Exploiting new CPU architectures in the SuperB software framework

M Corvo, F Bianchi, V Ciaschini, D Delprete, A Di Simone, G Donvito, A Fella, F Giacomini, A Gianoli, S Longo, S Luitz, E Luppi, M Manzali, S Pardi, A Perez, M Rama, G Russo, B Santeramo, R Stroili and L Tomassetti

1CNRS, Orsay, France
2INFN and Università di Padova, Padova, Italy
3INFN and Università di Torino, Torino, Italy
4INFN, Bologna, Italy
5INFN and Università di Napoli, Napoli, Italy
6INFN and Università di Roma Tor Vergata, Tor Vergata, Italy
7INFN and Università di Bari, Bari, Italy
8INFN and Università di Ferrara, Ferrara, Italy
9SLAC, Menlo Park, USA
10INFN and università di Pisa, Pisa, Italy
11LNF, Frascati, Italy

E-mail: corvo@pd.infn.it

Abstract. The SuperB asymmetric-energy $e^+e^-$ collider and detector to be built at the newly founded Nicola Cabibbo Lab will provide a uniquely sensitive probe of New Physics in the flavour sector of the Standard Model. Studying minute effects in the heavy quark and heavy lepton sectors requires a data sample of $75ab^{-1}$ and a luminosity target of $10^{36}cm^{-2}s^{-1}$.

These parameters require a substantial growth in computing requirements and performances. The SuperB collaboration is thus investigating the advantages of new CPU architectures (multi and many cores) and how to exploit their capability of task parallelization in the framework for simulation and analysis software. In this work we present the underlying architecture which we intend to use and some preliminary performance results of the first framework prototype.

1. Introduction

SuperB [1] is an international collaboration whose effort is aimed at the construction of a very high luminosity asymmetric $e^+e^-$ flavour factory. The collider and the detector will be built at the Cabibbo Lab [2] near Rome. A particular technique, developed by the Frascati INFN laboratory [3], allows to achieve a luminosity of $L = 10^{36}cm^{-2}s^{-1}$, that is two orders of magnitude bigger than similar colliders, like those of the Belle [4] and BaBar [5] experiments. The physics studies achievable by such a machine will provide a uniquely important source of information about the details of the new physics discovered at hadron colliders in the coming decade. Very rare events in the decay of elementary particles [6] provide a window to the very high energies that occurred at early times in the development of our Universe.
As in other HEP experiments, the computational effort required to produce simulated data and analyze real events is strategic [7] and the most modern CPU architectures must be exploited to optimize the usage of resources and minimize electric power consumption.

In this view the SuperB collaboration is investigating these architectures and how to write an analysis framework which is able to fully exploit them. In this work we present the underlying framework architecture which we intend to use, the results of the analysis of the performance of the current SuperB framework and the potential speed-up we could gain exploiting parallelism.

2. Measurements on legacy software
The first step in the design of a parallel framework prototype for SuperB consisted in taking measurements on some typical software code. To perform this task we have employed the SuperB Fast Simulation [8] (FastSim in the following), a software developed to study both the detector optimization and the physics in a super-flavour factory [9] like SuperB.

The FastSim software consists of several hundreds of modules, each specialized in tasks like event generation, track reconstruction, filtering and so on. Modules and their execution are managed by the BaBar framework, which enables users to define their analysis setup. By means of configuration files, a user can instruct the BaBar Framework to create lists of modules, set their parameters and then define the execution sequence of the analysis. For our measurements we have employed a typical use-case, with a full analysis chain: from event generation to the detector simulation and finally the track reconstruction and event selection.

At first we focused our attention on the knowledge of the parallelism at the module level, to understand how the execution pattern can be moved to a parallel environment. We have studied the dependencies of each FastSim module, to define the requirements for their execution, from which we have built the dependencies graph. The result of this analysis was quite interesting: as expected, we have found a strongly interconnected graph with several nodes that can run in parallel. Table 1 shows a summary of the results found for a typical simulation example: the graph has only 10 levels, with an average rank - defined as the number of nodes at the same depth - equal to 12, so our use case is intrinsically parallel.

| Num of modules | Graph Depth | Min Rank | Max Rank | Avg Rank |
|----------------|-------------|----------|----------|----------|
| 127            | 10          | 1        | 54       | 12       |

Even if the graph, with 127 nodes on 10 levels, is too complex to be shown in this paper, some examples can be useful to understand the structure found. Figure 1 shows a snapshot of the dependencies graph, where several dependent nodes are present: each node on a level requires something produced in a previous level of the graph to start the execution. Alongside to this type of subgraph we have found other subgraphs made by nodes without inter-dependencies as in Figure 2.

Close the completion of the analysis process it is also possible to find reduction subgraphs: sets of independent nodes converging in a single node in the next graph level.

3. Preliminary measurements
Having understood and consequently resolved the dependencies among modules, we moved to the measurements of theoretical performances that a parallel framework could produce. The
The basic idea is that two independent modules, that is modules whose algorithms do not require data from each other, can run concurrently thus reducing the amount of time spent in performing their operations.

We started this task developing a simple scheduling algorithm able to produce execution sequences of modules based on their dependencies. Then we run a simulation to calculate the computing time of the execution of FastSim, with parallel execution of its modules. We also run this simulation putting a constraint on the number of available cores, to determine whether we would be able to fill all resources and to see if in fact our upper limit in the number of executed modules is close to the average rank calculated.

With this software we have calculated an expected speed-up as a function of the available cores in a system [Figure 3]: the use case considered in our analysis exposes a good linear speed-up until 16 nodes, then it starts to flatten (from 32 to 128 cores the speed-up remains around 13.4). The last step was studying the thread distribution over time as a function of the available cores [Figure 4] that exposes a good resources utilization.

The specific use-case employed in this analysis, representative of a typical flavour factory computing job, has proved to be a good candidate for parallelization at module level, even if the case exposes some limits on the maximum number of usable cores.

4. Framework architecture

Modules operate on data which are generally encapsulated into a complex structure called event. Modules communicate among them through messages while putting the results of their computation inside the event. When a module generates a new physics object, e.g. a vector of tracks or particle candidates, it is called a producer. In general, to produce something, a module needs some input, possibly taken from the event itself, which is processed by its algorithms whose output is registered back inside the event. This scenario suggests a common feature of all modules, that is the existence of required input and provided output.

Having identified each module by the data it requires or produces, it is possible to define an execution schedule based on dependencies. This approach leads naturally to the definition of execution graphs, where each node represents a module and each edge a data product. In this structure the path form the root to each node is the sequence of products which are required to start the execution of that module. We have investigated several system libraries for thread management to understand which product most closely matches our requirements. Currently
our choice is Intel Threading Building Block [10] for the characteristics briefly described in the following. Intel TBB has a native support for graphs (see Figure 5), allowing a near 1 to 1 mapping of the graph we produce from module dependencies. Intel TBB manages graphs with a message passing like protocol: this approach easily fits the requires/provides node design where, in particular, we can also use directly the event as the passing message. This final choice not only simplifies the management of node scheduling but also allows us to design a framework able to parallelize the analysis also at the event level. Should the system have free cores, as the concurrently running modules were unable to fill all the resources, to perform the same analysis on several events should only require to inject the corresponding event object in the root node of the graph. Another potential aspect, which is not covered in this paper, is that Intel TBB should allow us also to parallelize algorithms inside each module.
5. Prototype
The framework prototype we are developing is based on Intel TBB as a proof of concept for the SuperB offline analysis. Our first approach is to modify the existing BaBar framework inserting a graph object at the configuration step. Having resolved all dependencies among modules, we can build the graph when modules are initialized. The requires/provides schema previously exposed is used to define edges which lead to the correct sequence of modules to be finally scheduled. This scheduling is done using the Intel TBB \texttt{flow::graph} scheduler.

Our job is currently focused also on the migration of analysis modules from the legacy serial environment to our prototype, in order to have an homogeneous system where to perform some real measurements.

6. Conclusions
In the present paper we showed that exploiting the most advanced CPU architectures, we can benefit both in terms of resources optimization and of power consumption. The preliminary simulations demonstrate that the current SuperB framework can be parallelized at the module level, improving the performances and execution time.

The analysis of the SuperB software represents also a good opportunity to find critical behaviours and improve code quality. Furthermore the framework architecture based on a mechanism of requirements and provides of modules, seems the most promising in terms of easiness of implementation, as it adapts very well to the software tools we decided to adopt. In the next months we will pursue the adaptation of the current code to the new architecture in order to make some complete measurements of the speed-up gained and collect more statistics of time consumption with different analysis and production chains.

References
[1] The SuperB Collaboration, SuperB Progress Report, Detector
http://arxiv.org/abs/1007.4241v1
[2] http://www.cabibbolab.it/
[3] M. Zobov et al. 2010 Phys. Rev. Lett. 104
[4] The Belle Collaboration 2002 Nucl. Instrum. Meth. A 479 117
[5] The BaBar Collaboration 2002 Nucl. Instrum. Meth. A 479 1-116
[6] Rakitin A 2011 Rare B decays potential at SuperB Proceedings of Science HQL(2010)036
[7] F Bianchi et al 2011 J. Phys.: Conf. Series 331
[8] R Andreassen et al 2011 J. Phys.: Conf. Series 331 32-38
[9] T Browder et al 2009 Reviews of Modern Physics 81
[10] http://threadingbuildingblocks.org/