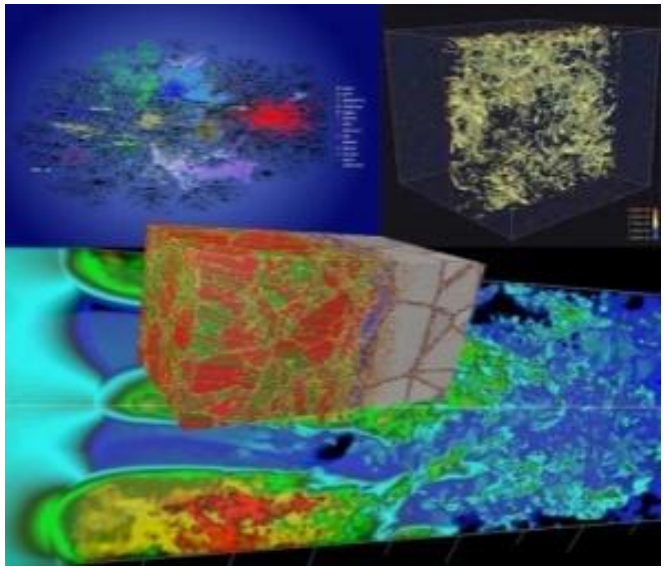


Programming Models at the Exascale

(and programming environments)

Brad Chamberlain, Cray Inc.

Cross-cutting Technologies for Computing at the Exascale

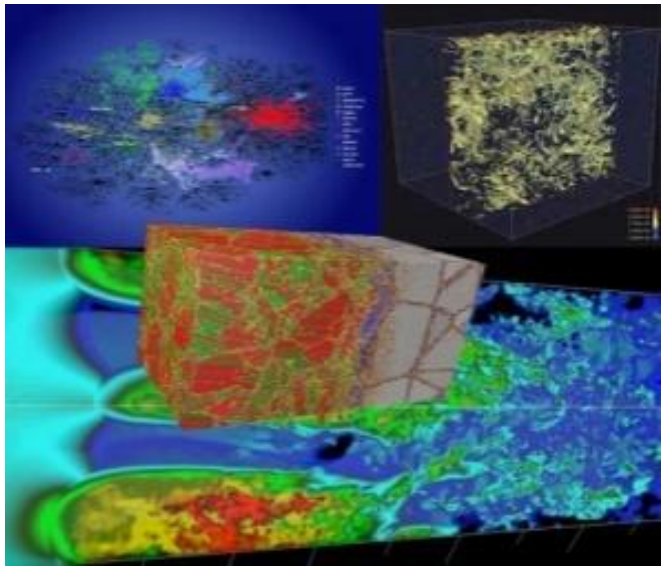
February 2nd 2010

Rockville, MD

The solution to all exascale's woes

Chapel!!!

Brad Chamberlain, Cray Inc.
Chapel technical lead and bigot



Programming Models at the Exascale

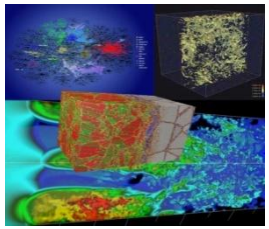
(and programming environments)

Brad Chamberlain, Cray Inc.

Cross-cutting Technologies for Computing at the Exascale

February 2nd 2010

Rockville, MD



Purpose of this Talk

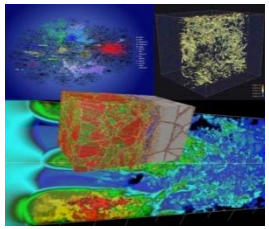
Goal: Summarize programming environment issues for exascale computing to set the stage for breakout sessions

Primary Input:

- December `09 workshop: Architectures and Technology

Additional Inputs:

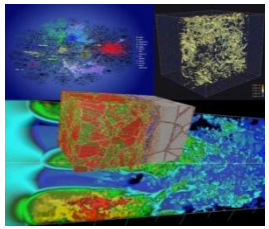
- March `09 workshop: Fusion Energy Sciences
- Exchanges with colleagues (most notably Yelick, Lusk, Sarkar)
- My own thoughts and biases



Definition of Terms

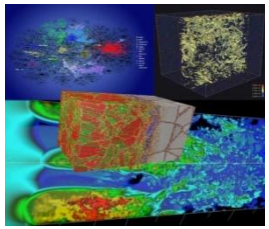
Programming Environment: the things that a user deals with when programming a computer

- *programming models (abstract):* user models for how the machine, execution, memory, communication, etc. will work
- *programming models (concrete):* notations used to write programs: languages, libraries, pragmas/annotations, …
- *tools:* debuggers, performance tools, editors, IDEs, …



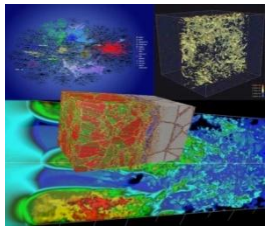
Outline

- ✓ Terms of engagement
- ➢ Exascale hardware: summary and implications
- Selected Timeline
- Programming Models: proposed directions
- Exascale Tools: a brief interlude
- Programming Models: more details
- Some questions



Exascale Hardware in a Nutshell

- *#nodes, network:* no dramatic changes expected
- *system size/complexity:* expected to grow
- *node architecture:* expected to undergo dramatic changes
  - massively parallel
  - multiple processor types
  - multiple memory types, including programmable (scratchpad)
  - generally more heterogeneous/hierarchical than today
- *memory:FLOPS ratio:* expected to get worse



Exascale Hardware Implications I

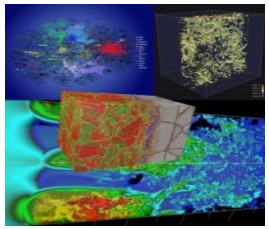
- *#nodes, network:* no dramatic change expected

  ⇒ MPI should continue to be useful on exascale systems

- *system size/complexity:* expected to grow

  ⇒ programs will need to be resilient to failures

  ⇒ tools will need to aggregate and synthesize information to prevent information overload



Exascale Hardware Implications II

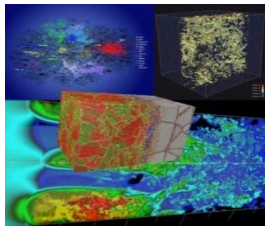
- *node architecture:* expected to undergo dramatic changes
  - massively parallel
  - multiple processor types
  - multiple memory types, including programmable (scratchpad)
  - generally more heterogeneous/hierarchical than today

⇒ we will need new abstract node models

(for programmers as well as for compilers)

⇒ applications will need to generate and manage more parallelism and map it to the appropriate resources

⇒ we will need new programming notations for nodes

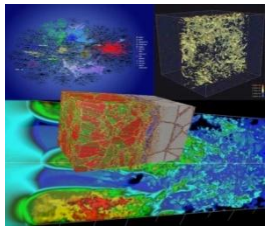


Exascale Hardware Implications III

- *memory:FLOPS ratio:* expected to get worse

  ⇒ will need to be more attentive to how memory is used

  e.g., a renaissance of out-of-core computations at multiple scales?



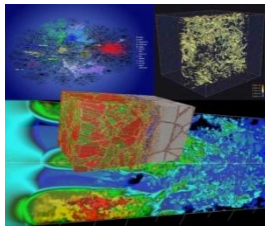
The Big Question

Which of these changes will directly impact the end user?

Or, put another way:

Which of these challenges can additional R&D ameliorate?

(Note that the answers are likely to vary depending on how you're willing to trade off level of control vs. portability vs. performance vs. …)



Anticipated Exascale Timeline (excerpts)

2010-2011: develop abstract node/machine model

2010-2012: initial programming models development

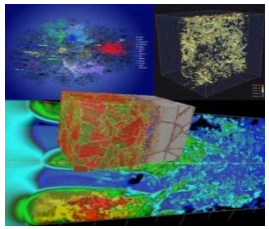
2012-2013: early demonstration of programming models, generating course corrections

2013-2015: continued programming models development

2013-2015: application development in programming models

2015: deployment on 100 petaflop systems

2018: deployment on exaflop systems



Timeline Implications

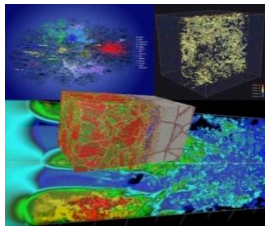
…

2013-2015: application development in programming models

2015: deployment on 100 petaflop systems

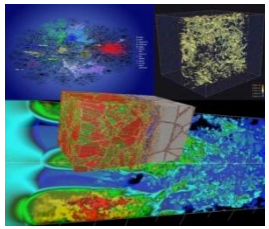
…

⇒ insufficient time to develop new notations from scratch; rather, evolve/extend existing programming models



Expected Characteristics of Exascale Programming Models

- **parallelism:** nested, dynamic, loosely-coupled, data-driven
  (*i.e.*, "post-SPMD" programming/execution models)
  - to take advantage of the architecture
  - to better support load balancing and resilience
- **locality:** concepts for *vertical* control as well as *horizontal*
  (*i.e.*, locality within a node rather than simply between nodes)

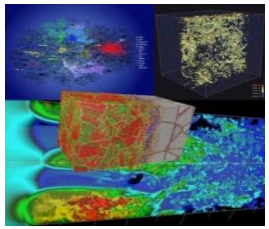


Programming Model Approaches

The December working group proposed investing in two major styles of programming models:

1) *hybrid/evolutionary:* MPI + _____?
   (MPI for inter-node programming, something else for intra-)

2) *unified/holistic:* _____?
   (a single notation for inter- and intra-node programming)

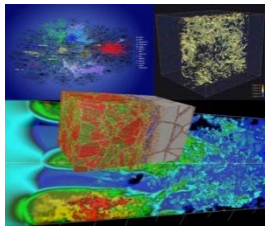


Hybrid Programming Models: MPI

MPI + [intra-node model]

- because #nodes, inter-node concerns not expected to change dramatically
- yet MPI probably still needs to evolve and improve:
  - support for hybrid programming/interoperability
  - better scalability, especially in terms of memory utilization, especially for collectives
  - improved resilience features
  - purer one-sided communication; active messages
  - asynchronous collectives
  - …

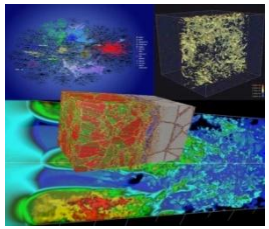
(these efforts are already well underway as part of MPI 3.0)



Hybrid Programming Models: [intra-node]

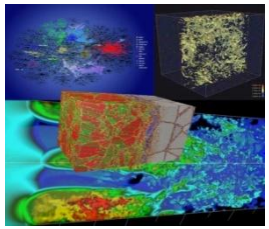
MPI + **[intra-node model]**

- OpenMP
  - would require extensions to support accelerator programming
    - e.g., similar to directives from PGI, CAPS
  - may require the introduction of locality-oriented concepts
  - these efforts are already underway as part of OpenMP 3.0
- PGAS languages
  - already support a notion of locality in a shared namespace
  - UPC/CAF would need to relax strictly SPMD execution model
- Sequoia: supports a strong notion of vertical locality
- CUDA/OpenCL: lower level than ideal for an end user



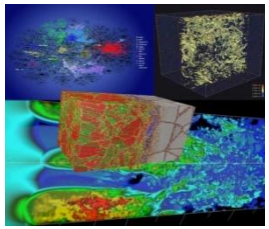
Unified/Holistic Programming Models

- traditional PGAS languages: UPC, CAF, Titanium
  - would likely require extensions to handle nested parallelism, vertical locality
- HPCS languages: Chapel, X10, Fortress(?)
  - designed with locality and post-SPMD parallelism in mind
- other candidates: Charm++, Global Arrays, ParalleX, …



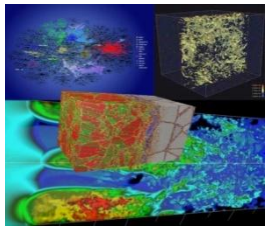
But what about…?

- mainstream multicore/GPU languages: exascale program should track, but not fund (without sufficient promise)
- domain-specific languages:
  - great if they fit your problem, but if they don't, a non-starter
  - exascale program should focus on more general solutions
- functional languages:
  - have never been heavily adopted in mainstream or HPC
  - their abundant parallelism is nice, yes, but…
  - copy-on-write optimization is the dual of alias analysis (?)
- parallel scripting languages: sound attractive to me



Exascale Tools: Debuggers, Perf. Analysis

- Primary challenges:
  - Given massive amounts of parallelism, need aggregation to avoid overwhelming the user with details
  - Given higher-level and/or loosely-coupled programming models, need to report information in the user's terms
- Timeline:
  - **2014:** integration with emerging programming models
- In all honesty…
  - tools have probably not received enough workshop time
  - seems like a good area for innovation (e.g., execution visualizations to understand mapping of code to hardware)



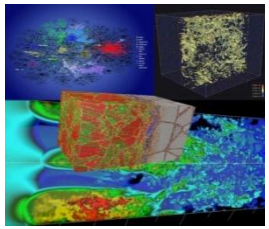
Programming Models Desiderata

- **Interoperability:** need to preserve legacy code even if it isn't mapped optimally to the exascale hardware

  [**ad:** watch for 2010 parallel language interoperability workshop being organized by Tom Epperly/Jim McGraw (LLNL)]

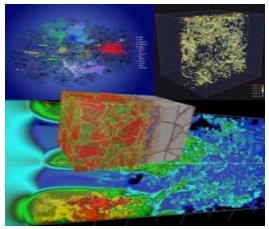
- **Multiresolution design:** ideally, a programmer should be able to program at more abstract or explicit levels within a single language as their needs require/schedule permits
  - support division of labor: science vs. parallel mapping
- **Autotuning:** given the huge parameter space at exascale, compiler assistance to help search it would be useful

These seem like no-brainers to support to me, though R&D is probably required to do them well



Programming Models Challenges

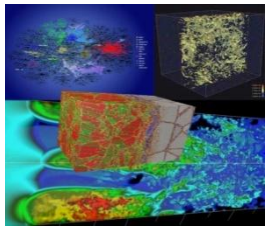
- **Resilience:** How can programming models help?
  - *checkpoint/restart:* may not be sufficient at exascale
  - *redundancy control:*
    - give user dials/abstractions to request redundancy in key areas
    - requires hooks from system software to gracefully catch failures
- **Power Management:**
  - given its impact at exascale, can programming models help?
- **Memory Consistency Models:**
  - we barely understand them now as a community
  - seems they will only get worse?
- **Out-of-Core:** Back-in-style? How can prog. models help?



Metrics for Success

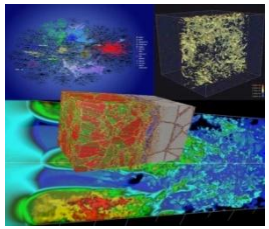
- # of apps that can effectively make use of exascale
- portability of those applications across machines (exascale and non-)
- utilization levels of most precious resources
  - i.e., probably not FLOPS; more likely bandwidths

- 10x productivity?
  - on one hand, it's a catchphrase we've leaned on heavily while still unable to define metrics for it very well, leaving people skeptical
  - on the other hand, it's still a highly desirable concept



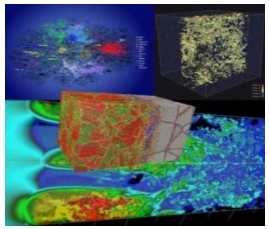
The Adoption Problem

- Most of us prefer not to have to learn new things
  - (though sometimes doing so can be liberating)
- Given that all of the models described here involve changes, how should we best support that effort?
- HPCS languages case study:
  - How to turn skeptical community into believers, esp. given that most opinions are from the gut, not investigation?
  - How to transition from “How will *you* do xyz in Chapel/X10?” to “How will *our community* do xyz in Chapel/X10?”
  - What demonstrations/stepping stones to build confidence?
  - Is dedicated funding for evaluation & study useful/necessary?
  - How to avoid the mistakes of Ada/HPF?



My Questions for apps/algs folks I

- What is your application's dream wishlist for prog. models?
  - i.e., "If only we could…" or "If only we didn't have to…"
- Are SPMD programming models natural for your algorithms or simply what you've been given to work with?
- To what extent do you want explicit control over every detail vs. automation at some cost vs. the ability to vary this choice for different program phases?
- What benchmarks do you look to as stand-ins for your applications? Is there a need for new benchmarks (a la the NPB) for your application area?



My Questions for apps/algs folks II

- What would motivate you to consider new programming models? (Is the carrot or the stick the bigger motivator?)
- Do you currently use out-of-core programming? Why/why not?
- Do you currently use checkpoint/restart (or other resilience techniques)? Why/why not?
- Which parts of this talk made you think “amen”? Which parts made you curse the December workshop attendees?