Development Environment Preparation (parallelism support, compilers, tools)

DOE HPC Operations Review

San Francisco, November 5-6, 2013

LLNL-PRES-646612

Breakout participants

- John Gyllenhaal, LLNL*
- Joel Stevenson, SNL*
- David Skinner, LBL; Richard Gerber, NERSC
- Ray Loy, ANL; Tim Williams, ANL
- Bronson Messer, ORNL
- Jennifer Green, LANL
- Karen Haskell, SNL
- Brian Carnes, LLNL

* Denotes breakout session lead

Effort estimate

- How big of an effort was this?
  - Every lab expressed desire for more people than they had.
  - Only subset of desired effort projects could be done

Processes (scope of activity)

- What needs to be done?
  - A useful HPC development environment doesn't come "shrink wrapped" from vendor
  - Prioritize - determine what you need (from who and by what date)
  - Recommended defaults - empirically determined "good" defaults
  - Continuity with previous platforms is helpful to users
  - Feature Complete environment – has all of the features necessary
  - Be precise/complete when procuring systems – must understand the system/user requirements
  - Ask users what they need and then verify (responses have a shelf-life)
  - Cross-compiling creates challenges for users
  - Handling of shared libraries on systems needs to be improved
  - Do you encourage users or do they "encourage" you (user driven at NERSC) – User: this is how I want things to work
  - Who gets to say "If you need xyz, then you are doing it wrong"

Processes (scope of activity), cont.

- What begins first: timeline for activities (before or after hardware)?
  - Representative benchmarks for key features
  - Direct investment as part of the contract - included tool development AND application code porting/development as part of the contract/project – got diversity, stratified support list
  - Involve users early
  - Run things you care about (and have had trouble with before) as early as possible
  - Timeline: s/w emulator, early h/w, etc.
    - However, by nature, simulators are very limited, several orders of magnitude slower, do not support I/O, too limited. Main benefit new feature exploration.
    - Early prototypes – spend time running out to the end only to find that the effort did not pay off (due to vendor changes in the delivered system)
    - The first piece of “real” hardware/software is the most important step
    - From a best practices perspective maybe there is a more productive may to spend our time/money rather than on emulators, prototypes – wasted time if the delivered hardware/software is different, going down the “wrong” path too many times, throwing code away, diverting people

Processes (scope of activity), cont.

- What begins first: timeline for activities (before or after hardware)?
  - Early super friendly users
    - Technically capable, forgiving, in need
  - Then friendly users
  - Start thinking about adding more users when?
    - Reasonable chance of success
    - Can you support them

Processes (scope of activity), cont.

- What is the role of early hardware access (either locally or remotely) and prototype systems?
  - Vendor interpretation of X may very different that ours – vendor usage model may not be accurate
  - Education of users and vendors – example: vendors assert they meet the spec, but users may have a different interpretation of what will be implemented and code to that – when things don't work we have a problem - compliance with spec does not mean that every feature is implemented – recalibration of expectations

Processes (scope of activity), cont.

- What is the role of vendor partnerships?
  - Does scope include people who write UPC, etc.? Yes
  - Does scope include 3rd party? Yes
  - External login nodes and internal nodes with different software stacks is problematic

Processes (scope of activity), cont.

- What are the roles of research and design and engineering?
  - Research agreement with vendor on key set of applications – vendor tuning of key apps – requires someone on vendor side and someone on tools/code team to be assigned
  - Research contracts after delivery of system
    - Openmp overhead reduction
    - CAPS – host and device resident in structs
  - More system focused issue – D&E – once you know who vendor will be, higher level of customization/focus – paying vendor to benefit end user community - helps end-user directly with this vendor

Processes (scope of activity), cont.

- What resiliency activities are executed (for example, redundancy)
  - Nice/necessary to have multiple compiler/mpi vendors (and versions)
  - Always good to have two development machines – can carve out if possible – its about the portion you can update separately
    - One machine for updating OS – machine will be unusable for a few days – code developers will be down during this time
    - Second machine would enable code developers to stay up

Organization and management

- What is the structure of the integration and preparation teams? (development environment team)
  - System level tools, compilers – sys admins
  - User-level tools and libraries – applications team
  - Application readiness team
  - Code developers, vendor staff, development environment staff - matrix development environment personnel to code development teams
  - Acceptance team that tests and accepts system

Organization and management, cont.

- What are the necessary skills for the activity team? (development environment team)
  - The development environment team needs the following skills:
    - Domain experts on tools (parallel tools team)
    - Application readiness team – computer science, domain knowledge (science, engineering)
    - User liaisons – problem-solvers – talk to domain experts, code teams and users (determine issues - code? platform?)
    - Front line support

Experiences and lessons learned

- What were the good and bad experiences and lessons learned?
  - Very limiting to have only one development environment (only one compiler)
    - Put language in procurement (more than one compiler, more than one debugger, etc – specify the numbers in contract) or go directly to vendor after selection with secondary contract
  - Secondary contracts – keep competition in mind
    - Debuggers contracts worked well
    - Memory tools, thread correctness, performance tools, stack trace analysis (where are you hanging or crashing), basic profiling
    - Competition is a good thing with tools – be very careful about funding only one vendor in one area
    - Salesmanship vs substance – put in SOW that tool will be provided through vendor or 3rd party
  - Licensing per rank is untenable – floating licenses vs fixed licenses – what is reasonable?
    - Hardware cost per rack is constant – we need software licensing to be similar

Experiences and lessons learned, cont.

- What were the most productive activities?
  - Not all activities were contractual – i.e. some “relationships” were productive
    - Getting tool developers on the system for early access
      - Getting a proper version of PAPI; HPC toolkit
    - Consultants to work with users on site
  - System workload testing (SWL) – series of functional, robustness, performance testing
    - Tests that allow you to construct/keep a baseline – basis for forensics, regression
    - Formal process – basis for acceptance testing, Q&A
  - Test frameworks/harnesses - store results and correlate events
  - Software build frameworks
  - Library instrumentation – everything gets scanned into a database

Experiences and lessons learned, cont.

- What was the highest risk? Was it a surprise or expected?
  - (not answered due to time constraints)

Most significant observation

- Provide a summary statement for the most significant observation
  - Disparity between the user group requests (requirements) and how the users convey those requests to us and how that goes into SOW and how the vendors interpret these requirements
  - Significant because this interpretation/miscommunication cycle is a persistent problem -
    - Getting harder to measure performance – no HW counters on GPU
    - Despite progress, debuggers are still having trouble at scale– usability is an issue - – remote use is not easy - client/server GUI is a prime example of usability - nx mentioned as possible mitigation technology
    - For sites with mainly transient users, debuggers had low utilization except for users with prior debugger experience. Other sites with ‘stable’ user sets reported many users that lived in the debugger.