Challenges for Component-based Multiphysics PDE Codes on Multicore Architectures

Eric C. Cyr and Roger P. Pawlowski*

The DOE has invested heavily in multiphysics simulation capabilities for solving important national security problems. Such capabilities typically require supporting multiple solution algorithms with multiple strongly coupled physics that may involve mixed discretizations. Due to the nature of multiple time- and length-scale mechanisms, multiphysics codes tend to require some form of implicitness to solve the forward problem using methods such as operator splitting, semi-implicit and fully-implicit solvers. Additionally, to be truly useful in a production capacity, applications are increasingly being designed to efficiently support “beyond forward simulation capabilities” (BFSC) including uncertainty quantification and optimization. Supporting BFSC requires evaluating quantities such as residual, parameter and response sensitivities. The goal of this paper is to highlight issues stemming from the requirements above for designing a general multiphysics framework that is efficient and portable across proposed exascale architectures. The issues in this paper stem from experience in coding general multiphysics tools in Trilinos including the Panzer and Drekar packages and from code couplings performed under the DOE CASL nuclear energy hub.

- **Suitable interface and data abstractions**: The role of abstractions will become increasingly important in the multicore era. Within our own code, Drekar, we have worked to separate the data structure internals from the implementation of user defined “physics”. Thus for an application expert or analyst they see an interface that provides the minimum data required to implement their piece of functionality; for instance they would code directly to the weak form PDE that should be implemented. This abstraction is largely sufficient on existing platforms and allows for quick implementation of many different formulations and physics. Multicore will require additional consideration of thread/memory affinity, alignment and array layout. These considerations may be beyond the knowledge base of an average analyst. Thus appropriate abstractions to hide these details will be required. Another set of issues that may not be so suitable for abstraction is how to develop a set of best practices on cache use, vectorization, and general memory access patterns. A hardware/software expert will be able to recognize these techniques at a glance; however, those implementing the physics may not. Consider for instance summing multiple arrays into a single array and the potential for cache thrashing. Appropriate abstractions will also help in the short term as the programming models are evolving.

- **Flexibility in code optimizations**: One approach to developing high performance kernels is specifying loop bounds at compile time, for instance, a fixed number of quadrature points for a particular operator, assuming a 3D/2D, or a fixed number of unknowns per node/element. This is important when issues of limited memory, memory/thread affinity and maturing programming models are relevant. Unfortunately many multiphysics codes have been written in a more dynamic style that allows reuse of key kernels by several physics formulations. Thus there is a common tradeoff between performance and flexibility that must be addressed. For applications pursuing a fixed physics, the compile time specification may be appropriate for many kernels. However, for applications pursuing a more general physics capability maintaining flexibility is a non-negotiable goal if capabilities are not to regress. Efficient multicore implementation may then require the development of memory allocation patterns that make the tradeoff between dynamic/optimized code more palatable. For instance, patterns like loop unrolling, memory pooling, and initialization passes may be useful approaches.

- **Beyond forward simulation capabilities**: One challenge of software for the multicore era has nothing to do with the specifics of the architectures themselves, and more to do with the capabilities that

*{eccyr,rppawlo}@sandia.gov, Sandia National Laboratories, Albuquerque, NM
increased computational power enables, primarily uncertainty quantification and optimization. These
methods will require software to support more flexible evaluation of quantities such as responses, re-
sponse sensitivities and parameter sensitivities. The set of possible responses and parameters required
by an analyst may represent a fairly large space of possible use cases. In particular, computing sensi-
tivities for these quantities may be challenging in a flexible multiphysics setting where the degrees of
freedom can be switched with different physics models. We believe that the use of automatic differen-
tion (AD) techniques will be more critical in future applications as a result of increased need. Thus,
scalable and robust AD approaches will need to be applied in particular to response quantities that
are defined possibly at run time. The AD tool must respect different architecture requirements such
as memory layout for optimal performance. In our experience on current architectures, AD tools have
been non-invasive to the physics, but for exascale, this may change.

• **Pathways for transitioning to exascale:** A mature research code can run scalably on existing
HPC architectures. However this does not imply ease of porting to next generation multi-core and
accelerator based architectures. For codes built using an Agile components philosophy, their multicore
implementation may depend heavily on the multi-core porting and implementation of the components.
The variation in software lifecycle models of incorporated components has already been shown to be
disruptive, providing for a difficult transition. For instance Drekar at its core has been written with
an eye towards highly synchronized assembly algorithms that should be suitable for next generation
architectures. However, achieving robust production runs, rather then demonstration calculations on
next generation architectures requires hardening of critical kernels (AMG, Krylov solve, ILU, etc...) deliv-
ered by subpackages. Currently, research into and implementation of these algorithms on next gen-
eration hardware is in its infancy and dependent subpackages are at varying stages of maturity/support
for new architectures. The application is essentially at the mercy of all required subpackages. This
will be a substantial bottleneck for porting any application code. Identifying and documenting various
transition paths for applications will be important. Within Sandia, we have seen teams use different
approaches (supporting multiple branches vs. runtime polymorphism vs. compile time polymorphism)
with varying levels of success.

• **Efficient interfaces for matrix/vector loading:** One challenge in multiphysics PDE codes is the
handling of the global assembly of matrices and vectors. The primary issue here is sparse memory access
and how it can be handled in code in a scalable and robust way. Multiphysics frameworks are usually
able to switch between different concrete implementations (i.e. PETSc and Trilinos) or even use them
concurrently, but are typically relegated to inefficient generalized interfaces to support differences in
the loading interface. This issue will only compound when exascale hardware is introduced. A focused
effort at designing efficient loading tools for DOE linear algebra libraries would be a major step forward
in interoperability and may ease transition to exascale.

• **Software Engineering and Education:** The refactor required for multicore change is a substantial
challenge, but it should also be viewed as an opportunity to reevaluate the infrastructure built up in
our codes. Investigating and implementing modern software engineering approaches could significantly
accelerate mathematical algorithm development. A major issue is the education of members of the
computational science community without experience in software engineering (and even more difficult
for willful ignorance). Overcoming these issues is the number one challenge of software in the multicore
era, which will inevitably be more complex, perhaps moving beyond where the “hero” programmer
model can handle many of the infrastructure issues.

The transition for exascale will require significant refactoring of application codes and provides new
challenges to multiphysics frameworks. The design of exascale versions of multiphysics library components
should account for a user base that is not computer science savvy. This should be done through careful
interface design that can separate the hardware complexities from physics and solution algorithms. A
concerted effort at interface design to abstract the memory layout models could go a long way towards
easing future disruptive technologies. When moving beyond forward simulation, the use of AD techniques
has been critical in our work, but these components will have to be refactored to work with new exascale
hardware requirements. Future codes should be designed with these technologies in mind. Building a set of
patterns for exascale transition that document the trade-offs may help teams transition more efficiently.