LA-UR-17-27092
Approved for public release; distribution is unlimited.

Title: Dependency graph for code analysis on emerging architectures

Author(s): Shashkov, Mikhail Jurievich
Lipnikov, Konstantin

Intended for: White paper for DOE ASCR Meeting

Issued: 2017-08-08

Title: "*Dependency graph for code analysis on emerging architectures*"
by M.Shashkov (LANL), K Lipnikov (LANL)

Direct acyclic dependency (DAG) graph is becoming the standard for modern multi-physics codes. It increases tremendeously code robustness and simplifies adding new physics models strongly coupled with the implemented models. Unlike previous attempts to develop external tools to analyze code for parallel blocks, the DAG is implemented within the code and represents on some level its skeleton. The ideal DAG is the true block-scheme of a multi-physics code. Therefore, it is the convienient object for insitu analysis of the cost of computations and algorithmic bottlenecks related to statistical frequent data motion and dymanical machine state.

The simplest example of a DAG graph is the calculation of coefficients in a system of nonlinear PDEs. For instance, relative permeability k of a porous rock depends on liquid saturation s which in turn depends on liquid pressure p which in turn depends on stiffness matrix, boundary conditions and deforming mesh. Automatic calculation of the Jacobian uses a chain rule on the dependency graph.

The nodes of the DAG graph have different complexity and may require different data communication patterns for different architectures. Automatic analysis of the DAG may show problems with the code design and algorithm bottlenecks on emerging computer architectures. For instance, if evaluation of a field requires a global solver for all primary variables, the code design most probably has a flaw. If a field evaluation requires a consecutive sequence of steps, e.g. $s = s(p)$ followed $k = k(s)$, the number of memory accesses could be reduced by stream-lining calculations by implementing one additional evaluator $k = k(p)$.

A comprehensive analysis of the DAG requires its blending with special metrics that reflect the type of a mathematical operation associated with a node, such as a field evaluation, update of a discrete operator, a mesh refinement/derefinement, or calcultion of the Jacobian. The metric may include actual and estimated flops to memory ratios. Development and analysis of various mathematical tools for DAG-enabled codes could become essential part in optimization on new multi-physics codes. It may help to select the most appropriate scalable methods (among a set of available methods) for a particular computer architecture and/or physics models.