

Position Paper: Efficient HPC software development Mike Kumbera (LLNL)

The software stages in this position paper were inspired by a blog post titled “The Architectural Evolution of DoD Combat Systems”¹. While reading their analysis I was amazed by the similarities to what happens to Software Development at LLNL. I feel the various DOD models closely mimic the stages we have between a Research and a Production Science code.

Almost all our software starts out as an “ad hoc” architecture. This Research Code is developed is highly stove-piped, course-grained and exhibit little or no shared capabilities with any other tools or libraries in use at LLNL. This is partly done because the code author starts an application as a simple little test code to flesh out an idea. There is little (no?) thought put into writing the new software in a way that would be easy to test or interface to an existing applications or tools. The resulting code from the ad-hoc development is typically tightly coupled physics packages that typically share information via copious use of global scoped data. The user base is just the Developer of the Application.

Once the development effort matures to the point of showing interesting results then the software is migrated to a “Modular” architecture. While still consider a research code there is more external interest in the capabilities of the application. The application’s results are now compared against the Production codes and the sources of any differences are studied. The “modules” are used to define some boundaries and are a transition away from a top-down designed application to a more Object Oriented design. This component-based decomposition is needed to scale the development to larger teams. There is a tension at this stage to determine which capabilities should be placed in a module versus tightly integrated into the ad-hoc developed core. The key to remember is that all the modules are still being written for the specific application. The user base may expand to include a small group of “nice” users who are allowed to experiment with the code at this stage.

The next stage is “Modular Open Systems Architecture with Standard Key interfaces. (MOSA). This is when the interfaces to packages are defined enough to allow for modules/capabilities to be developed independently of the Physics codes. There are many modules written at LLNL that fall into this category. Hypr², SCR³, PDB⁴, LEOS⁵... The advantage of this is that each module is written once and that capability can then be quickly added to a Physics application. This is the stage where we think the Software is useful enough for a core group of users and there is an effort to give the users all the functionality they need.

The final stage is Layered Architectures. At this stage the software is layered on “standard” (COTS) layers to provide an easier (and more modern) set of development tools. At LLNL examples of this are the use of MPI, OpenMP, HDF5 and even POSIX. The Software is in full production at this stage and porting to all the systems available. Performance, usability and scaling are being optimized at this stage. The user base is expanded to anyone that needs to run the Application.

¹ <http://blog.sei.cmu.edu/post.cfm/architectural-evolution-dod-combat-systems-359>

² http://computation.llnl.gov/casc/linear_solvers/sls_hypr.html

³ <https://computation-rnd.llnl.gov/scr/>

⁴ <https://wci.llnl.gov/codes/pact/pdb.html>

⁵ https://www-pls.llnl.gov/?url=about_pls-condensed_matter_and_materials_division-eos_materials_theory-projects-material_data

To accelerate the process from moving from the “ad hoc” stage to the Layered/MOSA stage we need easy to use tools and development environments for the Software teams. Research areas to fast-track the development of Physics applications on extreme scale systems might include:

Influence and enhance the API's for software we currently rely on. Enhancements for I/O issues include adding good introspection to HDF5⁶ and a richer interface to I/O than the POSIX layer provides. (Possibly looking at the Adios⁷ API?) The POSIX enhancements might be redundant if the work on “I/O Acceleration with Pattern Detection”⁸ proves to be smart enough (and can be deployed).

Portability support for scientific software is a basic need for all of our teams. We are impacted by the complexity being shoved on the software developer and NOT on the compiler/hardware manufactures. As an example, need a portable API or Library to control Floating Point Unit (FPU) behavior. All our applications set the FPU behavior for the CPU/Vector unit and GPU (accelerator). Being able to do this in a consistent and portable manner would be very useful. Another area that impacts portability is a consistent method of launching parallel jobs. (job launching MPIrun⁹, srun¹⁰, aprun¹¹...) Finally we need optimal math routines that efficiently use the hardware on every platform these include sqrt(), pow(), matrix math (multiply, transpose,...)

Robust test frameworks are of interest to our code teams as well. The frameworks need to interact with our various batch systems and be capable of running in our security environments. Jenkins¹², and ATS¹³ are good starts. It would be useful if code coverage analysis were integrated with the test suites! (Detect and warn if the code just modified or checked in wasn't covered by the test suite.)

The HPC efforts tend to be slow in identifying noteworthy software and impacting their designs to help in the field of Extreme Scale Computing. In order to impact tool and software development we need to efficiently identify “interesting” research and assist with money, people or sample applications. Some current interesting technologies include Rust¹⁴, JSON¹⁵, Julia¹⁶, Intel TBB¹⁷, OpenCL, OpenAcc and SPIR¹⁸. To effectively engage work being done externally we should have a (small 6-8 person) working group that produces a list of potentially interesting technology (twice a year?) The report should include what is interesting about the technology, what we need to do to make it more relevant to HPC and identify action items to take to nudge the technology.

One of the trickier issues to deal with in Extreme Scale Computing is that we are the first to do it. It's difficult to learn from what other groups have done when there are so few other groups. To resolve this I propose a periodic meeting where code development groups can get together and discuss our best practices in HPC software development. These should be in the model of JOWOG's. Where we try and address HPC issues. (Debugging, I/O, build systems...) This could also cover tools, techniques, interesting hardware, software engineering and needed research areas. It may also be useful to open this working group to other agencies that do HPC work as well NOAA, DOD, NSA.

⁶ <http://www.hdfgroup.org/HDF5/>

⁷ <https://www.olcf.ornl.gov/center-projects/adios/>

⁸ <http://www.cs.cmu.edu/~garth/papers/he-hpdc13.pdf>

⁹ <http://linux.die.net/man/1/mpirun>

¹⁰ https://computing.llnl.gov/tutorials/linux_clusters/man/srun.txt

¹¹ <https://bluewaters.ncsa.illinois.edu/using-aprun>

¹² <http://jenkins-ci.org/>

¹³ <http://code.google.com/p/ats/>

¹⁴ <http://www.rust-lang.org/>

¹⁵ <http://www.json.org/>

¹⁶ <http://julialang.org/>

¹⁷ <https://www.threadingbuildingblocks.org/>

¹⁸ <http://codeldivine.org/2012/10/07/opencl-spir/>