Working Group 5
Programming Environments and Tools

Chair: Dennis Gannon

Vice Chair: Rich Hirsh

WG5 – Programming Environments and Tools Charter

- Charter
  - Address programming environments for both existing legacy codes and alternative programming models to maintain continuity of current practices, while also enabling advances in software development, debugging, performance tuning, maintenance, interoperability and robustness. Establish key strategies and initiatives required to improve time to solution and ensure the viability and sustainability of applying HEC systems by the end of the decade.
- Chair
  - Dennis Gannon, Indiana University
- Vice-Chair
  - Rich Hirsh, NSF

WG5 – Programming Environments and Tools Guidelines and Questions

- Assume two possible paths to future programming environments:
  - incremental evolution of existing programming languages and tools consistent with portability of legacy codes
  - innovative programming models that dramatically advance user productivity and system efficiency/performance
- Specify requirements of programming environments and programmer training consistent with incremental evolution, including legacy applications
- Identify required attributes and opportunities of innovative programming methodologies for future HEC systems
- Determine key initiatives to improve productivity and reduce time-to-solution along both paths to future programming environments
- Example topics:
  - Programming models, portability, debugging, performance tuning, compilers

Charter

- Address programming environments for both existing legacy codes and alternative programming models to maintain continuity of current practices, while also enabling advances in software development, debugging, performance tuning, maintenance, interoperability and robustness.
- Establish key strategies and initiatives required to improve time to solution and ensure the viability and sustainability of applying HEC systems by the end of the decade.

Guidelines

- Assume two possible paths to future programming environments:
  - incremental evolution of existing programming languages and tools consistent with portability of legacy codes
  - innovative programming models that dramatically advance user productivity and system efficiency/performance
- Specify requirements of programming environments and programmer training consistent with incremental evolution, including legacy applications
- Identify required attributes and opportunities of innovative programming methodologies for future HEC systems
- Determine key initiatives to improve productivity and reduce time-to-solution along both paths to future programming environments

Key Findings

- Revitalizing evolutionary progress requires a dramatically increased investment in
  - Improving the quality/availability/usability of software development lifecycle tools
  - Building interoperable libraries and component/application frameworks that simplify the development of HEC applications
- Revitalizing basic research in revolutionary HEC programming technology to improve time-to-solution:
  - Higher Level programming models for HEC software developers that improve productivity
  - Research on the hardware/software boundary to improve HEC application performance

The Strategy

- Need an attitude change about software funding for HEC.
  - Software is a major cost component for all modern complex technologies.
- Mission critical and basic research HEC software is not provided by industry
  - Need federally funded, management and coordination of the development of high end software tools.
  - Funding is needed for
    - Basic research and software prototypes
    - Technology Transfer:
      - moving successful research prototypes into real production quality software.
  - Structural changes are needed to support sustained engineering
    - Software capitalization program
    - Institute for HEC advanced software development and support.
      - Could be a cooperative effort between industry, labs, universities.

The Strategy

- A new approach is needed to education for HEC.
  - A national curriculum is needed for high performance computing.
  - Continuing education and building interdisciplinary science research.
  - A national HEC testbed for education and research

The State of the Art in HEC Programming

- Languages (used in Legacy software)
  - A blend of traditional scientific programming languages, scripting languages plus parallel communication libraries and parallel extensions
    - (Fortran 66-95, C++, C, Python, Matlab)+MPI+OpenMP/threads, HPF
- Programming Models in current use
  - Traditional serial programming
  - Global address space or partitioned memory space (mpi+on linux cluster)
  - SPMD vs MPMD

The Evolutionary Path Forward Already Exists

- For Languages
  - Co-array Fortran, UPC, Adaptive MPI, specialized C++ template libraries
- For Models
  - Automatic parallelization of whole-program serial legacies no longer considered sufficient,
    - but it is important for code generation for procedure bodies on modern processors.
  - multi-paradigm parallel programming is desirable goal and within reach

Short Term Needs

- There is clearly very slow progress in evolving HEC software practices to new languages and programming models. The rest of the software industry is moving much faster.
  - What is the problem?
  - Scientists/engineers continue to use the old approaches because it is still perceived as the shortest path to the goal … a running code.
- In the short term, we need
  - A major initiative to improve the software design, debugging, testing and maintenance environment for HEC systems

The Components of a Solution

- Our applications are rapidly evolving to multi-language, multi-disciplinary, multi-paradigm software systems
  - High end computing has been shut out of a revolution in software tools
    - When tools have been available centers can't afford to buy them.
    - Scientific programmers are not trained in software engineering.
  - For example, industrial quality build, configure and testing tools are not available for HEC applications/languages.
    - We need portable of software maintenance tools across HEC platforms.

The Components of a Solution

- We need a rapid evolution of all language processing tools
  - Extensible standards are needed: examples -language object file format, compiler intermediate forms.
  - Want complete interoperability of all software lifecycle tools.
- Performance analysis should be part of every step of the life cycle of a parallel program
  - Feedback from program execution can drive automatic analysis and optimization.

The Evolution of HEC Software Libraries

- The increasing complexity of scientific software (multi-disciplinary, multi-paradigm) has other side effects
  - Libraries are an essential way to encapsulate algorithmic complexity but
    - Parallel libraries are often difficult to compose because of low level conflicts over resources.
    - Libraries often require low-level flat interfaces. We need a mechanism to exchange more complex and interesting data structures.
- Software Component Technology and Domain-specific application frameworks are one solution

Components and Application Frameworks

- Provides an approach to factoring legacy into reusable components that can be flexibly composed.
  - Resource management is managed by the framework and components encapsulate algorithmic functionality
- Provides for polymorphism and evolvability.
- Abstract hardware/software boundary and allow better language independence/interoperability.
- Testing/validating is made easier. Can insure components can be trusted.
- May enable a marketplace of software libraries and components for HEC systems.
- However, no free lunch.
  - It may be faster to build a reliable application from reusable components, but will it have performance scalability?
    - Initial results indicate the answer is yes.

Revolutionary Approaches:The Long Range View

- We still have problems getting efficiency out of large scale cluster architectures.
- A long range program of research is needed to explore
  - New programming models for HEC systems
  - Scientific languages of the future:
    - Scientist does not think about concurrency but rather science.
    - Expressing concurrency as the natural parallelism in the problem.
      - Integrating locality model into the problem can be the real challenge
    - Languages built from first principles to support the appropriate abstractions for scalable parallel scientific codes (e.g. ZPL).

New Abstractions for Parallel Program Models

- Approaches that promote automatic resource management.
- Integration of user-domain abstractions into compilation.
  - Extensible Compilers
- Telescoping languages
  - application level languages transformed into high level parallel languages transformed into … Locality may be part of new programming models.
- To be able to publish and discover algorithms.
- Automatic generation of missing components.
- Integrating persistence into programming model.
- Better support for transactional interactions in applications

Programming Abstractions cont.

- Better separation of data structure and algorithms.
- Programming by contract
  - Quality of service, performance and correctness
- Integration of declarative and procedural programming
- Roundtrip engineering/model driven software
  - Reengineering: Specification to design and back
- Type systems that have better support for architectural properties.

Research on the Hardware/Software Boundary

- Instruction set architecture
  - Performance counters, interaction with VM and Memory Hierarch
- Open bidirectional APIs between hardware and software
- Programming methodology for reconfigurable hardware will be a significant challenge.
- Changing memory consistence models depending on applications.

Research on the Hardware/Software Boundary

- Predictability (scheduling, fine-grained timing, memory mapping) is essential for scalable optimization.
- Fault Tolerance/awareness
  - For systems with millions of processors the applications/runtime/os will need to be aware of and have mechanisms to deal with faults.
  - Need mechanisms to identify and deal with faults at every level.
  - Develop programming models that better support non-determinism (including desirable but boundedly-incorrect results).

Hardware/Software Boundary: Memory Hierarchy

- There are limits to what we can do with legacy code that has bad memory locality problems.
- Software needs better control data structure-to-hierarchy layout.
- New solutions:
  - Cache aware/cache oblivious algorithms
  - Need more research on the role of virtual memory or file caching.
  - Threads can be used to hide latency.
  - New ways to think about data structures.
    - First class support for hierarchical data structures.
  - Streaming models
  - Integration of persistence and aggressive use of temporal locality.
  - Separation of algorithm and data structure, i.e. generic programming.
  - Support from system software/hardware to control aspects of memory hierarchy.

Best Practices and Education

- Education is crucial for the effective use of HEC systems.
  - Apps are more interdisciplinary
    - Requires interdisciplinary teams of people:
      - Drives need for better software engineering.
  - Application scientists does not need to be an expert on parallel programming.
    - Multi-disciplinary teams including computer scientist.
  - Students need to be motivated to learn that performance is fun.
    - Updated curriculum to use HEC systems.
    - Educators/student need access to HEC systems
  - Need to increase support for student fellowships in HEC.