Some Thoughts on Programming Languages for HPC

William D. Gropp
Mathematics and Computer Science

Argonne National Laboratory
A U.S. Department of Energy
Office of Science Laboratory
Operated by The University of Chicago
Questions to Panel

• Which language in 2015?
  - Something (C, Fortran, Java, …) + MPI-3
    - MPI is already 14 years old, 2015 is only 9 years away
    - Domain-specific, higher-level languages

• Do we need a new paradigm?
  - Yes: composable, customized micro-languages targeted at particular goals

• Commercial aspects?
  - Argues against an all-in-one solution; must leverage other tools with a broader market

• Performance and programming optimization?
Performance is #1

- No-one wants to write a parallel program
- They do it because they need to for performance (why use 8 oxen if you have 1 really strong ox?) or memory
- Lack of performance predictability makes it difficult to design algorithms, data structures, and code
  - “psychoanalyze the compiler”
(Almost) Every operation in the hardware is a split operation

- Every operation is pipelined
- Even on slow processors (like 700MHz BG/L), as many as 5 cycles required. 25+ for some chips
- Memory references can take 100+ cycles, remote refs in a Petascale system will take 1000-10000 cycles
- Several operations may start in the same cycle
- A sequential listing of program steps doesn’t reflect the hardware execution model, leading to a lack of performance transparency
- Programmers have a hard time thinking in terms of operations that are completed “later”, even at a high level
  - MPI_Get( &a, .... ); if (a == ...)
- Suggests that the separation between initiation and completion should be visible to the performance programmer
A Visual Performance-Oriented Language View

- Provide a view of, e.g., a loop that shows time in y and concurrent operations in x, color for op type, shading for begin/end, hazards
- Provide a view that shows memory access (query user and/or run program to determine necessary info)
  - E.g., access patterns for FFT kernel
- Changes to the *graphical representation* should edit the “conventional” representation
- Represent uncertainty (e.g., load costs for L1/L2/L3/Mem/RemoteMem)
- Has the computer do what performance programmers must do “by hand” now
- Not a new language — a different view of existing languages
  - May need small extensions to language, e.g., asm(…)