Programming Models at Exascale:

adaptive runtime systems, incomplete simple languages, and interoperability

L.V. (Sanjay) Kale (kale@illinois.edu)

Programming models to be used on the exascale machines should be influenced by which (kind of) applications will be running on these machines. We know that the codes will typically utilize higher computational capacity of such machines with techniques such as dynamic and adaptive refinements, rather than a uniform increase in resolution. This will make programming more challenging, because of issues such as dynamic load balancing. Also, applications will need to integrate independently developed modules, including those arising from multi-physics computations.

From this, it ***follows*** that the programming models (at least a major category of them) need to free the programmer from worrying about what data is stored where, and which computation happens on what processor. This need to remove "processors" from the ontology of the application programmer should be met by future models. *Overdecomposition* (e.g. object-based overdecomposition), and *virtualization*, are examples of such an approach. With such approaches, mapping to processors must be handled by an *adaptive runtime system*.

Because of the complexity of exascale applications, alluded above, we need to make efforts to ***simplify*** parallel programming further. Admittedly, techniques such as over-decomposition simplify programming since they take away resource management concerns from the programmer. However, expressing parallel interactions will still remain substantially complex (and in some cases more complex than before, because of the use of asynchrony to gain efficiency, and issues of expressing coordination among multiple collections of interacting objects). There are at least two complementary ways for further simplifying parallel programming.

1. **Frameworks:** (restating the obvious): This includes both data-structure specific frameworks (DSSF) and higher level domain specific frameworks. We know that a small number of data structures underlie most parallel CSE codes: structured meshes including adaptively refined ones, unstructured meshes, particles distributed in uniform cells or via spatially-decomposed-trees. DSSFs can embody common functionality so it doesn't have to be programmed repeatedly. Similarly, higher level domain-specific environments (E.g. computational-astronomy environment) can collect multiple techniques making it easy to put together a complete code with different combination of strategies.
2. **Simpler but incomplete languages:** (This is probably more novel and controversial than frameworks). Restricting modes of interactions among parallel entities leads to simpler languages. Such a language may not be "complete" in that they can't express all parallel interactions (and therefore algorithms and application modules); Yet, if it covers a significant class of modules, and simplifies expression of programs that it does cover, it is a useful language. There are several candidates for such languages, including those that allow only static data flow patterns, and those that allow restricted access patterns in a global address space. We should identify and develop such languages.

Interoperability: The above approaches will succeed (in the context of multi-physics, multi-scale, and multi-module exascale applications) only if they can be made to *interoperate* efficiently without losing simplicity. This means entities from different modules must be able to interleave there execution on individual processors without them being aware of each other explicitly. Message driven execution appears to be a pre-requisite for such interoperability, but the research community may aim at finding multiple methods for supporting such interoperability.