. IEEE Computer Society. DOI: 10.1109/ICPPW.2009.14.

19. Laxmikant V. Kale and Gengbin Zheng. Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects. In *Advanced Computational Infrastructures for Parallel and Distributed Applications*, pages 265–282. Wiley-Interscience, 2009. DOI: 10.1002/9780470558027.ch13.

20. Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S Müller, and Wolfgang E Nagel. The Vampir performance analysis tool-set. In *Tools for High Performance Computing*, pages 139–155. Springer, 2008. DOI: 10.1007/978-3-540-68564-7_9.

21. Anirban Mandal, Rob Fowler, and Allan Porterfield. Modeling memory concurrency for multi-socket multi-core systems. In *2010 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS2010)*, pages 56–75, White Plains, NY, March 2010. IEEE. DOI: 10.1109/ispass.2010.5452064.

22. Anirban Mandal, Rob Fowler, and Allan Porterfield. System-wide introspection for accurate attribution of performance bottlenecks. In *Second International Workshop on High-perfromance Infrastruture for Scalable Tools*, 2012.

23. Steven J. Martin and Matthew Kappel. Cray XC30 Power Monitoring and Management. In *Cray User Group Conference Proceedings*, 2014.

24. Renato Miceli, Gilles Civario, Anna Sikora, Eduardo César, Michael Gerndt, Houssam Haitof, Carmen Navarrete, Siegfried Benkner, Martin Sandrieser, Laurent Morin, and François Bodin. AutoTune: A plugin-driven approach to the automatic tuning of parallel applications. In *Applied Parallel and Scientific Computing*, number 7782 in Lecture Notes in Computer Science, pages 328–342. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-36803-5_24.

25. Yury Oleynik, Robert Mijaković, IsaíasA. Comprés Ureña, Michael Firbach, and Michael Gerndt. Recent advances in periscope for performance analysis and tuning. In *Tools for*

High Performance Computing 2013, pages 39–51. Springer International Publishing, 2014. DOI: 10.1007/978-3-319-08144-1_4.

26. Stephen L. Olivier, Allan K. Porterfield, Kyle B. Wheeler, and Jan F. Prins. Scheduling task parallelism on multi-socket multicore systems. In *International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '11, pages 49–56, New York, NY, USA, 2011. ACM.

27. Vincent Pillet, Jesús Labarta, Toni Cortes, and Sergi Girona. Paraver: A tool to visualize and analyze parallel code. In *Proceedings of WoTUG-18: Transputer and occam Developments*, volume 44, pages 17–31. mar, 1995.

28. J. Planas, R.M. Badia, E. Ayguade, and J. Labarta. Self-adaptive OmpSs tasks in heterogeneous environments. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 138–149, May 2013. DOI: 10.1109/IPDPS.2013.53.

29. Barry Rountree, Dong H Ahn, Bronis R de Supinski, David K Lowenthal, and Martin Schulz. Beyond DVFS: A first look at performance under a hardware-enforced power bound. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pages 947–953. IEEE, 2012. DOI: 10.1109/ipdpsw.2012.116.

30. Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making dvs practical for complex hpc applications. In *23rd International Conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM. DOI: 10.1145/1542275.1542340.

31. Sandia National Laboratories. *The Kitten Lightweight Kernel*. `https://software.sandia.gov/trac/kitten`.

32. S. Shende and A. D. Malony. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–331, Summer 2006. DOI: 10.1177/1094342006064482.

33. Thomas Sterling, Daniel Kogler, Matthew Anderson, and Maciej Brodowicz. Slower: A performance model for exascale computing. *Supercomputing frontiers and innovations*, 1(2):42–57, 2014. DOI: 10.14529/jsfi140203.

34. Yanhua Sun, Jonathan Lifflander, and Laxmikant V. Kalé. PICS: A performance-analysis-based introspective control system to steer parallel applications. In *International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '14, pages 5:1–5:8, New York, NY, USA, 2014. ACM. DOI: 10.1145/2612262.2612266.

35. Alexandre Tabbal, Matthew Anderson, Maciej Brodowicz, Hartmut Kaiser, and Thomas Sterling. Preliminary design examination of the ParalleX system from a software and hardware perspective. *SIGMETRICS Performance Evaluation Review*, 38:4, Mar 2011.

36. The National Energy Research Scientific Computing Center (NERSC). Edison. `https://www.nersc.gov/users/computational-systems/edison/`, April 2015.

37. Thomas Williams and Colin Kelley. Gnuplot homepage. `http://www.gnuplot.info`, April 2015.

38. Felix Wolf, Brian J. N. Wylie, Erika Abraham, Daniel Becker, Wolfgang Frings, Karl Furlinger, Markus Geimer, Marc-Andre Hermanns, Bernd Mohr, Shirley Moore, Matthias

Pfeifer, and Zoltan Szebenyi. Usage of the scalasca toolset for scalable performance analysis of large-scale parallel applications. In *Tools for High Performance Computing*, pages 157–167. Springer Berlin Heidelberg, 2008.

39. Gengbin Zheng, E. Meneses, A. Bhatele, and L.V. Kale. Hierarchical load balancing for charm++ applications on large supercomputers. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 436–444, Sept 2010. DOI: 10.1109/ICPPW.2010.65.

Received June 7, 2015.

DOI: 10.14529/jsfi150306

Visualization for Exascale: Portable Performance is Critical

***Kenneth Moreland*[1], *Matthew Larsen*[2], *Hank Childs*[2]**

Researchers face a daunting task to provide scientific visualization capabilities for exascale computing. Of the many fundamental changes we are seeing in HPC systems, one of the most profound is a reliance on new processor types optimized for execution bandwidth over latency hiding. Multiple vendors create such accelerator processors, each with significantly different features and performance characteristics. To address these visualization needs across multiple platforms, we are embracing the use of data parallel primitives that encapsulate highly efficient parallel algorithms that can be used as building blocks for conglomerate visualization algorithms. We can achieve performance portability by optimizing this small set of data parallel primitives whose tuning conveys to the conglomerates. In this paper we provide an overview of how to use data parallel primitives to solve some of the most common problems in visualization algorithms. We then describe how we are using these fundamental approaches to build a new toolkit, VTK-m, that provides efficient visualization algorithms on multi- and many-core architectures. We conclude by reviewing a comparison of a visualization algorithm written with data parallel primitives and separate versions hand written for different architectures to show comparable performance with data parallel primitives with far less development work.

Keywords: scientific visualization, exascale, performance portability, data parallel primitives.

Introduction

Although the basic architecture for high-performance computing platforms has remained homogeneous and consistent for over a decade, revolutionary changes are coming. Power constraints and physical limitations are impelling the use of new types of processors, heterogeneous architectures, and deeper memory and storage hierarchies. Such drastic changes propagate to the design of software that is run on these high-performance computers and how we use them.

The predictions for extreme-scale computing are dire. Recent trends, many of which are driven by power budgets, which max out at 20 MW [18], indicate that future high-performance computers will have different hardware structure and programming models to which software must adapt. The predicted changes from petascale to exascale are summarized in tab. 1.

A particularly alarming feature of tab. 1 is the increase in concurrency of the system: up to 5 orders of magnitude. This comes from an increase in both the number of cores as well as the number of threads run per core. (Modern cores employ techniques like hyperthreading to run multiple threads per core to overcome latencies in the system.) We currently stand about halfway through the transition from petascale to exascale and we can observe this prediction coming to fruition through the use of accelerator or many-core processors. In the November 2014 Top500 supercomputer list, 75 of the computers contain many-core components, including half of the top 10 computers.

A many-core processor achieves high instruction bandwidth by packing many cores onto a single processor. To achieve the highest density of cores at the lowest possible power requirement, these cores are trimmed of latency-hiding features and require careful coordination to achieve peak performance. Although very scalable on distributed memory architectures, our current parallel scientific visualization tools, such as ParaView [2] and VisIt [6], are inadequate on these machines.

[1]Sandia National Laboratories, Albuquerque, USA
[2]University of Oregon, Eugene, USA

Table 1. Comparison of a petascale supercomputer to an expected exascale supercomputer [1]

		Exascale (Prediction)		
System Parameter	Petascale	Swim Lane 1	Swim Lane 2	Factor Change
System Peak	2 PF	1 EF		500
Power	6 MW	≤20 MW		3
System Memory	0.3 PB	32–64 PB		100–200
Node Performance	125 GF	1 TF	10 TF	8–80
Node Core Count	12	1,000	10,000	83–830
Core Concurrency	1	10	100	10–100
Node Concurrency	12	10,000	1,000,000	830–83,000
System Size (nodes)	18700	1,000,000	100,000	50–500
Total Concurrency	225 K	1 B×10	1 B×100	40,000–400,000
Network BW	1.5 GB/s	100 GB/s	1,000 GB/s	66–660
I/O Capacity	15 PB	300–1,000 PB		20–67
I/O BW	0.2 TB/s	20–60 TB/s		100–300

Overhauling our software tools is one of the principal visualization research challenges today [7]. A key strategy has been the use of data parallel primitives, since the approach enables simplified algorithm development and helps to achieve portable performance.

1. Data Parallel Primitives

Data parallelism is a programming model in which processing elements perform the same task on different pieces of data. Data is arranged in long vectors, and the base tasks apply an operation across all the entities in one or more vectors. Using a sequence of data parallel primitives simplifies expressing parallelism in an algorithm and simplifies porting across different parallel devices. It takes only a few select data parallel primitives to efficiently enable a great number of algorithms [5].

Scientific visualization algorithms typically use data parallel primitives like map, scan, reduce, and sort, which are commonly available in parallel libraries [4, 16]. Several recent research projects for visualization software on next-generation architectures such as Dax [13], PISTON [9], and EAVL [11] use this data parallel approach to execute algorithms [17]. Based on this core similarity, a new project — VTK-m — is combining their respective strengths in execution and data models into a unified framework.

2. Patterns for Data Parallel Visualization

Using data parallel primitives greatly simplifies the process of implementing algorithms on highly-threaded machines and makes these algorithms performance portable. However, implementing many scientific algorithms in terms of data parallel primitives like scan and sort is not straightforward. Fortunately, many scientific visualization algorithms follow familiar algorithmic structures [14], and common patterns emerge.

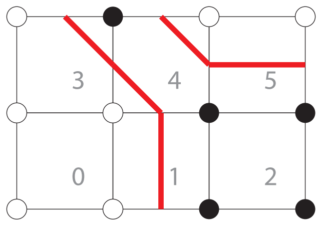


Figure 1. Mesh for contour algorithm examples

Three very common patterns in scientific visualization are stream compaction, reverse index lookup, and topology consolidation. In this section we describe these patterns using a Marching-Square-like algorithm applied to the simple example mesh shown in fig. 1.

2.1. Stream Compaction

One common feature of visualization algorithms is that the size of the output might depend on the data values in the input and cannot be determined without first analyzing the data. For example, in the mesh of fig. 1 we note that there is no contour in cells 0 and 2, a single contour line in cells 1, 3, and 5, and two contour lines in cell 4. When generating these contour segments in parallel, it is not known where to place the results. We could allocate space assuming the worst case scenario that every cell has the maximum number of contour segments, but that guess tends to be much larger than the actual required memory. Instead, we want to pack the result tightly in an array. This process is known as *stream compaction*. Stream compaction can be performed in two data parallel operations, which are demonstrated in fig. 2 (adapted from Lo, Sewell, and Ahrens [9]).

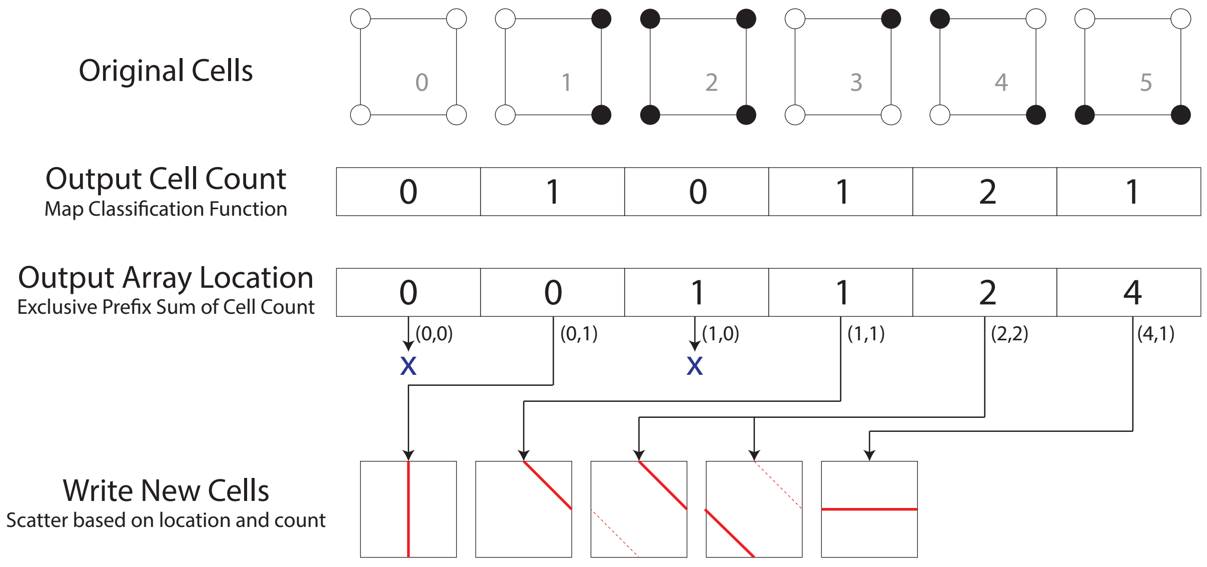


Figure 2. Steps to perform the stream compaction pattern using data parallel primitives

Firstly, a mapping operation is performed to count the size of the output per cell. Secondly, an exclusive prefix sum (scan) operation is performed. The result of the prefix sum for each entry is the sum of all output up to that point. This sum can be directly used as an index into the compact output array. A final map of the per-element algorithm can now run, placing its results into the appropriate location of the output array.

2.2. Reverse Index Lookup

Directly using the indices from the stream compaction operation results in a *scatter* operation where each thread takes data from an input element and writes to one or more output elements using random access. Although the scatter done by the basic stream compaction is functionally correct, it is known that current many-core processors tend to perform better with *gather* operations where each thread is assigned a single output element but can access random input elements [19]. The steps to reverse the index lookup from a scatter to a gather are demonstrated in fig. 3.

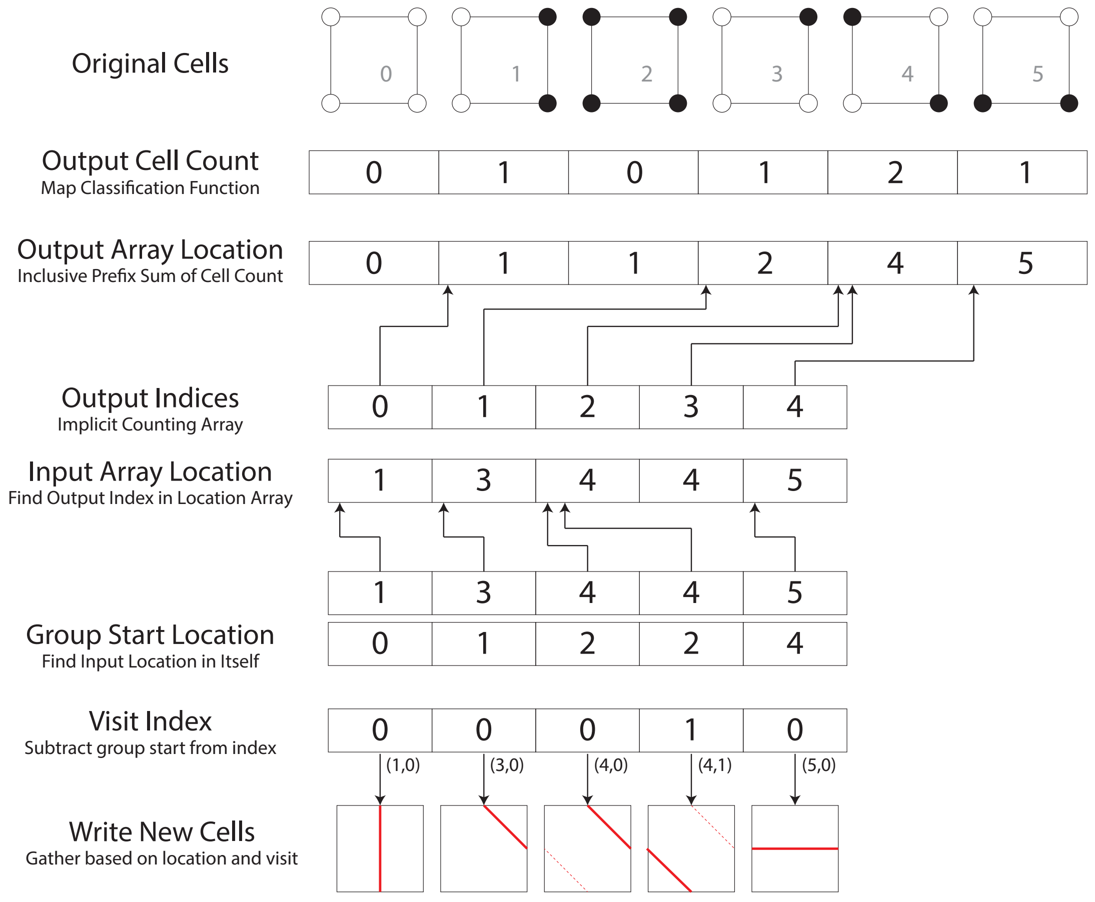


Figure 3. Steps to perform a reverse lookup after stream compaction using data parallel primitives

We start with an array that maps each input to the location in its corresponding output location. However, we generate this output array location using an inclusive scan rather than an exclusive scan. This has the effect of shifting the array to the left by one to make the indexing of the next step work better. The next step is to search for the upper bound of the array location for each output element index. The upper bound will be the first entry greater than the value we search for. This search requires the target array location indices to be sorted, which it is assuredly because it is generated from a prefix sum. The search for every index can be done independently in parallel.

The results from the upper bound give the reverse map from output index to input index. However, a problem that arises is that multiple output elements may come from the same input elements but are expected to produce unique results. In this example input cell 4 produces two contour elements, so two entries in the output array point to the same input cell. How are the two threads running on the same input cell know which element to produce? We solve this problem by generating what we call a *visit index.*

The visit indices are generated in two steps. First, we perform a lower bound search of each value in the input array location map into the same map. The lower bound search finds the last entry less than or equal to the value we search for in parallel. The result is the index to the first entry in the input array location map for the group associated with the same input element. Then we take this array of indices and subtract them from the output index to get a unique index into that group. We call this the visit index. Using the pair from input array location map and the visit index, each thread running for a single output element can uniquely generate the data it is to produce.

2.3. Topology Consolidation

Another common occurrence in visualization algorithms is for independent threads to redundantly create coincident data. For example, output elements 0 and 3 from fig. 2 and fig. 3 come from cells 1 and 4, respectively, in fig. 1 and share a vertex. This shared vertex is independently interpolated in separate threads and the connection of these output elements is lost. It is sometimes required to consolidate the topology by finding these coincident elements and merging them.

The general approach to topology consolidation is to define a simple hash for each element that uniquely identifies the element for all instances. That is, two hash values are equal if and only if the associated elements are coincident. Once hashes are generated, a sort keyed on the hashes moves all coincident elements to be adjacent in the storage arrays. At this point it is straightforward to designate groups of coincident elements and reduce the groups to a single element in parallel.

For the specific case of merging vertices, Bell [3] proposes using the point coordinate triple as the hash. However, that approach is intolerant to any numerical inaccuracy. A better approach is to use integer-based hashes, which can usually be derived from the input topology. For example, contour algorithms like Marching Cubes always define contour vertices on the edges of the input mesh. These edges (and therefore the vertices) can be uniquely defined either by an enumerated index or by the pair of indices for the edge's vertex endpoints. Miller, Moreland, and Ma [12] show this approach is faster than using point coordinates and can also be applied to topological elements other than vertices.

3. Building a Framework

We are taking the concepts of data parallel primitives and the patterns built on top of them and using them to build a visualization toolkit for multi- and many-core systems called *VTK-m*. VTK-m is a separate project from the similar VTK software and has a very different organization although the two toolkits can be used together to great effect.

At its core, VTK-m uses data parallel primitives to achieve performance portability. VTK-m defines a unit named a *device adapter* on which all parallel features within VTK-m are based. The device adapter of course provides the basic routines necessary to control the device such as allocating memory, transferring data, and scheduling parallel jobs. Additionally, the device adapter comes replete with the data parallel primitives scan, sort, and reduce with several variations. It is typical for these data parallel primitives to have very different implementations on different architectures, and while the implementation of efficient versions on each architecture can be challenging, this ultimately comprises only a small section of the code within VTK-m. Also, these data parallel primitives are very general, so the VTK-m implementation often shares the implementation provided elsewhere for more general purposes.

The patterns discussed in Section 2, which build upon data parallel primitives to form common visualization operations are also well utilized within VTK-m, but elsewhere in the framework. Rather, a unit called a *dispatcher* stands in between the device adapter and a specific algorithm implementation, and this is where these design patterns are employed. A dispatcher is responsible for analyzing the needs of an algorithm (inputs and outputs as well as execution requirements) and builds the necessary infrastructure to allow the algorithm to run without concern about parallel operation.

Depending on the type of algorithm a dispatcher is invoking, it might implement any number of these patterns. For example, if an algorithm does not have a one-to-one mapping between input and output values, the dispatcher will likely require the use of the stream compaction and reverse index lookup patterns. If an algorithm is generating new topology, it likely will have replicated elements that will benefit from the topology consolidation pattern.

4. Results

One of the promises of using data parallel primitives to build scientific visualization algorithms is performance portability. That is, a single implementation using data parallel primitives should work well across computing devices with vastly different performance characteristics from traditional latency-optimized multi-core CPUs to bandwidth-optimized many-core GPUs. Furthermore, portable data parallel primitive implementations should have close to the performance of a non-portable algorithm designed and optimized specifically for a particular device. Recent research indicates that data parallel primitive algorithms are in fact quite performance portable.

Maynard et al. [10] compare a threshold algorithm written with data parallel primitives across many devices. The algorithm shows good performance on both multi-core CPU and many-core GPU devices. Interestingly, the data parallel primitive algorithm running serially on a single CPU core still beats the equivalent VTK implementation.

Lo, Sewell, and Ahrens [9] demonstrate the performance of a Marching Cubes algorithm implemented with data parallel primitives. Their algorithm is compared with the equivalent CUDA reference implementation optimized for that architecture. The two implementations get comparable performance. The data parallel primitive implementation is also shown to get good performance and scalability on multi-core CPUs.

But perhaps the most encouraging evidence comes from a recent performance study conducted by Larsen et al. [8] for ray tracing in the context of data parallel primitives. Ray tracing is a challenging use case since it is computationally intense and contains both regular and irregular memory access patterns. Moreover, this is an algorithm with "guaranteed not to exceed" standards, in the form of Intel's Embree [20] and NVIDIA's OptiX [15]. These products each are supported by teams of developers and have been under development for multiple years. Further, they make full use of architectural knowledge, including constructs like intrinsics, and tune for Intel and NVIDIA products, respectively.

Larsen implements his ray tracer within EAVL and provides a performance study against OptiX on multiple NVIDIA GPUs and against Embree on Intel Xeon and Xeon Phi architectures.

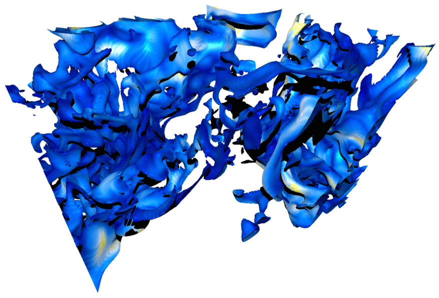


Figure 4. Rendering from ray tracing study on an isosurface of 650,000 triangles

His study includes both scientific data sets and standard ray tracing data sets (e.g., Stanford dragon). Fig. 4 shows one of the scientific data sets.

Encouragingly, the performance comparison finds that the EAVL ray tracer is competitive with the industry standards. It is within a factor of two on most configurations and does particularly well on the scientific data sets. In fact, it even outperforms the industry standards on some older architectures (since the industry standards tend to focus on the latest architectures).

Overall, this result is encouraging regarding the prospects for portable performance with data parallel primitives, in that a single, architecture-agnostic implementation was comparable to two highly-tuned, architecture-specific standards. Although the architecture-specific standards are clearly faster, the gap is likely acceptable for our use case. Further, the data parallel primitive approach is completed by a graduate student in a period of months whereas the industry standards take experts years (or more); the encumbrence from data parallel primitives could actually be even smaller given additional effort and expertise.

5. Conclusion

Visualization software will need significant changes to excel in the exascale era, both to deal with diverse architectures and to deal with massive concurrency within a node. Recent results show that data parallel primitives are a promising technology to deal with both challenges. Firstly, exploration into multiple algorithms have shown recurring trends, and will hopefully serve as a precursor to porting many of our community's algorithms reusing these same trends. Secondly, studies comparing performance with architecture-specific implementations have shown that the performance is very good. Researchers in this area — including the authors of this paper — are so encouraged that they have banded together to form a new effort, VTK-m, in an endeavor to provide production visualization software to the HPC community.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Numbers 10-014707, 12-015215, and 14-017566.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

1. Sean Ahern, Arie Shoshani, Kwan-Liu Ma, et al. Scientific discovery at the exascale. Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization, February 2011. `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Exascale-ASCR-Analysis.pdf`.

2. Utkarsh Ayachit. *The ParaView Guide: A Parallel Visualization Application.* Kitware Inc., 4.3 edition, January 2015. ISBN 978-1-930934-30-6, `http://www.paraview.org/paraview-guide/`.

3. Nathan Bell. High-productivity CUDA development with the thrust template library. GPU Technology Conference, 2010. `http://www.nvidia.com/content/pdf/sc_2010/theater/bell_sc10.pdf`.

4. Nathan Bell and Jared Hoberock. *GPU Computing Gems, Jade Edition*, chapter Thrust: A Productivity-Oriented Library for CUDA, pages 359–371. Morgan Kaufmann, October 2011.

5. Guy E. Blelloch. *Vector Models for Data-Parallel Computing.* MIT Press, 1990. ISBN 0-262-02313-X, `https://www.cs.cmu.edu/~guyb/papers/Ble90.pdf`.

6. Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*, pages 357–372. October 2012.

7. Hank Childs, Berk Geveci, Will Schroeder, Jeremy Meredith, Kenneth Moreland, Christopher Sewell, Torsten Kuhlen, and E. Wes Bethel. Research challenges for visualization software. *IEEE Computer*, 46(5):34–42, May 2013. DOI 10.1109/MC.2013.179.

8. Matt Larsen, Jeremy Meredith, Paul Navrátil, and Hank Childs. Ray-Tracing Within a Data Parallel Framework. In *Proceedings of the IEEE Pacific Visualization Symposium*, Hangzhou, China, April 2015. (to appear).

9. Li-ta Lo, Christopher Sewell, and James Ahrens. PISTON: A portable cross-platform framework for data-parallel visualization operators. pages 11–20. Eurographics Symposium on Parallel Graphics and Visualization, 2012. DOI 10.2312/EGPGV/EGPGV12/011-020.

10. Robert Maynard, Kenneth Moreland, Utkarsh Ayachit, Berk Geveci, and Kwan-Liu Ma. Optimizing threshold for extreme scale analysis. In *Visualization and Data Analysis 2013, Proceedings of SPIE-IS&T Electronic Imaging*, February 2013. DOI 10.1117/12.2007320.

11. J. S. Meredith, S. Ahern, D. Pugmire, and R. Sisneros. EAVL: the extreme-scale analysis and visualization library. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 21–30. The Eurographics Association, 2012. DOI 10.2312/EGPGV/EGPGV12/021-030.

12. Robert Miller, Kenneth Moreland, and Kwan-Liu Ma. Finely-threaded history-based topology computation. In *Eurographics Symposium on Parallel Graphics and Visualization*, 2014. DOI 10.2312/pgv.20141083.

13. Kenneth Moreland, Utkarsh Ayachit, Berk Geveci, and Kwan-Liu Ma. Dax Toolkit: A Proposed Framework for Data Analysis and Visualization at Extreme Scale. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization*, pages 97–104, October 2011. DOI 10.1109/LDAV.2011.6092323.

14. Kenneth Moreland, Berk Geveci, Kwan-Liu Ma, and Robert Maynard. A classification of scientific visualization algorithms for massive threading. In *Proceedings of Ultrascale Visualization Workshop*, November 2013. DOI 10.1145/2535571.2535591.

15. Steven G. Parker et al. OptiX: A general purpose ray tracing engine. *ACM Transactions on Graphics (TOG)*, 29(4):66, 2010. DOI 10.1145/1833349.1778803.

16. James Reinders. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism.* O'Reilly, July 2007. ISBN 978-0-596-51480-8.

17. Christopher Sewell, Jeremy Meredith, Kenneth Moreland, Tom Peterka, Dave DeMarle, Li-Ta Lo, James Ahrens, Robert Maynard, and Berk Geveci. The SDAV software frameworks for visualization and analysis on next-generation multi-core and many-core architectures. In *2012 SC Companion (Proceedings of the Ultrascale Visualization Workshop)*, pages 206–214, November 2012. DOI 10.1109/SC.Companion.2012.36.

18. Rick Stevens, Andrew White, et al. Architectures and technology for extreme scale computing. Technical report, ASCR Scientific Grand Challenges Workshop Series, December 2009. `http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Arch_tech_grand_challenges_report.pdf`.

19. John A. Stratton, Christopher Rodrigues, I-Jui Sung, Li-Wen Chang, Nasser Anssari, Geng Liu, Wen mei W. Hwu, and Nady Obeid. Algorithm and data optimization techniques for scaling to massively threaded systems. *IEEE Computer*, 48(8):26–32, August 2012. DOI 10.1109/MC.2012.194.

20. Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. Embree: A kernel framework for efficient cpu ray tracing. *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 33(4):143, 2014. DOI 10.1145/2601097.2601199.

Received July 8, 2015.

DOI: 10.14529/jsfi150307

A Case for Energy-Efficient Acceleration of Graph Problems using Embedded FPGA-based SoCs

Pradeep Moorthy[1,2], *Nachiket Kapre*[1,3]

Sparse graph problems are notoriously hard to accelerate on conventional platforms due to irregular memory access patterns resulting in underutilization of memory bandwidth. These bottlenecks on traditional x86-based systems mean that sparse graph problems scale very poorly, both in terms of performance and power efficiency. A cluster of embedded SoCs (systems-on-chip) with closely-coupled FPGA accelerators can support distributed memory access with better matched low-power processing. We first conduct preliminary experiments across a range of COTS (commercial off-the-shelf) embedded SoCs to establish promise for energy-efficiency acceleration of sparse problems. We select the Xilinx Zynq SoC with FPGA accelerators to construct a prototype 32-node Beowulf cluster. We develop specialized MPI routines and memory DMA offload engines to support irregular communication efficiently. In this setup, we use the ARM processor as a data marshaller for local DMA traffic as well as remote MPI traffic while the FPGA may be used as a programmable accelerator. Across a set of benchmark graphs, we show that 32-node embedded SoC cluster can exceed the energy efficiency of an Intel E5-2407 by as much as 1.7× at a total graph processing capacity of 91–95 MTEPS for graphs as large as 32 million nodes and edges.

Keywords: energy efficiency, sparse graphs, embedded SoCs, FPGAs.

Introduction

During the pioneering years of HPC, computer architects built systems exclusively from specialized vector hardware; such as the Cray-I [1] and other bespoke machines like the NEC SX-3 and Fujitsu Numerical Wind Tunnel. The early 90s saw x86-based systems rise in popularity due to their low cost, simplicity and standardization of the ISA/floating-point system (Intel 8087 was an early example of IEEE-754 compliant processor hardware). Beowulf clusters of these x86 platforms began as low cost hobbyist alternative to state-of-art HPC systems. Based on the idea of connecting relatively inexpensive COTS computers to solve a particular problem collectively, the first such cluster was developed in 1994 by connecting 16 Intel DX4 processors with 10Mbps Ethernet. This eventually paved way for the creation of the first cluster based supercomputer in 1997, the ASCI Red, which employed 7,246 Intel x86 Pentium Pro processors linked using a custom-interconnect architecture. Peaking the TOP500 list for nearly three years, it set out the foundation for the dominance of x86 cluster systems we see today.

The same era saw the introduction of Reduced Instruction Set Architecture (RISC) based systems in place of Complex Instruction Set Architecture (CISC) machines in the form of PowerPC processors used in the IBM BlueGene. This supercomputer series was launched in 2004 to exploit the low power capabilities of RISC instead of CISC chips by combining multiple PowerPC processors onto each chip. Thus, the usage of multiple low power processors, typically RISC based, in place of a single power hungry "fat" processor was recognized as a way to improve energy efficiency. In lieu of PowerPC hardware, ARM chips have been gaining more interest in the research community since they are fabricated extensively in mobile devices to deliver low power at low cost. The largest ARM-based cluster studied was the Tibidabo cluster [2], which consisted

[1]Nanyang Technological University, Singapore
[2]University of Toronto
[3]nachiket@ieee.org

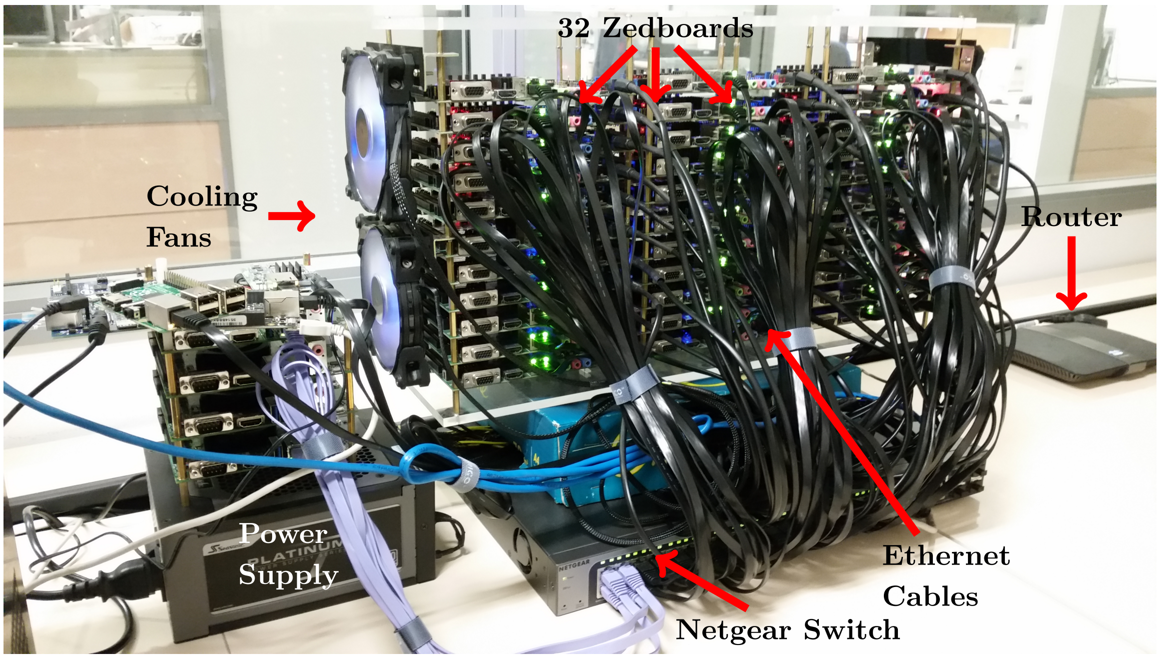


Figure 1. Zedwulf Cluster: 32 Zynq Z7020 SoC boards

of 192 NVIDIA Tegra-2 SoCs, interconnected using 1GbE network. The study concluded the lack of high-bandwidth I/O interfaces such as 10GbE/InfiniBand and the absence of hardware support for interconnection protocols on the Tegra-2's ARM Cortex-A9 processor as the sole limiting factors in adopting the SoC for HPC usage. While the present day performance gap between HPC-grade x86 processors and commercial ARM processors can be as high as an order of magnitude, large graph problems with low spatio-temporal locality can eliminate the performance gap between the two architectures while retaining the energy efficiency advantages. To investigate this claim, we prototype a Beowulf cluster composed of 32 Xilinx FPGA-based Zynq SoC boards, interconnected using a Gigabit Ethernet Switch. We map sparse-graph oriented irregular computations of varying dimensions to stress the memory and network throughputs of the cluster nodes. Fig. 1 shows a photograph of our "Zedwulf" (ZedBoard+Beowulf) cluster.

In this paper, we make the following key contributions:

- **Microbenchmarking of COTS SoCs**: We analyze the memory potential and network characteristics of various embedded SoCs using micro-benchmarking tools.
- **Prototype a 32-node Zynq SoC cluster**: We prototype physically a 32-node Zynq SoC cluster using the Xilinx Zedboard and Microzed platforms.
- **Communication optimization for sparse-graph access on the Zynq cluster**: We develop customized Message Passing Interface (MPI) routines and DMA engines optimized for irregular access exhibited by graph problems.
- **Performance and power evaluation of the Zynq cluster vs an x86 server node**: We benchmark our cluster for a few representative sparse graphs and compare against the Intel E5-2407 CPU.

1. Microbenchmarking COTS SoC Platforms

We first evaluate a range of COTS embedded SoC-based platforms listed in tab. 1 to assess their feasibility for scaling to larger-scale systems. Our characterization experiments focus on a single chip and measure raw compute throughput, memory performance as well as MPI support for these systems.

Table 1. Comparing datasheet specifications and microbenchmarking of various COTS SoCs

	Zedboard	Microzed	Parallella	Intel Galileo 2	Raspberry Pi	Beaglebone Black
Technology	28nm	28nm	28nm	32nm	40nm	45nm
SoC	Xilinx Zynq 7020	Xilinx Zynq 7010	Xilinx Zynq 7010	Intel Quark X1000	Broadcom BCM2835	TI AM3359
Processor	ARMv7, FPGA	ARMv7, FPGA	ARMv7, FPGA, Epiphany III	i586	ARMv6	ARMv7
Clock Freq.	667 MHz CPU 250 MHz FPGA	667 MHz CPU 250 MHz FPGA	667 MHz CPU 250 MHz FPGA	400 MHz	700 MHz	1 GHz
On-chip Memory	32 KB L1 512 KB L2 560 KB FPGA	32 KB L1 512 KB L2 560 KB FPGA	32 KB L1 512 KB L2 240 KB FPGA	16 KB L1	16 KB L1 128 KB L2 -	32 KB L1 256 KB L2
Off-chip Memory	512 MB 32b DDR3-1066	1024 MB 32b DDR3-1066	1024 MB 32b DDR3-1066	256 MB 32b DDR3-800	512 MB 32b DDR2-400	512 MB 16b DDR3-606
DMIPS	1138	1138	1138	237	862	1778
Coremark	1591	1591	1782	526	1314	2457
Network[4]	57 MB/s	59 MB/s	32 MB/s	18 MB/s	10 MB/s	21 MB/s
L1 B/W	7.7 GB/s	7.7 GB/s	7.5 GB/s	2.8 GB/s	2.7 GB/s	7.6 GB/s
L2 B/W	1.4 GB/s	1.4 GB/s	1.4 GB/s	-	1.4 GB/s	3.4 GB/s
DRAM Seq.	654 MB/s	641 MB/s	537 MB/s	270 MB/s	187 MB/s	278 MB/s
DRAM Rnd.	32 MB/s	32 MB/s	28 MB/s	12 MB/s	10 MB/s	11 MB/s
Power	5 Watts	3.6 Watts	7.5 Watts	4 Watts	3.75 Watts	3.25 Watts

Recent academic studies have examined the feasibility of HPC systems based on mobile SoCs [3] for HPC-oriented workloads and investigated the status of networking support in these SoCs. Additionally, there are many contemporary hobbyist clusters built from **Apple TV** [4], **Raspberry Pi** [5], and **Beagleboard xM** [6] that use off-the-shelf devices for delivering proof-of-concept systems with high power efficiency. These studies are insightful but it remains to be seen if pure ARM-based SoCs have future prospects in the cluster computing space.

Our preliminary experiments on the **Intel Galileo 2** platform indicate the Quark SoC would not be competitive at this stage with its under-powered 400 MHz 32b CPU when compared to ARM-based embedded SoC platforms. It reported the lowest DMIPS score of 237 and had poor Ethernet throughput of 10 MB/s (100M Ethernet NIC). Occupying the lower-end of the ARM spectrum, the **Raspberry Pi** reported a 3x higher DMIPS/Coremark score than the Galileo 2. Nevertheless, its relatively slower DDR2 memory limits the overall performance gains. The **Beaglebone Black** further doubles the compute performance to 1778 DMIPS. However, the 16b 400 MHz DDR3 memory barely keeps up with its superior compute capabilities constraining overall performance. Besides, these devices are also limited by 100 Mb/s network links. The Zynq SoC-based platforms (Zedboard, Microzed and Parallella) overcome some of these shortcomings by coupling the Zynq SoC to a 1 Gb/s network link and a respectable 32b DDR3 1066 MHz memory. The **Zedboard** and **Microzed** delivered the highest sequential and random access memory bandwidths. Complemented by the Gigabit Ethernet connectivity, these platforms averaged bi-directional network throughput at a high 60 MB/s. Nonetheless, that corresponds only to a network efficiency of 24%. This behavior is attributed to the slower clock rate of the ARM cores (35% slower ARM CPU relative to the Beaglebone running at 1 GHz). In addition to the Zynq SoC, the **Adapteva Parallella** [7] platform also attaches an Epiphany floating-

[4] Intel MPI Benchmark Suite result for `MPI_Sendrecv` for all systems in 2-node configurations

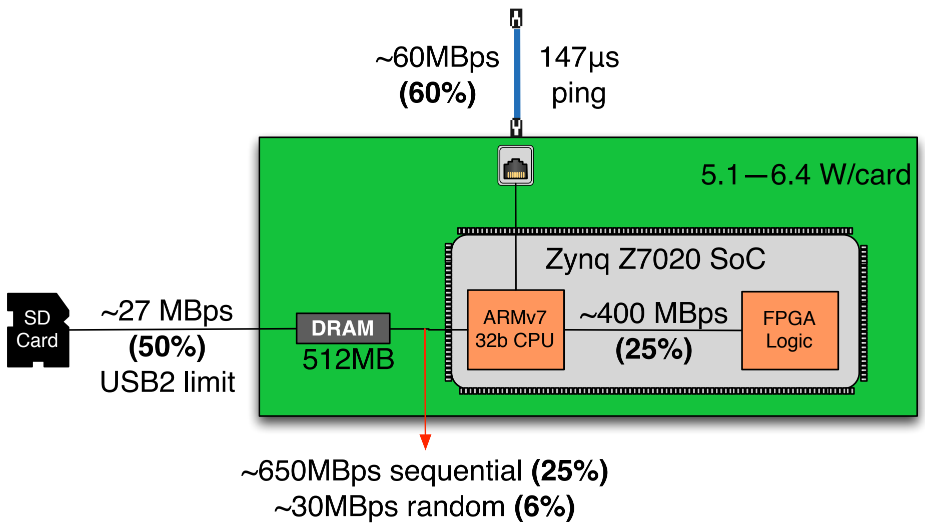


Figure 2. A Zynq node (Zedboard) with peak and achieved bandwidths

point co-processor as a separate chip thereby improving its compute capability substantially. We recorded comparable DMIPS and memory bandwidth scores on the Zedboard/Microzed, but the network throughput saturated at a disappointing 32 MB/s. The high local DRAM and remote MPI throughputs suggest that the Zedboard can become a viable candidate for energy-efficient operation for sparse irregular workloads. It is worth noting that these Zynq platforms are development systems with extraneous supporting logic for audio, video and configurable IOs that can be eliminated in a pure datacenter/HPC-focused design.

2. Zedwulf Organization

The Zedwulf cluster is composed of 32 Zedboards (Rev. D) or 32 Microzed (eval. kit), interconnected using a Netgear GS748T 48-port Gigabit Smart Switch. With a rated switching capacity of 96 Gb/s, the switch can sustain 2 Gb/s duplex bandwidth per 1GbE Ethernet link connecting each Zedboard. We powered the system using a Seasonic Platinum 1KW PSU from the PCIe EPS12 power rail with fuse protection. We stacked the Zedboards on top of each other in three columns with 10/11 boards on each column. We provided air cooling from 2 fans placed on either sides of the stack (4 fans total) as shown in fig. 1. While every Zedboard has a SanDisk Ultra 32 GB SD card attached to host the OS, the master node has an additional Samsung 840 Pro SSD attached to the USB2 port using a SATA-USB adapter. We setup the SSD as the primary secondary storage device for our cluster to hold our large graphs and it offers a convenient lower latency solution for quickly loading and distributing sub-graphs. A single Zynq node with various interface bandwidths is shown in fig. 2. We also built a 32-node Microzed cluster by simply replacing the Zedboard with Microzeds.

The Zynq is a heterogeneous multicore system architecture consisting of a dual-core ARM Cortex-A9 on the Processing System (PS) and a FPGA fabric on the Programmable Logic (PL). Residing on the same chip, the PS and PL are interconnected using AXI on-chip buses. This contrasts to traditional FPGA implementations, whereby the latter is connected to an x86 host using PCIe buses. This approach allows ARM processors to benefit from low-latency links to the FPGA which allow tightly-coupled CPU-based control of FPGA operation.

We configured each Zedboard to run Xillinux-1.3a, an Ubuntu-12.04 based Linux distribution with Xillybus drivers to communicate with the FPGA using an AXI 2.4 GB/s channel. We compiled software libraries such as MPI and other utilities with `gcc-4.6.3` with appropriate optimization flags enabled. We use NFS (Network File System) to synchronize files (graphs) across all 32 nodes. We setup MPI to use Myrinet Open-MX patch to deliver a marginal improvement in network latency. We also choose MPICH over OpenMPI as it provided a 20–30% lower latency and higher bandwidth in our initial stress benchmarks.

3. Communication Optimization

Graph processing is a communication-dominated algorithm that can often be organized as lightweight computations on vertices and message-passing along edges. We map bulk-synchronous parallel (BSP) graph computations of the style used in neural network evaluation, page-rank calculations, and sparse matrix-vector multiplication. We map these evaluations to our cluster by careful optimization of local communication (irregular memory access) and remote communication (MPI access) and compare it against simple x86-based implementations that leverage multi-threading and compiler optimizations.

3.1. MPI Optimization

Partitioning the graph structure to fit across multiple *Processing Elements (PEs)* creates network traffic which connect local vertices to vertices present in other PEs. Unlike local edges, which connect vertices present within the same PE, updating remote edges is typically an order of magnitude slower as the data needs to be transferred from the origin PE to the target PE using the ARM CPUs to handle network packet transfers. Hence, there is an inherent need to reduce the time spent in fulfilling the network operations for maximizing performance gains while using distributed systems. We designed an optimized graph-oriented global scatter technique [8] using the Message Passing Interface (MPI) library.

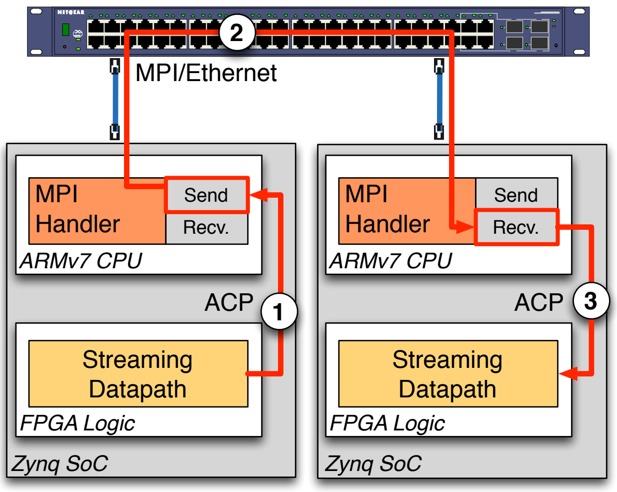


Figure 3. Sequence of steps for synaptic communication along edge of sparse graph. Step ① and Step ③ operate over the ACP links, while Step ② is managed by the MPI library over Ethernet

Our approach leveraged coalesced data transfers between PEs to take advantage of the network bandwidth, rather than being limited by the high network latencies. The high-level packet flow in the system is shown in fig. 3. We used **MPI_type_indexed** API to encode the *send* and *receive* buffer displacements in an MPI friendly manner. We then employed **MPI_Sendrecv** as the building block of our scatter routine. The send-recv operations were scheduled in a periodic fashion to avoid network contention across MPI nodes. This coalesced approach of edge updates offered the speedup of **60**× when compared to performing fine-grained message transfers. We show a simplified code sketch of our MPI optimization in fig. 4.

```
// build MPI data structure from graph
for(j=0:total_proc-1) {
        send_type=MPI_Datatype();
        recv_type=MPI_Datatype();
        MPI_Type_indexed(send_count, send_addr,send_type);
        MPI_Type_indexed(recv_count, recv_addr,recv_type);
        MPI_Type_commit(send_t);
        MPI_Type_commit(recv_t);
}

// Loop over multiple bulk-synchronous steps
for(BSP steps) {
        // Manage local messages

        // Send MPI data between nodes
        for(j=0:total_proc-1) {
                // scheduling to avoid conflicts
                int target = (rank+j)%total_proc
                int source = (total_proc+rank-j)%total_proc;
                MPI_Sendrecv (send_buf, recv_buf, ...);
        }
        MPI_Barrier();

        // Do compute stuff
}
```

Figure 4. Basic MPI Communication Skeleton that shows how the `MPI_Datatype` is built and the mechanism of using `MPI_SendRecv` for sparse communication

First, we translate the sparse graph adjacency lists into MPI-compatible data types that encode the graph structure as a series of addresses and counts for send and receive between all-possible pairs of MPI nodes. This is done for those edges that cross compute node (SoC board) boundaries. We perform a coalesced transfer to one MPI target in a single function call to avoid MPI overheads of finer-grained messages. To achieve this coalesced transfer, we setup the `MPI_Datatype` using `MPI_Type_indexed` to encode a custom sequence of blocks with source and destination positions. We employ the Passive Target Communication paradigm here, using `MPI_Win_lock` and `MPI_Win_unlock` functions for executing Remote Memory Access (RMA) calls. We exploit opportunities for overlapping communication in the system by (1) having simultaneous epoch sessions in progress, ensuring load-balanced scheduling of message transfers in a cyclic fashion, and (2) replacing local `MPI_Put` with simple array-indirection.

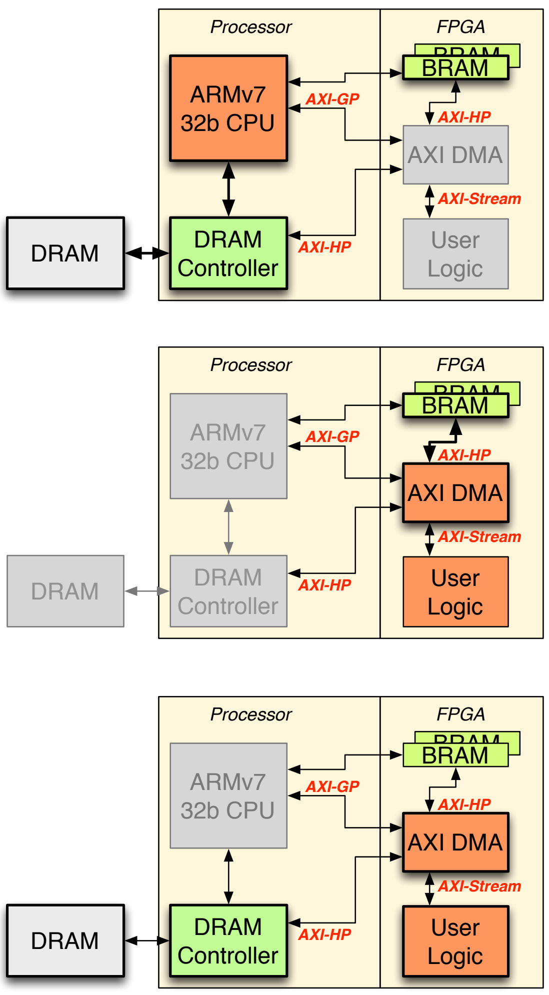


Figure 5. MMU Optimization: Scatter-Gather operation

3.2. Memory Access Optimization

For each vertex in the graph, the graph processor needs to fetch adjacent vertex data from local memory wherever possible. The graph data is conventionally stored in a compressed sparse format (row based or column based), which is a memory storage optimization for sparse graph structures. However, memory access patterns can still result in frequent cache misses under this memory organization scheme.

While the FPGA on the Zedboard has 560KB of on-chip memory, they can barely accommodate 100-1000s of graph vertex and edges. Using the off-chip DRAM memory carelessly would result in poor DRAM bandwidth utilization. Hence, we designed a Memory Management Unit (MMU) for Zedwulf to optimize irregular data transfers. We configure the AXI DMA IP block to use low-level AXI descriptor chains to encode the sparse graph access sequence. With our approach we are able to improve random DRAM access throughput for graph operations by as much as **3–4**×.

We operate the MMU in optimized Scatter-Gather mode [9]. This allows the `AXI_DMA` engine to avoid requiring frequent assistance from the CPU and enables somewhat independent operation. In this mode, instead of programming the internal registers for each DMA transfer, the CPU only needs to construct a one-off linked list of AXI descriptor commands for the complete series of transfers. This can be done once at the start and reused repeatedly for iterative BSP-like graph algorithms. The descriptor chain is stored locally on the FPGA fabric in BlockRAMs and coupled to the `AXI_DMA` engines over an AXI-HP interface. It can even be constructed on-the-fly based on the compressed sparse-row representation of the graph, but we do not explore this at present.

```
XScuGic InterruptController;

struct axi_desc_t {
  u32 next;
  u32 base_addr;
  u32 control;
  u32 status;
};

struct axi_desc_t axi_desc[GRAPH_ACCESSES];

// Initialize graph access pattern as DMA descriptor chain
for (i=0; i<GRAPH_ACCESSES; i++)  {

    // create an entry in linked list
    axi_desc[i].base_addr = base_addr[i];
    axi_desc[i].control = length[i];
    Xil_Out32(BRAM_ADDR + i*ALIGN + NXTDESC, axi_desc[i].next);
    // copy other fields to BRAM
}

// Initialize DMA engine
Xil_Out32(DMAREG_ADDR + MM2S_CURDESC, BRAM_ADDR );
Xil_Out32(DMAREG_ADDR + MM2S_DMASR, 0x0000000);
Xil_Out32(DMAREG_ADDR + MM2S_DMACR, 0x5001);

// Perform DMA on the descriptor chain
Xil_Out32(DMAREG_ADDR + MM2S_TAILDESC, BRAM_ADDR + (GRAPH_ACCESSES-1)*ALIGN);
```

Figure 6. Scatter-Gather-Mode `AXI_DMA` device driver

In fig. 5 we show the three-step configuration flow for the Scatter-Gather DMA mode. In scatter-gather mode we represent the irregular list of accesses as a linked list of `<base_addr>,<length>` tuples stored in local on-chip FPGA BlockRAM. We instruct the DMA engine to get `length` bytes starting from `base_addr` location of the graph representation. Instead of forcing the interrupt after each transfer, we are able to perform a set of back-to-back transfers directly without interrupting the host until after the full sequence has been transferred. This ability to avoid frequent CPU interrupts coupled with FPGA-based storage of AXI descriptor chain provides low-latency turnaround times between consecutive DMA transactions. This is loaded once at the start over AXI-GP ports from the CPU. We represent this in fig. 6 in the `InitializeDescriptors` function. The address of the next descriptor is specified in each descriptor. The head and tail descriptors are provided to the DMA engine and it will process one descriptor after another.

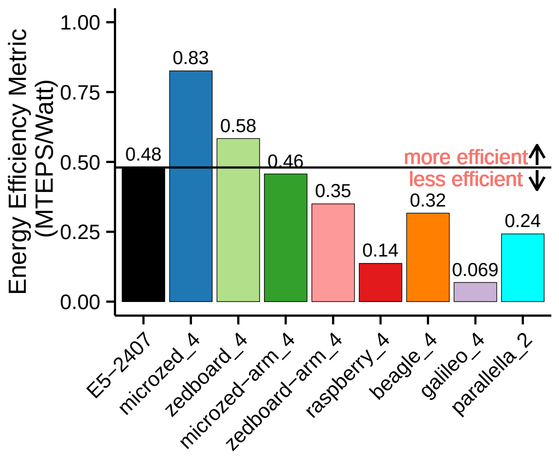


Figure 7. Performance-Power Tradeoffs across embedded SoCs platforms (4-node and 2-node SoCs) and a single x86 node ("-arm_4" versions exclude FPGA and only use ARM). Graph size is 32 million vertices and 32 million edges

4. Results

We analyze the performance and energy-efficiency of various embedded clusters for sparse graph processing. We perform bulk-synchronous evaluation on randomly-generated graphs (Erdos-Renyi [10] technique) with upto 32 million vertices and 32 million edges that can safely fit within the limited memory available on the embedded platform. For the first experiment, we setup 4-node clusters of each unique embedded platform (except Parallella with 2 nodes) and compare it against one x86 node. We scale our setup for the second experiment where we compare the 32-node Zedboard and 32-node Microzed clusters against a single x86 node. The code running on the single x86 node is parallelized using OpenMP pragmas to use all available cores (4 cores for the E5-2407). For completeness, our power measurements include the Ethernet switch and PSU along with the Zynq boards.

In fig. 7 we plot processing efficiency (MTEPS/W) across various embedded and x86 platforms. The Galileo 2 and Raspberry Pi clusters have the lowest performance while demanding high power usage. The Beagle cluster doubles the performance achieved while consuming 10% less power, thereby improving the power efficiency. The 2-node Parallella cluster nearly matches the performance of the 4-node Beaglebone but it needs more power for the extra Epiphany co-processor. The Zedboard and Microzed boards offer the highest energy efficiency when using the FPGA accelerators instead of simply relying on their ARM CPUs. The Microzed stands out with its 30% less power use over the Zedboard as it eschews unnecessary development support (audio, video, IO chips) in favor of a low-cost implementation.

In fig. 8 we show the performance (in MTEPS, millions of traversed edges per second) of the x86 node and the Zynq clusters plotted against their measured power consumption. We are able to marginally exceed the energy efficiency of the x86 node (0.48 MTEPS/W vs. 0.58 MTEPS/W) when using the Zedboard cluster. However, the lower-power and cheaper Microzed-based cluster is able to deliver a 1.7× improvement in energy efficiency (0.83 MTEPS/W) due to its lean design.

This measured 0.83 MTEPS/W energy efficiency figure is within striking distance of the 1.89 MTEPS/W[6] possible in the `SMALL DATA` category of the Green Graph500 list. We look

[6]http://green.graph500.org/lists.php, July 2015 list, University of Luxembourg

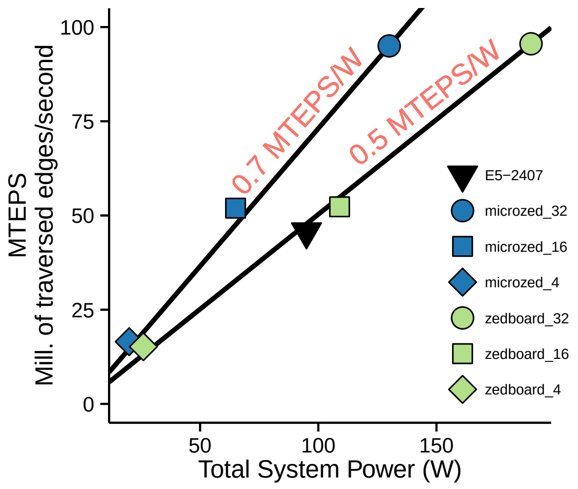


Figure 8. Energy Efficiency of Zynq FPGA cluster against an x86 node (4-node, 16-node and 32-node Zynq setups). Graph size is 32 million vertices and 32 million edges

forward to implementing the Graph500 benchmarks on larger problem sizes with larger cluster of Zynq nodes in the near future that builds upon this work.

5. Conclusions

We show how to use the Zynq SoC with ARMv7 32b CPUs supported by FPGA-accelerators to prototype energy-efficient HPC systems for sparse graph acceleration. For a range of graphs up to 32 million nodes and edges, we are able to deliver a performance of 91–95 MTEPS at an energy-efficiency of 0.58 MTEPS/Watt (32-node Zedboard), and 0.83 MTEPS/Watt (32-node Microzed) which exceeds the x86 efficiency of 0.48 MTEPS/Watt by as much as 1.7×. While the Zynq SoC we evaluated in this study is promising, performance gains were limited by (1) slow 1G Ethernet speeds of 50% peak, (2) limited DRAM capacity per node 512 MB, (3) poor CPU-FPGA link bandwidth of 400 MB/s, and (4) extraneous devices and interfaces for audio/video processing. Upgraded Zynq SoCs optimized for data-center processing that address these concerns can further improve performance and energy efficiency of these systems.

6. Future Work

While the Zynq SoC we evaluated in this study is promising, performance gains were limited by a variety of factors. The slow network transfers that saturate at only 50% of peak 1G Ethernet speed and MPI stack overheads result in a communication time that is roughly 2× worse than network performance of the x86. The limited DRAM capacity of 512 MB per node constrained the size of the largest graphs we could evaluate in this study. The poor CPU-FPGA link bandwidth of 400 MB/s meant that data-transfer time dominated FPGA runtimes. The Zedboard platform chosen in this study contains extraneous devices and interfaces that can be removed for HPC-like scenarios that reduces size, power and cost to be a better candidate for a future study. It may even be prudent to evaluate the smaller Z7010 SoC (cost $56/chip compared to $100/chip for the Z7020) with a smaller FPGA fabric for better balanced design. To improve the programmability of the FPGA design, the use of arrays of soft-processor tiles [11] overlayed on top of the FPGA but fully-customized to particular graph problems would be a promising approach. The Parallella platform with specialized high-performance I/O banks could be used as a superior interconnect alternative to Ethernet for sparse low-latency communication between SoC chips.

References

1. Richard M. Russell. The CRAY-1 Computer System. *Commun. ACM*, 21(1):63–72, January 1978. DOI: 10.1145/359327.359336.
2. Nikola Rajovic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones, and Alex Ramirez. Tibidabo: Making the case for an ARM-based HPC system. *Future Generation Computer Systems*, 2013. DOI: 10.1016/j.future.2013.07.013.
3. Nikola Rajovic, Paul M Carpenter, Isaac Gelado, Nikola Puzovic, Alex Ramirez, and Mateo Valero. Supercomputing with commodity CPUs. In *the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, New York, New York, USA, 2013. ACM Press. DOI: 10.1145/2503210.2503281.
4. Karl Fürlinger, Christof Klausecker, and Dieter Kranzlmüller. The AppleTV-cluster: Towards energy efficient parallel computing on consumer electronic devices. *Whitepaper, Ludwig-Maximilians-Universitat*, 2011. DOI: 10.1007/978-3-642-23447-7_1.
5. Simon J Cox, James T Cox, Richard P Boardman, Steven J Johnston, Mark Scott, and Neil S O'Brien. Iridis-pi: a low-cost, compact demonstration cluster. *Cluster Computing*, 17(2):349–358, June 2013. DOI: 10.1007/s10586-013-0282-7.
6. E Principi, V Colagiacomo, S Squartini, and F Piazza. Low power high-performance computingon the Beagleboard platform. In *Education and Research Conference (EDERC), 2012 5th European DSP*, pages 35–39, 2012. DOI: 10.1109/ederc.2012.6532220.
7. Linley Gwennap. Adapteva: More Flops, Less Watts. *Microprocessor Report*, pages 1–5, June 2011.
8. P. Moorthy and N. Kapre. Zedwulf: Power-performance tradeoffs of a 32-node zynq soc cluster. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 68–75, May 2015. DOI: 10.1109/fccm.2015.37.
9. N. Kapre, Han Jianglei, A. Bean, P. Moorthy, and Siddhartha. Graphmmu: Memory management unit for sparse graph accelerators. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 113–120, May 2015.
10. Paul Erdös and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.
11. N. Kapre. Custom fpga-based soft-processors for sparse graph acceleration. In *Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on*, pages 9–16, July 2015. DOI: 10.1109/asap.2015.7245698.

Received June 3, 2015.