Exascale Computing at the LHC : Narrative

Principle Investigators :
Reneta Barneva, Matthew Jones,
Steven Ko, Salvatore Rappoccio, Lukasz Ziarek

Mentors : Valentin Brimkov, Peter Elmer

May 15, 2013

1 Funding Strategy

This research proposal outlines the necessity to extend the computing capacity of the experiments at the Large Hadron Collider (LHC) in Geneva, Switzerland, to the exascale of high-throughput processing. This is an enormously challenging task, but a necessary one to ensure the long-term success of the LHC experiments.

Exascale computing (in this case, in high throughput) is an enormously growth-oriented area. Even during lean economic times, the US Federal Government is pledging to support this area of research, for instance in the Department of Energy Exascale Computing Initiative [1]. Indeed, the DOE Office of Science quotes :

> The Exascale initiative will be significant and transformative for Department of Energy missions.

The current level of funding for the DOE EIC and related activities is $21M. [2]. This "transformative" strategy is one that is envisioned to continue well into the future. In addition to DOE programs, the National Science Foundation (NSF) has several programs to address the problems of high-throughput exascale computing [3, 4].

In addition to governmental programs such as above, private sector funding sources are also available, such as the Google Faculty Research Awards [5], which has an interest in exascale computing projects also.

We plan to apply for these projects in the coming year 2013-2014.

2 Project Organization

The principle investigators (PIs) of this proposal have a widely-varied and applicable skill set to accomplish the goals of extending LHC computing to the exascale.

- Salvatore Rappoccio has 15 years of experience programming in a high-energy physics environment, as well as other numerical software design for the private sector. He is an expert in critical areas of event reconstruction at CMS which can be optimized for multicore usage.

- Lukasz Ziarek has 9 years of experience in language, compiler, and runtime design targeted at improving multicore performance. He has worked on 5 compilers and 3 Java VMs. He is an expert at speculative and transactional computation focusing on the extraction of parallelism and lightweight concurrency.

- Steven Ko has 10 years of experience in distributed systems. His recent focus has been large-scale data processing in the cloud using MapReduce and other technologies built on top of it. He also has 5 years of experience in large-scale storage and data management in data centers.

- Matthew Jones has more than 12 years experience in scientific software development, with a particular emphasis on parallel programming. As the Associate Director of the Center for Computational Research at the University at Buffalo, he has also been responsible for designing and administering high-performance computing facilites for large-scale scientific computing.

- Reneta Barneva has 29 years of experience as a computer scientist, and is the author of over 50 publications in the subjects of combinatorial image analysis, pattern recognition, and computational geometry.

There are also two senior mentors to the project to offer guidance, insight and technical information.

- Valentin Brimkov has over 20 years of experience as a mathematician, and has extensive expertise in design and analysis of algorithms, combinatorial optimization, and discrete geometry.

- Peter Elmer is the deputy offline software coordinator of the CMS experiment at the LHC. He was responsible for the design and implementation of the data and workflow management system used by CMS, as well as its core event processing software and software development environment. He has made important contributions to the software and computing of several high-energy physics experiments over the past 20 years and is currently focused on planning the computing R&D needed for the next decade in CMS.

In addition, the proposal includes funding for one graduate student from the CSE department at SUNY Buffalo at 100% FTE, and one graduate student from the Physics Department at SUNY Buffalo at 50% FTE. They will be performing the study of this problem under the direction of the PIs, and will also be responsible for deployment and scalability of the solution.

3 Narrative

With the discovery of the Higgs boson by the Large Hadron Collider (LHC) experiments ATLAS and CMS [6, 7], the standard model (SM) of particle physics is now complete. This model unifies the electromagnetic force (carried by the *photon*) with the weak force, responsible for radioactive decay (carried by the *W and Z bosons*). At long last, physicists now understand that via interactions with the Higgs field, the W and Z bosons acquire a mass, but the photon does not. This is referred to as "electroweak symmetry breaking".

A new phase of particle physics has therefore begun. The questions have shifted from the cause of electroweak symmetry breaking, to the study of the Higgs boson and its interactions in detail. To understand the larger picture of the fundamental forces in nature, the past excellence of the LHC experiments must therefore continue unabated in the face of new technical challenges.

One of the major technical challenges that lies ahead is the continuation of the scaling of computational power year by year, known colloquially as "Moore's Law". To set the scale, at the CMS experiment with the LHC collision flux ("luminosity") reaching $7 \times 10^{33}\text{cm}^{-2}\text{s}^{-1}$, the processing time to reconstruct each collision event by CMS was approximately 20 seconds per event. However, as the luminosity is increased, the computational time currently scales quadratically. As the upgraded LHC is expected to deliver $> 12 \times 10^{33}\text{cm}^{-2}\text{s}^{-1}$ in the upcoming run, the processing time per event is expected to reach several minutes per event as shown in Figure 1. Furthermore, in future runs of the LHC in the next 15 years, the luminosity is expected to reach as high as $> 1 \times 10^{34}\text{cm}^{-2}\text{s}^{-1}$, which would correspond (naively) to several hours of computational time per event! Clearly, it is necessary for the computing power to scale in order to compensate for this dramatic increase in CPU time with increasing luminosity.

However, with the expected end of the historic scaling of single-core processing capability [8], it is imperative to utilize a parallel processing strategy in order to maintain the levels of computational speed that are required.

As a starting point, we first note that during 2012, the CPU usage for CMS was very roughly divided up as:

- $\sim$40% event simulation
- $\sim$20% prompt event reconstruction (within 48 hours of data-taking)
- $\sim$40% mixed user analysis applications

Oftentimes, the codes used by CMS (and experimental HEP in general) tend to lack clear numerical "kernels" where optimization efforts can be focused. Given these characteristics they are generally more properly classified as "high throughput computing" (HTC) rather than "high performance computing" (HPC). In terms of their detailed behavior on the CPU many of these codes resemble more general enterprise or "cloud" applications [9, 10].

However, there are several numerical algorithms where parallelization could be exploited more directly. One of these is the so-called "jet clustering" algorithms used at CMS. At CMS, this computation is done very often by individuals rather than centrally, and so is accounted for in the 40% of CPU usage from "mixed user analysis applications" as described above. It is likely, therefore, that improvements observed in jet clustering will primarily benefit this portion of the CPU usage. We now discuss the prospects for utilizing parallelization in jet clustering in detail.

3.1 Jet Clustering

The energetic deposits of particles in detectors need to be clustered to obtain the complete response. This is because the process inherently involves a shower of particles called a "jet". This "jet clustering" is a well-established technique employed at many different particle physics experiments worldwide, and is implemented in a common software framework called `fastjet` [11]. The mathematical problem is analogous to the "K-nearest neighbors algorithm" [12] (kNN). The single-core optimization of jet clustering is outlined in Ref. [13]. In a single core, the computational time scales as $O(N^2)$ or $O(N \ln N$, where N is the number of inputs to the algorithm, which scales linearly with luminosity.

There is existing work and literature on the topic of the parallelization of the kNN algorithm, for instance, in Refs. [14, 15, 16], where improvements $O(100)$ in CPU performance are observed over standard algorithms. Since the proposed use case is very similar to the kNN algorithm, similar improvements to the processing time by parallelization strategies are expected.

We now discuss specific strategies that can be developed in order to apply to this problem.

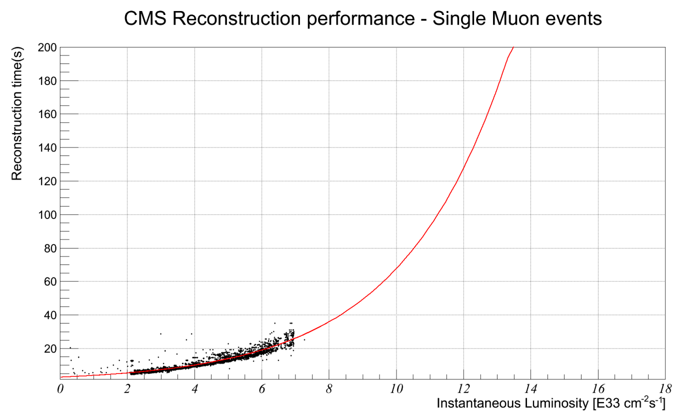

Figure 1: Event processing time versus instantaneous luminosity.

3.2 Full Stack Parallelization

To achieve the necessary improvements in performance required for scalability of jet clustering, we propose to examine parallelization opportunities across the entire software stack, including three specific areas : (1) the use of lightweight concurrency extraction to mask high-latency computations or I/O actions, (2) extraction of parallelization from the computation itself in the form of optimistic speculation and specialized transform, and (3) new methods for distributing the computation to maximize parallelization on each node.

These three techniques are now discussed in turn.

3.2.1 Lightweight Concurrency for Latency Masking

Many mathematical kernels contain opportunities for extracting “micro parallelism,” usually on the order of tens of instructions, from their computational components. Unfortunately, it is very difficult to parallelize this computation profitably as the overhead of thread creation, scheduling, synchronization, and migration outweigh the gains in parallelism. Instead of extracting explicit parallelism from such computations, we proposed to explore methods of lightweight asynchrony to allow for computation to proceed while waiting on high latency I/O operations to complete or the results of other computations. Since the creation of threads and associated schedule and synchronization costs are typically prohibitive, we will explore new threading models that allow for logically-distinct computations to execute within a given construct. The PIs previous research has indicated that such schemes can profitably boost overall performance in the context of ML code [17, 18]. The salient research challenges in applying this strategy are as follows: 1) identifying what computation can be executed safely during high latency operations at compile time, 2) providing a lightweight threading runtime and programming model in the context of an imperative language, 3) specializing the approach to numeric kernels, and 4) building support for computation in a distributed setting.

3.2.2 Speculative Computation

In addition to exploring explicit parallelization of the numeric kernels in jet clustering, we propose to explore extraction of parallelism via speculative computation. At is core, speculative computation breaks apart sequential or parallel tasks into smaller tasks to be run in parallel. Once the speculation has completed, the runtime system validates the computation. If the computation is incorrect (*i.e.* a "datarace" is detected, the computation cannot be serialized, *etc.*), the incorrect computation is re-executed in a non-speculative manner. If the rate of mis-speculation is low, such techniques can be leveraged to extract additional parallelism. There have been many different proposals, including large efforts on transactional memory [], lock elision [], thread level speculation and speculative multithreading [], for integrating speculative computation into programming languages and their associated runtimes []. The PIs have extensive experience with transactional memory [19], lightweight rollback methods [20], leveraging memoization to reduce re-computation costs [21, 22], and deterministic speculation [23]. We propose to explore a specialized speculation framework leveraging different speculation strategies, including speculation extracted by the programmer via programming language primitives, library level speculation, and compiler extracted speculation. The salient research challenges in applying this strategy are as follows: 1) identification the appropriate speculation model and discovering speculation points at compile time, 2) providing a speculative runtime specialized for jet clustering and capable of realizing user, library, and compiler injected speculation, and 3) exploring new and specialized lightweight validation and re-execution mechanisms, including validation across multiple speculation strategies.

3.2.3 Smart Distribution

In order to increase parallelism, we will explore the use of the MapReduce execution framework [24, 25]. MapReduce is a runtime system recently developed for large-scale parallel data processing. It enables programmers to easily deploy their applications on a cluster of machines. Programmers only need to write two functions, Map and Reduce, and submit these two functions as a job to the system. Then the MapReduce framework takes care of all the aspects of the execution of the job. For example, the framework packages and distributes the two functions over the cluster so that the whole cluster can be utilized to execute the job; it also takes care of fault-tolerance by monitoring the cluster during the execution of the job and redistributes the job if some machine fails.

Due to this simplicity and power, it is quickly gaining popularity in industry for large-scale data processing. Unfortunately, scientific computing has not explored the use of MapReduce in depth, so it is unclear what kinds of scientific computing applications can be adapted to benefit from the power of MapReduce. We plan to explore this question in the context of our jet clustering. Previous research has explored implementing kNN with MapReduce [26, 27]; however, these approaches all make various assumptions based on their scenarios, thus cannot be directly adapted to our context. Hence, we will investigate how to efficiently implement our jet clustring using MapReduce.

3.3 Summary

In summary, the problem of expanding LHC computing to the exascale is a difficult, but tractable one. This proposal investigates the possibility of applying cutting-edge computer science research to the real-world application of LHC data processing, specifically by reducing the computational time for k-nearest-neighbor-like numerical kernels via parallelization. The investigators of this proposal have extensive experience in the various aspects of the problem, and the synergistic application of this experience is expected to attain considerable improvements in this avenue.

References

[1] US Department of Energy Office of Science. The Opportunities and Challenges of Exascale Computing. http://science.energy.gov/~/media/ascr/ascac/pdf/reports/Exascale_subcommittee_report.pdf, 2010.

[2] US Department of Energy Office of Science. Congressional Budget for ASCR. http://science.energy.gov/~/media/budget/pdf/sc-budget-request-to-congress/fy-2013/Cong_Budget_2013_ASCR.pdf, 2013.

[3] National Science Foundation. Core Techniques and Technologies for Advancing Big Data Science & Engineering (BIGDATA). http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504767, 2013.

[4] National Science Foundation. High Performance System Acquisition: Building a More Inclusive Computing Environment for Science and Engineering. http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503148, 2013.

[5] Google. Faculty Research Awards. http://research.google.com/university/relations/research_awards.html, 2013.

[6] Serguei Chatrchyan et al. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys. Lett. B*, 2012.

[7] Georges Aad et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett. B*, 2012.

[8] Samuel H. Fuller and Lynette I. Millett. *The Future of Computing Performance: Game Over or Next Level?* The National Academies Press, 2011. Committee on Sustaining Growth in Computing Performance; National Research Council.

[9] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a

study of emerging scale-out workloads on modern hardware. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, 2012.

[10] P. Calafiura, S. Eranian, D. Levinthal, S. Kama, and R.A. Vitillo. GOoDA: The generic optimization data analyzer. *J.Phys.Conf.Ser.*, 396:052072, 2012.

[11] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. FastJet User Manual. *Eur.Phys.J.*, C72:1896, 2012.

[12] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[13] Matteo Cacciari and Gavin P. Salam. Dispelling the N^3 myth for the k_t jet-finder. *Phys.Lett.*, B641:57–61, 2006.

[14] Vincent Garcia, Eric Debreuve, Frank Nielsen, and Michel Barlaud. k-nearest neighbor search: fast GPU-based implementations and application to high-dimensional feature matching. In *IEEE International Conference on Image Processing (ICIP)*, Hong Kong, China, September 2010.

[15] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using gpu. In *CVPR Workshop on Computer Vision on GPU*, Anchorage, Alaska, USA, June 2008.

[16] Vincent Garcia. *Suivi d'objets d'intrt dans une squence d'images : des points saillants aux mesures statistiques*. PhD thesis, Universit de Nice - Sophia Antipolis, Sophia Antipolis, France, December 2008.

[17] Lukasz Ziarek, KC Sivaramakrishnan, and Suresh Jagannathan. Composable asynchronous events. In *ACM SIGPLAN Notices*, volume 46, pages 628–639. ACM, 2011.

[18] KC Sivaramakrishnan, Lukasz Ziarek, Raghavendra Prasad, and Suresh Jagannathan. Lightweight asynchrony using parasitic threads. In *Proceedings of the 5th ACM SIGPLAN workshop on Declarative aspects of multicore programming*, pages 63–72. ACM, 2010.

[19] Lukasz Ziarek, Adam Welc, Ali-Reza Adl-Tabatabai, Vijay Menon, Tatiana Shpeisman, and Suresh Jagannathan. A uniform transactional execution environment for java. *ECOOP 2008–Object-Oriented Programming*, pages 129–154, 2008.

[20] Lukasz Ziarek and Suresh Jagannathan. Lightweight checkpointing for concurrent ml. *Journal of Functional Programming*, 20(02):137–173, 2010.

[21] Lukasz Ziarek and Suresh Jagannathan. Memoizing multi-threaded transactions. *Workshop on Declarative Aspects of Multicore Programming*, 2008.

[22] Lukasz Ziarek, KC Sivaramakrishnan, and Suresh Jagannathan. Partial memoization of concurrency and communication. *ACM Sigplan Notices*, 44(9):161–172, 2009.

[23] Lukasz Ziarek, Siddharth Tiwary, and Suresh Jagannathan. Isolating determinism in multi-threaded programs. *Runtime Verification*, pages 63–77, 2012.

[24] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[25] Hadoop MapReduce. `http://hadoop.apache.org/mapreduce`.

[26] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proc. VLDB Endow.*, 5(10):1016–1027, June 2012.

[27] Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 38–49, New York, NY, USA, 2012. ACM.