

Programming Models Break-Out

2

John Shalf, Kathy Yelick, Marc Snir, Robert Harrison, Josh Fryman, Saman A., Robert Clay, Sue Kelly, Thomas Sterling, John Bell, Jackie Chen, Jim Belak, Stephane Ethier, Lucy Nowell, Sonia Sachs, Bill Harrod

Overarching Questions for Programming Models Breakout

- Map abstractions to application requirements from session 1
- Identify dependencies
- Role for runtime in prog environments

Programming Model

abstractions and their semantics

- Parallelism
- Global name space
- Locality/distribution/memory hierarchy
- Synchronization
- Introspection
- Resilience
- Communication
- Time/space/energy management

Enumerating what the abstractions or implementations are

Notes on Programming Models

- **What abstractions automate tedious things that programmers do in practice to optimize codes or to make them portable across systems**
 - Which programming models/products offer abstractions or features that automate things that are tedious, complex and expensive
 - What does it look like to use them?

**Enumerating what the abstractions DO for programmers
(what low-level things are they doing that they shouldn't be?)**

How can a programming model help?

Reformulating the Questions around *abstractions to automate tedious or complex tasks*

- **Data layout** : Domain-Decomp/Tiling/cache-blocking, data placement (*Raja, TiDA Kokkos*)
- **Thread Affinity**: (thread pinning, associate with data and with computing elements)
- **Threading** (Task, SPMD, DAG dependencies)
- **Memory spaces** (virtualize or not, different kinds, manual copying between)
- **Communication** (lightweight, MPI+x or all threads peers)
- **Handling nonuniformity** (async, tasking, runtimes)
- **Resilience**: (user-driven vs. system driven)
- **Modularity**: (kernel/driver model in frameworks)
- **Data Storage Model**: break assumption of contiguity? What abstraction to store data this way (post-posix/object storage models`?)

Reformulating the Questions around *abstractions to automate tedious or complex tasks*

- **Data layout** : Domain-Decomp/Tiling/cache-blocking, data placement (Raja, TiDA, Kokkos) (*can be related to memory spaces and thread affinity*)
 - *What is architecture independent*
 - *What is the binding to architecture specific features?*
- **Thread Affinity**: Associating threads to data locations & processor or memory location (thread affinity / Barry's thread communicator??)
- **Can we express enough parallelism (how): Task vs. data (what else)**
- **Memory spaces**: (*can be related to communication*)
 - (OMP target or X10/UPC++ places)
 - Memkind (explicit malloc and memcpy())
- **Communication**
 - can we make it implicit for some examples?
 - Should we make communication implicit instead of packing things into buffers?
 - Encapsulation? (we do between nodes, but don't encapsulate within the node)
 - Reduced latency and overhead?
- **Handling nonuniformity (async, tasking, runtimes)**
 - Async (Background data transfers)
 - and load balancing (domain specific or not?)
 - Separate schedule from algorithm description? (e.g. Halide vs. dynamic DAG-sched runtimes)
- **Resilience**:
 - GVR (making arrays permanent or not)
 - Containment Domains (application specific resilience)
- **Modularity**: Kernel/Driver separation (frameworks... is there a programming model here? Is there a programming system play here, or is it frameworks and software engineering?)
- **Object Storage Model**: break assumption of contiguity? What abstraction to store data this way (post-posix?)

Follow-up Questions

- **How do we evaluate these options?**
 - **What does it look like to use these programming abstractions in the different pmodels?**
 - **How does it perform?**
 - Could it perform better if the implementation were better? (or is it the compute/data-movement complexity)
 - **How invasive?**
 - **Does it interoperate with other pmodels?**
 - Engineering problem that has been around 10 yrs
 - We want a kitchen sink programming environment (do we?)
 - Need rich ecosystem around it (composability)
 - Interoperability between levels is essential
 - **Is it maintainable?**
 - How much code to maintain (1k lines or 1M lines?)
 - who else would maintain the implementation?

What actually happened

Application concerns P1

- **Stephane Ethier: (GTC)**
 - what if my codes doesn't work at all
 - What is required to make them run
 - Will my algorithms work
 - What if I have to rewrite my entire code?
- **Robert Harrison: (Materials/Chem/TCE)**
 - *Interoperability*: tools don't work together (BLAS, OpenMP); it's an engineering problem, but nothing solves it
 - *Composability*: can actually interact, not just side by side
 - We want a kitchen sink programming environment (*do we?*)
 - *Tools*: big infrastructure needed to move
 - Need to build an ecosystem around it
- **Jim Belak (ExMatEx)**
 - Would like to feel warm and fuzzy that there is a mapping from HL to LL
 - Hierarchy to go down to the lower level
 - Want to be able to drill down (recurring theme... opaque SW stacks are bad)

Application concerns P2

- **Jackie Chen (ExaCT)**
 - Data-centric task-based (promising initial results from Legion)
 - More work on how the application-specific information use used to map onto different parts of the memory hierarchy, etc.
 - Information gained either through machine learning (but initially
- **Rob Hoekstra (ASC Codesign)**
 - Abstract away what you know how to automate
 - Would like something where we start out with the infrastructures and insert information into the framework
- **John Bell (ExaCT)**
 - Interested in block structured AMR
 - We have a distributed collection of boxes
 - Would like to think of them boxes over which I sweep, but
 - Perhaps do some latency hiding as well
 - How do we do this if the individual nodes are very heterogeneous
 - NUMA effects and remeshing
 - MPI will be the right answer? Need something lighter weight?
 - We have large libraries and if the communication doesn't work we can jack it up and use something different

What separates us from exascale computing?

- **Thomas:** How to expose parallelism
 - How to deal with latency?
 - How does the programming model interrelate to the expectation of the underlying runtime system
 - Explicit, direct control over resources vs virtualization
 - Runtime system is more than just hiding information about hardware, it also has information not available to the program to make the right choices
- **Josh Fryman**
 - Layers of hierarchy from the application scientists to low level
 - Decomposing into ranks
 - Should you, as an application scientist be expressing my parallelism
 - Additional technologies to bridge the gap
- **Robert:** In the what are the thing you're worried about question:
 - Do we give people abstractions?
 - Or expose all parallelism and have programming system map it
 - Madness and NWChem: you get ossified into it fairly quickly
 - Express all available parallelism and how to manipulate it?

MPI Discussion

- **Bill Harrod**
 - For a long time we've repeatedly heard that MPI (the ½ dozen functions that most people use) will not result in optimal coding in an exascale system
 - Will it work efficiently: then what must the new model look like?
 - If you end up back where you started, that is OK
- **Josh Fryman (Intel)**
 - Will MPI eventually be modified is not the question? How can I write a layer than is not brittleness
- **New Stephane**
 - Partly disagree with MPI as a low level implementation
 - We have our MPI encapsulated so our application programmer doesn't do ghost zone exchange (*indicates that it is, but encapsulation important*)
 - If we have to go to a model where ghost zone exchange
- **Sriram**
 - Flat on Hopper, we don't have to have to worry about tasks
 - Global work stealing: 90%
 - Got it working on MPI; got it working
 - MPI isn't the issue
- **John Shalf**
 - Does every thread communicate or does only 1 thread communicate
 - Do we have many threads under 1 rank, or every thread is a peer (*regardless of whether it is MPI or PGAS, it will affect communication encapsulation*)

Grappling with Abstraction

- **Jim Belak**
 - I've exposed the application in a way that separates concerns (Is this classical software engineering or not?)
 - The machine abstractions have change in the long term: Spiral is an example of that: express things at a high level and let prog system do the mapping
- **Saman**
 - Highest level is something like math equations
 - Parallelizable algorithms
- **Josh**
 - Need to decouple data layout and compute
 - Otherwise too brittle
- **Marc Snir:**
 - Kokos and Raja are very nice things, but are programming in the small? ... Not exascale programming per se
 - Not programming in the large: just at the node
- **Robert Clay**
 - If we don't make it easier, we are basically screwed
 - Kokkos is a direct response to that
- **Sue**
 - Does the application programmer have an abstract machine model in your head (Bell: yes)
 - Do you worry about the number of cores per node (Bell and others: no)

List of Ideas from Marc Snir

- How to split data
- How to split computation
- How to map it to the machine?
- How do they relate to each other?
- Internode and intranode
- Domain-specific approach is one way to make it more tractable
- Higher level vs. low-level constructs

Grappling with Abstraction Levels p2

- **High level abstractions**
 - Concurrency
 - Locality
 - Dependency
 - Algorithmic choice
 - Algebraic properties
 - Correctness
 - Guidance about the shape of data (high level data layout abstractions)
- **Next level down**
 - Mapping data and tasks to AMM (*still might be directed by expert programmer, but would imagine future with ML to make decisions*)
- **Machine/Low-level**
 - Mapping from AMM to hardware
- **Automatic system cannot infer these mappings from end-to-end (needs hints)**
- **If something is very expensive, you need to expose it at the high level**
 - Performance transparency and performance awareness
 - Need control when needed

The Importance of Codesign

(we have less difficulty talking to each other about these kinds of topics)

- **Changing Hardware is very expensive and takes a long lead time**
- **Changing Software (rewriting our codes) is very expensive and takes a long lead time**
- **Quantitative information about those cost trade-offs to enables rational and thoughtful decision making for both code teams and for our industry partners who will be developing the machines that run those codes. *(risk mitigation for expensive decisions... parroting about machine futures backed by analysis of the trade-offs)***

Extra

Concerns from Scientists/Apps dev.

- Will I need to rewrite my code
- Concerned that flat MPI will not work efficiently
 - What must the new model look like?
- Levels of hierarchy between application scientist and low level areas
 - How much brittleness are we exposing
 - How can I express without brittleness at the top level (how do I get away from injecting brittleness)

What is absolutely necessary?

- Virtualize services within the node
- Need load balancing in the node level just to handle large variants
 - But might be pre-supposing a particular machine behavior

- Data Layout
 - Last thing we want is apps to code to runtime API
 - Domain decomposition (automate the mapping). This should be ultra-simple to do. What depth of ghosting do you want.
 - Migration of work for load balancing
- Kathy's points (
 - Abstractions for people to code to
 - Or express all available parallelism and people code to that thing (abstractions to code this hierarchically)
 - Josh's points brittleness at each level of abstraction

- Lower overheads
 - Will be critical for strong-scaling limit
- What can the programming model do to reduce overheads
 - Lightweight communication
 - Lower-overhead task scheduling
 - Reduced synchronization
- What is the high level approach to hide latency
 - Use non-blocking operations
 - Use threads (overdecomposition)
 - Reducing latency by moving work to data
- Not express parallelism at

- High level abstractions
 - Concurrency
 - Locality
 - Dependency
 - Algorithmic choice
 - Algebraic properties
 - Correctness
 - Guidance about the shape of data (high level data layout abstractions)
- Next level down
 - Mapping data and tasks to AMM < still might be directed by expert programmer, but would imagine future with ML to make decisions >
- Machine/Low-level
 - Mapping from AMM to hardware
- Automatic system cannot infer these
- If something is very expensive, you need to expose it at the high level
 - Performance transparency and performance awareness
 - Need control when needed