



ASC Exascale Environment Workshop Out-brief



Applications Working Group

B. Still (LLNL), T. Adams (LLNL), M. Anderson (HQ/LANL), A. Arsenlis (LLNL), J. Belak (LLNL), J. Bell (LBNL), M. Bement (LANL), R. Bond (SNL), T. Brunner (LLNL), L. Cox (LANL), D. Daniel (LANL), T. DeGroot (LLNL), W. Aldo (HQ), C. Ferenbaugh (LANL), R. Harrison (ORNL), J. Johannes (SNL), A. Koniges (LBNL), K. Mish (SNL), A. Nichols (LLNL), R. Robey (LANL), M. Steinkamp (LANL), S. Swaminarayan (LANL), D. Womble (SNL)



Working Group Description



- Scope (i.e. major sw stack elements responsible for)
 - Science and Specialized codes [Physics and Engineering Models (PEM)]
→ model development and improved science understanding
 - Integrated Codes (IC) → mission critical national security applications and uncertainty quantification (UQ)
 - Codes to be used in Validation and Verification (V&V)
- Key dependencies on other working groups
 - (SAL) linear solvers, applied math algorithms
 - (PM) programming models used to implement physics models, compilers
 - (TOOLS) code development, compilers, debugging, performance analysis, and correctness checking
 - (IONS) checkpoint/restart
 - (VDA) exploration and analysis of simulation results
 - (SSW) and (HWA) compilers, runtime environment, fault tolerance/recovery, power management



Exascale Challenges



- List 3-5 most important challenges, pertaining to this WG's scope, to achieving exascale
 - Portability / Maintainability (exascale and non-exascale architectures)
 - Codes are complex already; maintaining arch specific implementations not practical
 - Transition from existing code base while needed / Interoperability
 - Managed Code Development (ie, migration to new physics modules)
 - Application Robustness, Resilience, Fault Tolerance
 - Data Locality / Memory Management
 - Required for power efficiency; manage flops → manage data
 - Overall Concurrency
- Challenges we'd like to see handled elsewhere, perhaps with hints from the Apps...
 - Power Management (HWA, SSW)
 - System Robustness, Resilience, Fault Tolerance (SSW, HWA, ...)



Additional Challenges for Exascale



- **Application users are a missing element in our discussions**
 - **Social Engineering – confidence through the transition**
 - **Next generation design code has to be certified before it can be put into production**
 - App users are more likely to accept changed answers if differences are understood (eg, new rad model ok, different optimization bad)
 - **Some robustness and resilience requirements are specified by our user community**
- **Programmatic support is essential during exascale transition**
 - **Programmatic resource adjustments have been studied**
 - **Need to **augment** with a right-sizing study for exascale development**
 - **Not only the **right number** of staff, but also the **right mix of technical skills** to address the complex and varied challenges of exascale**
 - **Workforce development/recruitment is a challenge**
 - **Redefine code team to include more cross-discipline (no more “throw it over the fence”) – the code team /S co-design**



Cross-cutting Challenges



- A viable *standardized* programming model (or a few) is critical to all of the WG areas
 - Apps must be supported on capacity as well as capability systems so **portability is key**
 - PM must be portable and standardized (multi-vendor, multi-architecture support)
 - **Higher level abstractions** are a demonstrated path to portable performance
 - PM should be hierarchical
 - Exposing underlying details is good for codes that can take advantage, but **low-level programming should not be required**
 - Something beyond C+MPI+threads (e.g. CILK) is needed
- Exascale environment evolution will likely follow two phases: mutation and natural selection
 - Different challenges and opportunities exist for each phase
 - How to take advantage and mitigate risk?



Path Forward 1



- **Develop skeleton compact/mini apps**
- Initial next steps
 - Identify important app kernels, and select a few for implementation, discussion with other WG
 - Follow a top-down approach to engaging WG
- Timeline (Phase I, II or III; Evolutionary or Revolutionary) I, Evo – FY12
- Required Partnership: SAL, PM, TOOLS [+ OASCR, Academia]
- Follow on: Phase I-II
 - Engage with IONS, VDA, SSW and HA
- Risks: Simplicity vs Relevance: Too simple, not relevant
 - It is important to allow failure and learn from it.
 - Multiple iterations likely to be needed.



Path Forward 2



- **New performance metrics (other than FLOPS)**
 - For current codes and mini-apps
- Initial next steps
 - Identify architectural characteristics and constraints (memory, locality, I/O, or ... ugh... power)
 - Feed back into TOOLS and PM
 - Implement into mini-apps
- Timeline (Phase I, II or III; Evolutionary or Revolutionary) I, Rev – FY12
- Required Partnership: ALL [+OASCR, Academia]
- Risks:
 - Hardware is in flux, and simulators may not be sufficient
 - Access to early test-bed systems can reduce risk



Path Forward 3



- **Application resilience, robustness, fault tolerance**
- Initial next steps
 - Co-design with TOOLS, PM, SAL
 - Incorporate mitigation techniques for system faults not treated lower in the software stack
- Timeline (Phase I, II or III; Evolutionary or Revolutionary) **II, Rev**
- Required Partnership: **TOOLS, PM [+ OASCR, Academia]**
- Risks: System may be too unstable to compute on reliably
 - Hard to realistically quantify frequency expected failure (no good model)
 - Existing algorithms may not be good enough to do the job
 - Desired algorithms may not fit within allowable power envelope
 - TOOLS and PM may not make commensurate investment on needed time scale



Path Forward 4



- **Develop new algorithms for increased concurrency**
- Initial next steps
 - Task based parallelism, unsplit, vector/array
 - New physics enabled by exascale
 - Embedded UQ
 - In-situ data analysis
 - Note: power mgmt could help load balancing: down-clock lighter loads
- Timeline (Phase I, II or III; Evolutionary or Revolutionary) **II, Rev**
- Required Partnership: **ALL [including OASCR, Academia]**
- Risks: Unknown (High) level of complexity
 - Staffing and availability of required skills
 - Exascale development versus mission demands



Path Forward 5



- **Applying knowledge to production codes**
 - Migration of existing code (evolutionary)
 - Implementation of new next generation codes (revolutionary)
- Initial next steps
 - Analysis and abstraction of current apps; new computational algorithms
- Timeline (Phase I, II or III; Evolutionary or Revolutionary) **III, Evo+Rev**
- Required Partnership: **ALL [including OASCR, Academia]**
- Risks: **Yes. (Failure is not an option...)**
 - Other path forward items help mitigate some risk
 - PM and TOOLS must be robust and stable
 - Staffing and availability of required skills
 - Exascale development versus mission demands
 - Inertia



Points for Co-Design/Co-Exploration



- **Data layouts → Abstractions across packages**
 - How is it decided? Where is it decided? Data iteration/traversal has to be abstracted. How many layers?
 - If PM generates, may compile to different architectures
 - Expressing isolation (functional programming)
- **Develop skeleton compact/mini apps – simplicity vs relevance**
 - Language construct or API? Evolving PM
 - Composition: Is it Y or MPI+X? How about non-SPMD?
- **Task level concurrency (Concurrency flow)**
 - Convergence, Reliability, Stability, Error Analysis for MPMD physics integration and for inline models
- **Embedded UQ, Viz/Data analysis, ...**
- **Profiling techniques and feedback**
 - Tool building block use within apps



The End





Notes From Initial Discussion 1



- One big issue isn't technical: We'll need people with a unique skill set, with detailed architectural programming and unique physics.
- Robust middleware. (Compilers, libraries, etc.) have to work, and adhere to standards, and be high quality. They need to be robust products, with good SQE and support.
- Resiliency in the libraries to code or hardware failures.
- Solvers need good algorithmic scaling. (Scaling with unknowns and processors.) Coupling various algorithms in multiphysics.
- Data structures shared between host code and libraries, maybe not ideal for either, but good compromise of both to reduce data motion.
- Compatibility/interoperability between multiple libraries, the apps, etc. Could have detailed API's, or pull together multiple library teams and app team to write specific solution for app. Use specialized knowledge library teams to solve problem.
- Get better at good documentation to record physics and algorithm choices.
- Have some way to better translate the math/science into code. The code changes for each machine, but the problem doesn't. Can we have different abstraction layers?



Notes From Initial Discussion 2



- Migration of legacy codes (including new ASC MPI codes and the really old ones) to the exascale will be difficult. How do we capture all the knowledge in those codes so we can run well on new machines. Social engineering the migration between codes to get designer acceptance. How to resolve differences in results.
- Machines are getting more complex, as are the codes and physics problems. It's becoming unmanageable, especially moving between multiple, different. Sustainability is key. Can we be given a tool that will tell us how a high-level code is translated into machine code so we know how to make it more efficient.
- What does correctness mean in Exascale venue? (Solution verification)
- Abstraction ideals: How to express computational goals so that Apps->Programming model->compiled binary. Do we need task abstraction too?
- What are possible programming models? MPI+X, or something else. Whatever it is needs to be compatible with current code to facilitate migration of current code base.
- Multiple failures drive restart: Physics failure, single-few node HW failure, whole machine, end of allocation to start of next job.
- Abstractions are dangerous too. Key new algorithms came from struggling with the details of MPI and data motion.



Notes From Initial Discussion 3



- Fault tolerance is a big issue. Do we just run each calculation multiple times. We can interleave check pointing and work. Dispatching master/slave model (in some sort of tree), to be robust to workers dying. But we need to take into account dependency graph between physics modules/tasks.
- Single physics codes can run on all the whole machine. Integrated codes tend to run a mixture of small jobs (~1-2k processor jobs) and full scale capability simulations (exploratory, high fidelity physics, ...). Efficiency of end-user, not the machine.
- We're going to have to go to more recomputing instead of storing data. We're shifting from managing FLOPS to memory.
- Co design with the other end: the Users. What questions would they ask differently, or what questions they'd ask?
- Tools to tell us what memory movement is good/useful vs bad/wasteful? (Latency hidden is OK, for example.)