Exascale Software Center

Co-Design Code Team Survey (Application Inventory)

Version 1.2
02/15/10

Below is a survey developed to provide a snapshot in time of what science application teams, working software infrastructure builders and hardware vendors in the co-design process, believe they will need to achieve their goals for exascale science. To this end, the survey should be applied to each major application code; if the application area in view has multiple codes, please repeat for each. Not having an answer yet for some of the hard questions is completely fine, but we certainly welcome "best guesses" when they are available.

Questions in each area generally focus on one of two things: what the application teams/codes are using now, and what they expect to use or do for exascale. In the "Application Overview," "Parallel Programming," and "I/O Patterns and Strategy" sections, the questions related to each of these two perspectives are factored out and grouped together in clearly marked "a" and "b" subsections. When it comes to setting expectations for exascale, we recognize that application teams can only provide a snapshot, given the available information.

We also recognize this is a lot of work. It may take more than an hour to walk through the material. However, we believe, and the experience we have had so far confirms, that it will be extremely valuable for planning when combined with a top-down perspective of what the future architectures and software stacks may be able to achieve.

If there are data or responses that should not be shared outside the ESC planning team, please mark it appropriately.

---

1. Survey Metadata

- Project Name:
- Date:
- Name(s) of ESC person(s) facilitating the survey:
- Name(s) of Science Application team member(s) contributing to the survey:

2a. Application overview: Current

- *Science Goals*: What are the main goals of scientific research agenda that inform this application?

- *Sponsors*: Who are the major sponsors?

- *Application function*: In brief, what does the application do?

- *History*: In brief, what is the history of this project?

- *Platforms*: What platforms/architectures does the code currently run on?

- *Science application team*: What are the main characteristics, with respect to overall workflow, of this application's research team/community, e.g. size, geographic distribution, data providers, result interpreters, etc.

- *Size of code*: What would be a rough estimate of current size of code base and languages currently used (e.g. Fortran90, C, C++, Python, etc.)?

- *Computational method*: What computational methods are used today? (E.g. structured grid, n-body, graph, dense matrix, sparse matrix, Monte Carlo, spectral, etc.)

- *Future application goals*: What are your general application goals for 3, 5, and7 years from today?

- *Application pain points*: What are your top current pains?

- *Mechanics for collaboration:*
    - Briefly, what is your software development, testing, and release tools/methodology

    - How can other software developers get a copy of the code? (e.g. licensing, etc.)

    - Does the project currently maintain a "compact" version with standard input and outputs for testing and benchmarking? Is that part of your exascale plan?

2b. Application overview: Expectations/conjectures for exascale

- *Science Goals*: What science breakthroughs do you expect when your application is scaled to exascale systems?

- *Platforms*: What work are you doing, or planning to do for GPGPUs? (e.g.: OpenCL, CUDA, Compiler technology)

- *Science application team*: What, if any, changes in the science application team/community are expected on the way to exascale (e.g. new data sources coming on-line, etc.)

- *Computational method*: What are your expected changes in computational methods for Exascale science?

- *Application pain points*: What do you expect your top pains to be for exascale?

3a. Parallel Programming: Current

- *Expression of parallelism*: How is parallelism currently expressed? (MPI everywhere, MPI + OpenMP, Pthreads, Global Arrays, Language (X10, UPC, Chapel, etc.), etc.)

- *Exposure of parallelism*: Do most developers/users see the constructs for parallelism, or are they largely hidden in a framework/library? If so, please explain.

- *Data communication mechanisms*: How does the code pass data between address spaces (nodes) (e.g.: MPI, UPC, shmem put/get, GlobalArrays, custom)? Do you use MPI-2 RMA?

- *Load balancing*: How is your code load balanced? Do you rebalance work dynamically during execution?

- *Memory requirements*: What are the current memory requirements of your application, including size, bandwidth, scaling, duplication, write-once, and usage patterns?

- *Library requirements*: What libraries do you currently link with (e.g. PETSc, scalapack, FFTW, Trilinos, Hypre, Plasma, etc)? What are your future plans?

- *Coupled codes*: Do you currently use coupled codes?

- *Run-time software infrastructure*: What run-time support and libraries does your application need on compute nodes (e.g.: user-level threads, pthreads, java, multiple processes, python, sockets, perl, etc.)?

- *Global data operations*: Do you use global operations/collectives? If so, which?

- *Data layout*: Briefly, what are the key data structures and memory layout for your computation?

3b. Parallel Programming: Expectations/conjecture for exascale

- *Expression of parallelism*: How will parallelism likely be expressed for your exascale code?

- *Exposure to parallelism*: How do you expect the exposure of developers/users to the constructs for parallelism to change as the application moves toward exascale? For example, will more have to be hidden within libraries?

- *Data communication mechanisms:* What plans do you have, if any, for passing data between address spaces (nodes) on exascale systems?

- *Load balancing*: What changes in your approach to load balancing do you expect to have to make for exascale?

- *Memory requirements*: Given what we understand about the platform architecture "swim lanes" so far, what do you think your job size and per-node memory footprint would be for routine and heroic runs on the 2015 and 2018 systems? If this categorization isn't appropriate, please feel free to reformulate.

- *Library requirements*: Which libraries, if any, do your current plans assume will be available on future exascale platforms?

- *Coupled codes*: Are coupled codes in your roadmap? If so, please explain.

- *Exascale architecture opportunities*: What are your plans for exploring advanced architectural features (e.g. transactional memory, user-defined memory prefetch lists, SIMD, gather/scatter, etc.)?

4a. I/O Patterns and Strategy: Current

- *Specifics of data usage*:
    - How much data is typically needed as input to a run?

    - How is this data organized and selected?

  - What other data needs occur at job startup? For example, do you count on dynamic loading of libraries? If so, how are these linked in (e.g., Python libraries)?

- *Strategy for fault tolerance*:
  - Do you near term have plans for fault tolerance in your application that would eliminate the need for checkpointing?
  - If not, what fraction of the allocated memory do you expect to be needed for restart in the event of failure?
  - How do you see this fraction changing as systems scale?

- *Data I/O strategy:*
  - What model best represents how your code treats data in memory: regular structured mesh, regular unstructured mesh, adaptive structured or unstructured, objects, something else?
  - Do you use libraries for storing application data, or do you write directly to the file system? If you do use libraries, which ones?
  - Is there a plan to change I/O strategy in the near future? If so, will it involve using existing libraries or developing your own?

- *Data access pattern*: How many files per rank does your application access? Is it dependent on phase (e.g., an initial file, then checkpoints with intermediate results once every N checkpoints)?

- *Temporary file requirements:* Some applications are "pipelines", i.e. are composed out of several job launches that work sequentially to accomplish a task. If your application is a pipeline, does it produce intermediate temporary files that are written between the various stages of the pipeline? If there are temporary files, can you briefly describe these temporaries for us (size, lifetime, IO pattern)?

- *"Out-of core" requirements*: If your application does work "out-of-core" (i.e., reads back what it has previously written in the same run), are you contemplating continuing this approach in the future? How would your plans for this strategy change if the ratio of memory capacity to CPU changes radically?

- *I/O benchmarks requirements*: Do any benchmarks currently exist that are representative of your I/O patterns? Which one(s)? What parameters are relevant for the benchmarks?

- *Analysis of data output*:
    - How is data analyzed?
    - Is any analysis performed during runtime? If so, is analysis built into your application in some way?
    - To what degree is your analysis performed using parallel tools?

4b. I/O Patterns and Strategy: Expectations/conjecture for exascale

- *Strategy for fault tolerance*: One general concern about Exascale systems is the overhead inherent in global synchronization, such as synchronized checkpointing. To what degree do you see your application moving away from this kind of synchronization in the 2015, 2018 time frame?
- *Analysis of data output*: Given the amount of data you expect your application be generating in the 2015 and 2018 time frames, please answer the following:
    - What fraction of the allocated memory will be needed to perform analysis?
    - How often would you need to capture data?
    - How much total data would be generated in a routine or heroic run?
    - If the amount of data needed is different between testing and science runs, why are the two cases different?

5. Data Analysis and Visualization: Current and Expectations/conjectures for exascale

- *Data exploration/analysis tools*: How do users currently explore/analyze the data generated? What tools are used? Who provides them?
- *Visualization workflow*: How would you describe your visualization workflow, if any?
- *Data exploration/analysis at exascale*: What do you expect the main data visualization and analysis problems to be at exascale? How do you plan to address them?

6. Performance: Current and Expectations/conjectures for exascale

- *Performance tools*:

  - What tools do you use now to understand performance (e.g. TAU, HPC Toolkit, PAPI, VAMPIR, etc.)? Are sample trace or log files available?

  - What features would you like to see in performance tools (e.g. ease of instrumentation, different measurements, embedded collection of I/O, etc.)

- *Performance models:* Do you have an analytic performance model (at any level of detail)? If so, how accurate or predictive is this model?

- *Special instructions*: Does the application make effective use of vector (e.g., SSE or VSX)?

- *Presumed performance bottleneck:* What do you believe is a key bottleneck to better performance?

- *Presumed scaling bottleneck*: How large does your application currently scale (cores)? What do you believe is a key bottleneck to better *scaling*?

- *Application autotuning*: How, if at all, you used an autotuner or other tools that compile, run, and test the performance of blocks of code to optimal parameters? Would you consider using an autotuner? If not, why not?

- *Exascale performance optimization*: What are your future plans for exascale performance optimization?

Tools: Current and Expectations/conjectures for exascale

- *Application debugging:*
  - How do you currently debug your code for small runs?

  - How do you currently debug your code for large, at-scale runs?

- Special tool needs:
  - What other tools do you use? (e.g. correctness tools, modeling tools, workflow job managers, etc.)

  - Do you have or any special needs for job submission, control, and management? (E.g. many-task, interactive connections for visualization, graph-execution, etc.).

  - Will these special needs change for exascale?