Thoughts on possible R&D projects for SuperB computing

Peter Elmer
Princeton University
SuperB Computing Workshop, Frascati
16 December, 2008

Overview

- The time scale of SuperB is such that there are probably 2-3 years in which some amount of “R&D” work could be done, in parallel to TDR work.
- In this presentation I'll cover a few things that would be reasonable proposals for such work, assuming:
  - That some amount of dedicated effort is available, separate from effort for the TDR, o(a few FTE?) say
  - That completely “blue skies” R&D is not what one is really looking for, but rather R&D that leads concretely to choices that could be “closed out” as concrete implementations/choices/design-decisions by 3-4 years before data-taking when a post-TDR, “ramp-up of software development” period starts

Overview 2

- There are three areas that are probably interesting (and they are somewhat related):
  - General code performance issues
  - Multicore CPU issues
  - I/O and storage issues
- I'll go through these in turn in the next slides

Code Performance

- One unfortunate thing that was true about the early BaBar software development period was that performance was not emphasised. There were (not unreasonable) arguments for that choice:
  - C++/OO-design were new to many collaborators, even fewer had any experience with performance issues in C++/OO software
  - Time was extremely short to complete even the basics before data taking started
  - Good tools were lacking, multiple platforms/compilers were used
- Most of the work was done late (in 1999-2001) or even later (I understand some amount of effort went into this in the past year or two) and only sporadically outside of that.

Code Performance

- None of the special conditions from the previous slide apply in 2008. Any new project (such as SuperB) should weave a *systematic* emphasis on performance into its software development model and train developers from the start to understand the issues. As an R&D project, this could consist of:
  - A review, with significant overlap with what Dave was just describing, of the performance of the BaBar/SuperB code, leading to adiabatic performance improvements
  - Complete reimplementations of some example pieces with significant design/performance issues with a strong/primary emphasis on performance
  - Most important: documentation/dissemination of experience in order to shape a SuperB SW development model including code performance at a more fundamental level than BaBar achieved.

Multicore CPU's

- For most of the BaBar era, subsequent generations of CPU's were more performant predominantly due to increases in clock frequency
- Due to power issues, the trend since a couple of years has now shifted towards subsequent generations integrating more CPU “cores” on a single die, without increasing the clock frequency significantly
- HEP applications are basically single threaded, for good reasons as many people writing code do not understand the issues with multithreaded programming, hence they can only exploit one core at a time.
- For small numbers of cores, our HEP applications can simply be run as if each core were a separate CPU, providing enough memory/core is provided.
- This naive approach is likely to have some issues in the coming years assuming industry continues this trend.

Multicore CPU's – Memory

- The simple method of one job/core implies that the physical memory needed now needs to scale with each subsequent generation of CPU:
  - Previously one bought a new CPU (twice as fast) and could buy the same memory as for the previous CPU
  - Now one buys a new CPU (with twice the cores) and needs to buy twice as much memory as for the previous CPU
- This has both purchase cost and power implications. Moreover it is extremely wasteful: much of what is actually in the physical memory is redundant between the multiple jobs running on that machine. There are simply multiple copies of the same thing.

Multicore CPU's – Cache Memory

- The main memory is not the only memory relevant to understand the application performance. Much of the performance of CPU's is due to the fact that a memory hierarchy is used, with much faster (and smaller) cache memory located “nearer” to the CPU than the main memory.
- The naive model of one job/core assumes that the size of the cache memory will scale with the number of cores. It makes the worst use of the cache memory possible in that running the entirely application “vertically” on the cache available reduces the chances of cache hits. The result could be that the performance/throughput of the applications doesn't even scale with the number of cores.

Multicore CPU's – I/O

- The naive model of one job/core also leads to an increase in the number of (more or less) random writers/readers hitting the local disk and the remote storage incoherently.
- Previously (with frequency scaling) the number of jobs needed went down with each successive CPU generation (and individual applications could be tuned for best performance), the naive model leads to the number of (incoherent) readers/writers scaling with the total CPU throughput available.
- This may not play well the I/O and storage issues (see later slide).

Multicore CPU's - R&D

- There is some R&D effort for LHC on multicore CPU's to try to understand what to do, given the trends. SuperB can certainly learn some number of things from that effort and apply them.
- A large component (not all, however) of that effort is focused on how to retrofit various solutions on frameworks and code-bases that need to be used “in anger” in the next years. Most of them probably won't scale well to (for example) 128 or 256-core CPU's.
- Moving to multi-threaded applications is one solution, but exposing that to everyone developing SuperB software is likely to be a bad choice.
- Unlike the LHC experiments, SuperB has the time to do a bit of R&D (e.g. in terms of Framework development) to support multicore CPU's as part of the design, and without requiring “advanced” programming knowledge from everyone developing code.

I/O and storage

- The BaBar (CM2) computing model has three big choices related to I/O and storage:
  - Data reduction (with deep copies of events) in the form of skims to "localize" data for access and transport to other sites. Limited use is (or at least was) made of "pointer skims".
    - This implies both the famous factor of 4-5 for total storage costs and the organization of periodic CPU-intensive skim productions (with adminstrative effort to understand what is actually used)
  - ROOT TTrees are used to facilitate access to particular event data products as opposed to a "one record per event" structure
    - This implies some amount of tuning to make sure that the read access pattern is efficient and doesn't overwhelm the I/O capacity of the storage

I/O and storage

- All would be fine if nothing changed and disks and I/O scaled together with their version of Moore's law. (And CPU's, see multicore notes on I/O earlier.)
- As has been pointed out in numerous places, the I/O capacity of disks doesn't scale with their storage capacity and one can easily wind up purchasing more disk than needed simply to meet the I/O needs.
- There is also the wildcard of SSD's and their cost/availability on the time scale of SuperB. (Aside: “disk will completely replace tape” has been with us for a long time, but it hasn't happened. I'll venture a guess that “SSD's will completely replace disk (or tape)” won't happen trivially, either.
- I personally don't know exactly what will happen or what could/should be done, but as getting this wrong can have big effects on cost, an R&D project to understand exactly what SuperB would need if it scales naively from BaBar (and what alternatives are available) probably makes sense.

Conclusions

- SuperB has a lot going for it: the BaBar software development model and computing model (in the end) largely worked and a significant legacy code-base exists which can be evolved or used as a concrete reference to support the SuperB software/computing.
- Some significant issues may arise for SuperB computing/software however from the evolution of the hardware available, making naive Moore's Law extrapolations for the SuperB computing cost difficult (or worse, wrong).
- Some R&D projects in the next 2-3 years are probably appropriate regarding:
  - General code performance
  - “Native” support for multi-core CPU's
  - I/O and storage issues resulting from computing model persistence and data reduction choices