Experience developing CnC versions of DOE Applications

Developing applications for exascale systems involves dealing with complexities such as resilience, power management, data movement and parallelism. A key exascale challenge is to define programming models, such as CnC, that hide these complexities from the user. CnC is an asynchronous, high-level task-based programming model matched to our OCR execution model. In this report, we describe our experience converting the *Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics* (LULESH) code to a CnC application.

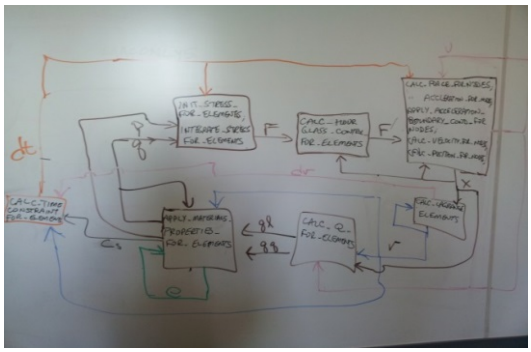

Figure 1 LULESH whiteboard sketch

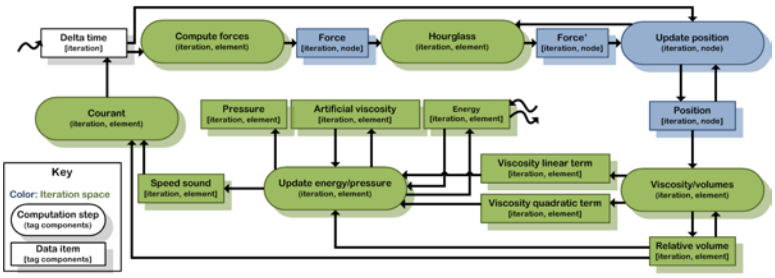

Figure 2 Formal CnC graph of LULESH

The process of developing a CnC application starts with a domain-expert's white board view of the application depicting the dataflow among software components. This sketch is then formalized, which allows compiler analysis, and optimization of parallelism, energy efficiency, and data movement. A LULESH whiteboard sketch was produced at the TG Application Workshop in August 2013 (see Figure 1). Next, we converted the whiteboard sketch to a formal graph (see Figure 2), a straightforward process that can usually be done by domain experts. In addition to being a good software design practice, formalizing the application description enables compiler analysis and optimization. It also reduces the cost associated with the development and testing stages and serves as documentation that, by the nature of its role in the process, tracks the applications evolution.

We found the process of formalizing the original graph enlightening. At the graph level before the computation steps were implemented, some things became apparent. These included misunderstandings among us, some actual bugs and some simplifications. The resulting graph hid distinctions among computations that didn't contain any communication. This caused the communication issues to pop out loud and clear. It became apparent that some optimizations we thought might have been possible were not actually legal and that several legal optimizations were now obvious. After formalizing the graph, we took the computation code from the existing version of LULESH, broke it down into to chunks corresponding to the CnC steps in our graph and wrapped them up as CnC steps and then executed the resulting application to ensure correctness.

Our next stage will be to tune the application. A tuning spec can be developed by tuning experts, domain experts or it might be automatically generated by analysis and optimization techniques. Because the tuning spec is separate there may be many tuning specs for a fixed domain spec, each of which can be developed without altering any of the CnC application code. The different specs might be for different platforms, tuning goals, input characteristics, or just multiple attempts for the same scenario.

With the implementation of LULESH completed, our next tasks will be to develop tuning specifications for it and to apply the same CnC conversion process to a second code, MiniGMG. Using LULESH, we have shown how to transform an existing application into a formal CnC specification which can then be implemented and executed using an approach that will be able to take advantage of future exascale systems.