# Exascale Computing at the LHC : Narrative

Peter Elmer, Matthew Jones, Steven Ko,
Salvatore Rappoccio, Lukasz Ziarek

May 14, 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.

# 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.

- 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.

- Matthew Jones ....

## 2.1 Introduction

With the discovery of a new boson with mass around $m = 125$ GeV [6, 7] (henceforth referred to as the $H$), a new phase of particle physics has begun. The questions have shifted from the cause of the breaking of the electroweak symmetry, to the nature of that symmetry breaking. Two major questions arise. The first is the exact nature of the particle responsible for the electroweak symmetry breaking. The second is how a particle with a relatively low mass around $m = 125$ GeV can be responsible for electroweak symmetry breaking without extremely large fine tuning in nature, canceling the large radiative corrections to its mass.

With 25 $\text{fb}^{-1}$ of 7 and 8 TeVdata delivered by the LHC in 2011-2012, and instantaneous luminosities reaching $7 \times 10^{33}\text{cm}^{-2}\text{s}^{-1}$, the processing time to reconstruct each event collected by CMS was approximately 20 seconds per event. However, as the instantaneous 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 instantaneous 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 of LHC data in the immediate future in experiments such as CMS. As a starting point we first note that during 2012, the CPU usage for offline computing activities 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 accounted 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.

## 2.2 Jet Clustering:

The energetic deposits of charged and neutral hadrons in the hadronic calorimeter, as well as the deposits of electrons and photons in the electromagnetic calorimeter and the deposits of charged hadrons described above, (which are combined into a single "particle flow candidate" at CMS) need to be clustered to obtain the complete response. This is because the process inherently involves a "shower" of particles that spreads in both the lateral and radial directions, 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 single-core optimization of the mathematical implementation of the nearest-neighbor (NN) algorithm chosen is outlined in Ref. [12], and depends on the number of "candidates" that are input to the jet clustering algorithm. The single-core optimization yields $O(N^2)$ or $O(N \ln N$ operational times. This is analogous to the "K-nearest neighbors algorithm" [13] (kNN).

Even though the single-core computational strategy is sufficient for many applications, there are two key components of jet clustering that are still inefficient for future LHC data processing, which can be solved with appropriate development of a parallelization strategy. The first is the parallelization of the NN algorithm itself to make optimum use of the more advanced vectorization capabilities of modern multicore CPUs. The second is to use a "divide and conquer" strategy to reconstruct a single collision event in several disjoint sections of the CMS detector in parallel.

There is existing work and literature on the topic of the parallelization of the kNN algorithm, for instance, in Refs. [14, 15, 16], where improve-

ments $O(100)$ 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. Furthermore, a critical piece of information in the jet algorithm is the "area" of each individual jet (in the NN-related metric). Currently, the procedure to estimate this area is to add a large number of infinitesimally small candidates called "ghosts" that are uniformly distributed over the total area. These are fed into the jet clustering algorithm, and the positions of these ghosts are used to estimate the size. The resolution of this area is inversely related to the number of ghosts per unit area, so in practice many ghosts ($O(1000)$) are added in the neighborhoods of the jets. This severely limits the computational speed with which these can be processed. Judicious usage of parallelization of the algorithm may be able to solve this problem, and drastically reduce the computational time needed to compute the area.

In addition to the usage of GPU and other parallelization strategies to estimate the jet area, it is also possible to parallelize the computation by dividing the event into disjoint sections, and computing the jet clustering in parallel over multiple cores. This optimization is factorized from the previous, in that it can be done with current single-core computational algorithms, but simply divides the problem into several smaller ones. There are both computational and physics-related challenges that need to be overcome in this strategy, and using a synergistic approach is absolutely critical.

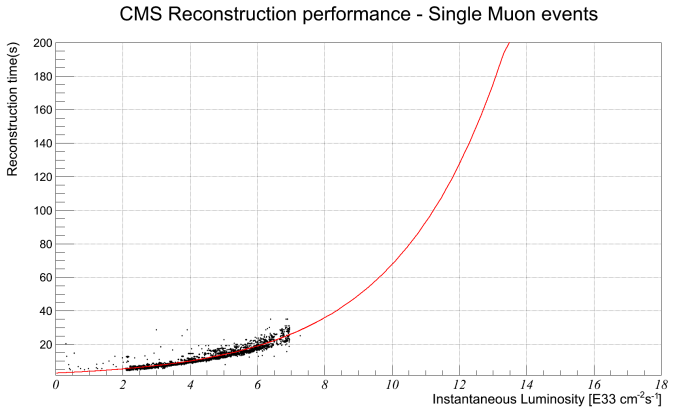


Figure 1: Event processing time versus instantaneous luminosity.

# 3 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 the use of lightweight concurrency extraction to mask high latency computations or I/O actions, extraction of parallelization from the computation itself in the form of optimistic speculation and specialized transform, and new methods for distributing the computation to maximize parallelization on each node.

## 3.1 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 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 spec-

ulation 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.

### 3.3 Smart Distribution

# References


[1] US Department of Energy Office of Science, (2010), The Opportunities and Challenges of Exascale Computing.

[2] US Department of Energy Office of Science, (2013), ASCR Budget.

[3] National Science Foundation, (2013), Core Techniques and Technologies for Advancing Big Data Science & Engineering (BIGDATA).

[4] National Science Foundation, (2013), High Performance System Acquisition: Building a More Inclusive Computing Environment for Science and Engineering.

[5] Google, (2013), Faculty Research Awards.

[6] CMS Collaboration, S. Chatrchyan et al., Phys. Lett. B (2012), 1207.7235.

[7] ATLAS Collaboration, G. Aad et al., Phys. Lett. B (2012), 1207.7214.

[8] S.H. Fuller and E.C.o.S.G.i.C.P.N.R.C. Lynette I. Millett, The Future of Computing Performance: Game Over or Next Level? (The National Academies Press, 2011).

[9] M. Ferdman et al., Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '12, 2012.

[10] P. Calafiura et al., J.Phys.Conf.Ser. 396 (2012) 052072.

[11] M. Cacciari, G.P. Salam and G. Soyez, Eur.Phys.J. C72 (2012) 1896, 1111.6097.

[12] M. Cacciari and G.P. Salam, Phys.Lett. B641 (2006) 57, hep-ph/0512210.

[13] T. Cover and P. Hart, Information Theory, IEEE Transactions on 13 (1967) 21.

[14] V. Garcia et al., IEEE International Conference on Image Processing (ICIP), Hong Kong, China, 2010.

[15] V. Garcia, E. Debreuve and M. Barlaud, CVPR Workshop on Computer Vision on GPU, Anchorage, Alaska, USA, 2008.

[16] V. 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, 2008.

[17] L. Ziarek, K. Sivaramakrishnan and S. Jagannathan, ACM SIGPLAN Notices Vol. 46, pp. 628–639, ACM, 2011.

[18] K. Sivaramakrishnan et al., Proceedings of the 5th ACM SIGPLAN workshop on Declarative aspects of multicore programming, pp. 63–72, ACM, 2010.

[19] L. Ziarek et al., ECOOP 2008–Object-Oriented Programming (2008) 129.

[20] L. Ziarek and S. Jagannathan, Journal of Functional Programming 20 (2010) 137.

[21] L. Ziarek and S. Jagannathan, Workshop on Declarative Aspects of Multicore Programming (2008).

[22] L. Ziarek, K. Sivaramakrishnan and S. Jagannathan, ACM Sigplan Notices 44 (2009) 161.

[23] L. Ziarek, S. Tiwary and S. Jagannathan, Runtime Verification (2012) 63.