Exaflops, Petabytes, And Gigathreads… Oh My!

Sriram Swaminarayan

Group Leader, Applied Computer Science

Los Alamos National Laboratory

LA-UR-11-11113

What We've Been Told About Exascale - I

- **Everything is different**
  - …Except MPI between nodes
- **Everything on the node is more parallel**
  - More data parallelism
  - More task parallelism
  - More cores: All cores are not created equal
  - More hierarchy in the memory architecture
  - Less Memory per core
  - Less Memory bandwidth per core
  - Flops are cheap
  - Data movement is expensive
- **Resiliency and fault tolerance will be a big deal**
- **Power will drive design**

Toto, I've a feeling we're not in Kansas any more.

What We've Been Told About Exascale - II

- **Just recompiling will not get you performance**
  - You will have to rethink your algorithms
  - You will have to rewrite your codes
- **You will have to write code using a different programming model and possibly different languages**
  - FORTRAN lives: just take these two pragmas and call me in the morning
  - FORTRAN is dead: pragmas are snake oil
  - C/C++ is the only way to get performance
  - OpenMP is the way to get performance
  - OpenCL is the way to get performance
  - CUDA is the way to get performance
  - Intel Parallel Building Blocks will solve all your on-node concurrency problems (at least on Intel architectures)
  - PGAS languages will solve all your on-node concurrency problems
  - Functional programming will solve all your on-node parallelism problems
- **We can use mini-apps to explore these different characteristics**

Come along, Dorothy. You don't want any of *those* apples

So, You're Telling Me…

- **I should rethink my methods and algorithms**
  - We understand why this is so
- **I should be using a different programming model / language**
  - OK, we can accept this but you haven't told me what the new paradigm is
- **So, I should rewrite my application how many times?**
  - Do I really need to give up FORTRAN?
  - Do I really need to use C/C++?
  - What about this HASKELL thing?
  - Do I use OpenMP?
  - Do I use CUDA?
  - Do I use OpenCL?
  - …

You, my friend, are a victim of disorganized thinking

From An Application Perspective…

- **The promise of exascale is not just bigger simulations**
  - Rather it is more detailed simulations that allow better physics in the same time
  - Unfortunately this will require rethinking and rewriting existing code bases, even for applications that choose to only scale to larger sizes
- **However, we get money to do science, not for rewriting our codes**
- **Application developers shouldn't have to care about the architecture**
  - They need to care about the *abstraction*, but not the architecture itself
- **We should have to care about the language / utilities we use**
- **We don't mind rethinking methods and algorithms**
  - In fact, we enjoy doing that
- **We do mind rewriting for every architecture**
- **We do mind having to rewrite every few years: portability and longevity in codes is extremely important for continuity of research**
- **And, believe it or not, we recognize that the world is changing around us and that something has to change in the way we develop applications**

Now I... I know we're not in Kansas!

What About These Mini-app thingies?

- **Folks agree about mini-apps because they mean different things to different people!**

What About These Mini-app thingies? (Proxy App? Mini App? Skeleton Apps?)

Here is what they mean to us:

- **Full Blown App: 10 Tons**

What About These Mini-app thingies? (Proxy App? Mini App? Skeleton Apps?)

- Full Blown App: 10 Tons

- **Proxy App: 1 Ton**

What About These Mini-app thingies? (Proxy App? Mini App? Skeleton Apps?)

- Full Blown App: 10 Tons

- Proxy App: 1 Ton

- **Mini App: 10 Kg**

What About These Mini-app thingies? (Proxy App? Mini App? Skeleton Apps?)

- Full Blown App: 10 Tons
- Proxy App: 1 Ton
- Mini App: 10 Kg
- **Skeleton App: 100 g**

Language Improvements We'd Like To See…

- **Ease of leveraging underlying hardware**
  - OpenCL, TBB, ArBB, CUDA and OpenMP are baby steps in this direction
- **More data parallel constructs**
- **More thread aware constructs**
- **Fewer pragmas**
- **More abstraction of the hardware**
  …But not so much that there is no performance
- **Ability to control underlying hardware:**
  - Ability to describe hardware characteristics to runtime (or compiler)
  - Ability to tune for resilience
  - Ability to tune for performance
  - Ability to tune for power
  - Ability to tune tune for <your favorite feature here> …
- **A migration path would be nice, but not critical**
  - I'll rewrite my code once, but more than that you're asking too much
  - I'll even rewrite it in a completely alien language (OK, so this is stretching it)

I *do* believe in spooks, I *do* believe in spooks. I do, I do, I do, I *do* believe in spooks, I *do* believe in spooks, I do, I do, I do, I *do*!

Utilities We'd Like To See - I

- **Resource Allocation / Sharing Framework**
  - Simplifies mapping a hierarchical scientific problem to hierarchical hardware
    - Informs application of underlying hardware capabilities
    - Maps application based on hints provided
    - More informed control of memory layout and movement
  - Allows application to specify hardware characteristics for specific sections of code
  - Allows application to specify instruction mix for specific sections of code
  - Simplifies allocation, sharing, and destruction of objects between different tasks
  - Simplifies (or even obviates) thread management
- **Hints Framework: Provide hints to runtime / compiler**
  - Compute intensive sections
  - Memory (pointer chasing) intensive sections
    - Disjoint memory spaces
    - Shared memory spaces
  - Communication intensive sections
  - Accuracy desired (trading off power for accuracy, for example)

First they took my legs off and they threw them over there!
Then they took my chest out and they threw it over there!

Utilities We'd Like To See - II

- ***In-situ* data analysis / visualization framework**
  - Shares resources with other tasks
  - Runs concurrently with other tasks
  - Is platform independent
- **DSL creation utility**

  Allows 'elite' users to create a DSL for others to use
  - DSLs can be light-weight
  - DSL holds all the knowledge for some subset of the science of interest
  - Can perform tasks seamlessly across different platforms
  - Should be inter-operable
  - Should be able to use a common data model across all pieces
  - Should be able to share resources with other pieces
  - Provide an abstraction layer between the system and application developers

Pay no attention to that man behind the curtain

Conclusions: It Will Not Be Easy

From an application perspective, some features that would ease the transition to exascale are:

- **A migration path for existing codes**
  - Perhaps a piece-wise migration for large code bases with large number of users
- **An investment in rethinking methods and algorithms**
- **We need new means of programming that:**
  - Abstracts away some details of the hardware
  - But exposes other features in fine grained details
  - Allows for creation of application specific DSLs / libraries that ease the burden on the vast majority of developers
  - Ability to share resources between different tasks
  - Ability to spawn tasks effortlessly
- **Proxy-apps / Mini-apps / Skeleton-apps are a good way of exploring different algorithms, languages, software paradigms and data models**

Frightened? Child, you're talking to a man who's laughed in the face of death, sneered at doom, and chuckled at catastrophe... I was petrified