Next Generation Programming Models

Kathryn O'Brien IBM Research

The Programming Model Challenge

IBM Research

Architectural complexity impacts many aspects of the system software stack and execution environment:

OS and runtime implications
Control Systems, Resource managers, schedulers
Tools – performance monitors, visualization
Debuggers,
. . .

Architectural complexity impact is expected to increase significantly by 2022

Memory attributes
Data management
Resiliency
. . .

The programming model is the interface between all that complexity and the application developer . . .

What the Programming Model Provides

IBM Research

- A programming model is a common conceptual framework to abstract the underlying hardware complexity
  - Used by application developers, algorithm designers, compiler-writers, runtime developers, tool builders to communicate with each other and write code.
  - A **good** programming model should be **architecture agnostic** and enable high productivity and **portability**
    - Programming model **implementation** should provide architectural differentiation and **performance**
- May be realized through one or more of:
  - Libraries
  - Language/compiler extensions – pragmas, directives
  - New languages
- Different programming models may co-exist at different levels of abstraction.
- A good programming model *can* lead to new industry-wide eco-systems

Programming Model Constraints

IBM Research

- Address a broad range of programmer expertise
  - Low level programming expertise – **performance** is paramount
  - Application expertise – rapid deployment, **productivity and portability** are key
- Provide high level abstractions with ‘breakout’ mechanisms for critical performance paths
- Provide a migration path for Legacy codes and cross-platform performance portability
- Support a range of implementation paths
  - Libraries
  - Language/compiler extensions – pragmas, directives
  - New languages
- Barriers to adoption

The Application Programmers Dilemna

IBM Research

Highest performance with programmer control

Sequential languages with explicit threading

Manual Intervention

Manual techniques: unrolling, reversal etc

performance at expense of readability …

Parallel languages, pragmas and libraries

Semi-automatic

Higher level pragmas and directives preserve readability of the code to some extent

Automatic Optimization

Compiler supplied

Automatic echniques for:

- Auto-parallelization
- Locality optimizations
- DMA optimization
- Speculative parallelization
- Helper Threads
- Dynamic techniques

User code is left unchanged

Highest productivity with automatic compiler technology

Exascale Programming Models: … thoughts from 2009/10

IBM Research

- For 2015 – new models not feasible in the timeframe
  - Abstract machine model is changing - Major focus needs to be on intra-node
    - HPC programming models have tended to follow rather than lead in the area of GPU technology
  - Inter-node – MPI is likely to be good enough
    - Unified models another option – but need hardware support for global address space
  - Interaction of programming model and RAS will be very important
  - More focus on asynchronous design
    - will enable applications to be more resilient, latency tolerant and more resistant to impact of jitter in large systems

- Invest in a range of programming models
  - Monitor evolving models beyond exascale community: CUDA, OpenCL, TBB …
  - Evolve established hybrid : MPI + OpenMP, Pthreads,
  - Develop new hybrid: MPI + PGAS ??
  - Holistic models: CAF, UPC, HPCS,
  - Revolutionary approaches - new languages not a good idea - unlikely that revolution will happen …

- Consensus to pursue three technologies:
  - Well defined abstract machine model and open runtime layer
  - Multiple diverse node level models with MPI internode
  - Tools

(Mostly) Mainstream Programming Models (Node Level)

IBM Research

Prog Model	Example Implementation	Types of Parallelism	Memory Model	Accelerator affinity	Adoption
PGAS	**UPC, CAF Global Arrays**	Data	Single Global Address Space	Not easily	Low
	Chapel	Task and Data Parallelism. Nested Parallelism	Abstractions for data distribution and data driven placement of computations	Yes	Low
APGAS	**X10**	Task and Data		Yes	Low - Moderate
Heterogeneous	**CUDA, OpenCL OpenACC**	Task and Data	Separate memories but evolving – with UVM	Yes	High in the GPU arena
	HAS/C++AMP			Yes	Low – limited domain
Shared Memory	**OpenMP Pragmas**	Task and Data	SMP but separate memories with OpenMP4.0 forward	Yes	High
	CILK – Lang extensions	Task and Data		Potentially if added keywords	Low

Research Programming Models (X-stack and Others)

IBM Research

Project	Languages/Runtimes	Types of Parallelism	Memory /Data Representation	Accelerator affinity	Motivation
DEGAS	Extensions to Fortran(CAF 2.0), C (Habanero C and UPC) and Python	Tasks and Asynch tasks	APGAS Distributed Hierarchical Place Trees	No	Hierarchical Programming concepts for Locality and Parallelism
D-TEC	Will be constructed from DSLs with & without syntax extensions to current languages such as C/ C++ and Fortran.				Defining approaches to constructing DSLs to bury the HW complexities os future Exascale systems.
Traleika	R-Stream Opts SWARM runtime	Data Flow inspired. Event Driven HW Threads	HTAs	Yes	Adheres to a separation of concerns – separating domain specification from HW mapping
XPI/ Parallex	Source to source for high level languages Lower level parallel programming syntax, implemented through libraries, look and feel of MPI		Intrinsics to encapsulate logically local data and tasks; Adaptive locality semantics that establish relative associations (data-data, data-actions, and actions-actions).		Syntactic representation of the Parallex execution model
X-Tune	OpenMP, CUDA etc performance tool				Expert programmer and compiler collaborate to tune code - encapsulated in domain specific tools for less sophisticated users
Phalanx	Extends C/C++ with explicit constructs for parallelism and locality (library implementation where possible)	Tasks composed of hierarchical thread arrays Asynchronous Task Graphs	Global Address	Yes	Unified model for heterogeneous parallel machines Heterogeneity is explicit and exposed in the type system

A Selection of Activities that didn't make it to the spreadsheet

IBM Research

- StarSS
  - Dependency Aware Task Based Model
  - runtime system limits scalability (strong dependence on the task execution time)
- Scala – more targeted at the commercial domain
- HybridFortran – Weather Prediction Model – Fortran, Not happy with OpenACC functionality
- KernelGen – Focus on filling the gaps in OpenACC
- Several other directive-based, GPU programming models have been proposed, such as OpenMPC, hiCUDA,HMPP,R-Stream, etc.
  - current high-level GPU programming models may need to be further extended to support an alternative way to express GPU- specific programming model and memory model to achieve the best performance in some applications.

Current Programming Model Summary

IBM Research

- Landscape is huge
  - Vendor driven
    - Exploit proprietary HW features
    - Can create ‘lock-in’
  - Standards Driven
    - Create a unified approach that benefits end users
  - Research oriented
    - Govt funded
    - University driven
- Few new models have captured the field in the last 10 years
  - Many have fallen by the wayside
- Established hybrid models continue to have the most traction

The New Mission Statement of OpenMP

IBM Research

- OpenMP's new mission statement
  - "Standardize directive-based multi-language high-level parallelism that is performant, productive and portable"
  - Updated from
    - "Standardize and unify shared memory, thread-level parallelism for HPC"

OpenMP 4.0 features for Exascale

IBM Research

- **Heterogeneous architecture**: CPU with GPU, APU, DSP, co-processors, embedded processors
  - Address heterogeneity challenge of exascale: computing using accelerators
  - Exascale machines: lower energy , higher abstraction
- **Improve locality:** Handle nested parallelism: more control with thread affinity
  - more user input on how to map computation to threads
    - currently: no affinity support provided by user
  - a new thread-affinity to OpenMP standard committee
  - contributed reference implementation in research runtime
- **Improve Tools interface:** Provide timing feedback, Debugging API
  - user want to know where is the time spent
  - But with little or no overheads
  - Have a uniform way of debugging

Accelerator Execution Model

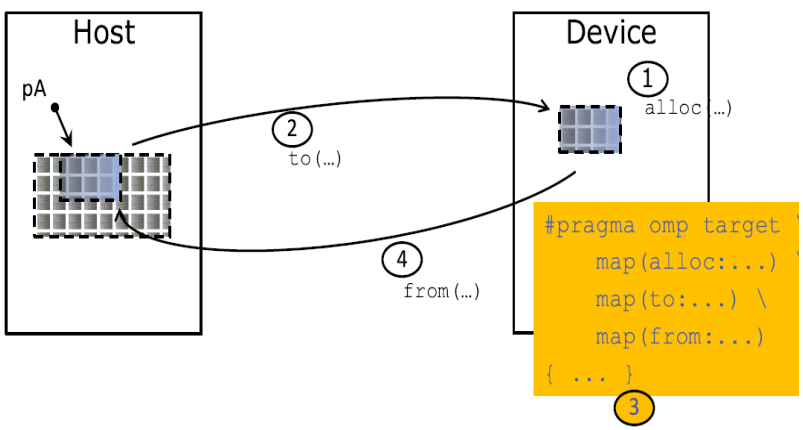
IBM Research

- The `target construct` transfers the control flow to the target device
  - → The transfer clauses control direction of data flow
  - → Array notation is used to describe array length
- The `target data` construct creates a scoped device data environment
  - → The transfer clauses control direction of data flow
  - → The device data environment is valid through the lifetime of the target data region
- Use `target update` to request data transfers from within a target data region

Execution Model and Data Environment

IBM Research

- Data Environment is lexically scoped
  - Data environment is destroyed at closing curly braces
  - Allocated buffers are automatically released
- Can create an environment to keep data on device
  - Avoid frequent transfers /overlap computation/communication
  - Pre-allocate temporary buffers




```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N]) map(from:res)
  {
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    do_some_other_stuff_on_host();

#pragma omp target device(0)
#pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
      res += final_computation(tmp[i], i)
  }
```

host target host target host

Supporting Thread and Data Affinity

IBM Research

Thread Affinity

- User often knows what he/she wants at a high level
  - get threads on separate cores to get more L1/ L2 caches
  - get threads collocated on same core to maximize cache reuse
- Current runtimes have a fixed policy
  - runtime tries to even out load balance across the machine
  - this works well for single level of parallelism,
  - not as well for nested parallelism
- Want to allow users to specify where to get threads
  - broad policies that cover most cases
- Want to allow users to specify where threads are allowed to migrate
  - for load balancing purpose

Places

- OpenMP runtime only deals with (abstract) places
  - there are one or more processors / hardware threads per place
  - runtime may move OpenMP threads within a place
  - runtime should not move OpenMP threads between places
- When executing a parallel region construct
  - affinity directives can impact which OpenMP threads are selected
  - i.e. on which place to pick OpenMP threads
- User is not required to give a list of places
  - there is a vendor-specific default list of places
- User can specify a list of places
  - using high-level names (e.g. threads, cores, sockets) or low-level (numbers)

In 2011, a typical machine could have the following flops

- ~90% GFLOP GPU

- To program the GPU, you could use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP

In 2011, a typical machine could have the following flops

- ~90% GFLOP GPU+ ~7%GFLOP AVX

- To program the GPU, you could use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP
- To program the vector unit, you could use Intrinsics, OpenCL, or auto-vectorization

In 2011, a typical machine could have the following flops

- ~90% GFLOP GPU+ ~7%GFLOP AVX+ ~3%GFLOP 4 cores

- To program the GPU, you may use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP
- To program the vector unit, you may use Intrinsics, OpenCL, or auto-vectorization
- To program the CPU, you might use C/C++11, OpenMP, TBB, Cilk, MS Async/then continuation, Apple GCD, Google executors

In 2014, you can program a typical machine as follows…

- ~90%GFLOP GPU+ ~7%GFLOP AVX+ ~3%GFLOP 4 cores

- To program the GPU, you could use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP, **OpenMP**
- To program the vector unit, you could use Intrinsics, OpenCL, or auto-vectorization, **OpenMP**
- To program the CPU, you might use C/C++11/14, **OpenMP,** TBB, Cilk, MS Async/then continuation, Apple GCD, Google executors

Exascale Thoughts Looking Forward Beyond 2017

IBM Research

- Exascale systems must also address mainstream
  - Broad range of users and skills
  - Range of computation requirements: dense compute, viz, etc
- Applications will be workflow driven
  - Workflows involve many different applications
  - Complex mix of algorithms,
  - Strong Capability Requirements
  - UQ, Sensitivity Analysis, Validation and Verification elements
  - Usability / Consumability
- Unique large scale attributes must also be addressed
  - Reliability
  - Energy efficiency

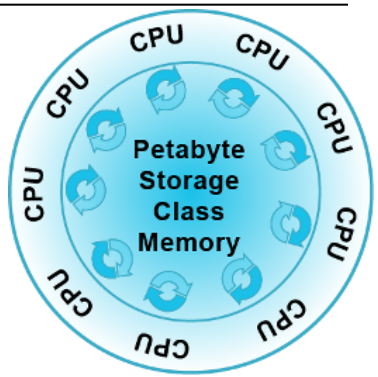
Underlying data scales pose significant challenge which complicates systems requirements

- ***Convergence of analytics, modeling, simulation, data management***
- ***Drives to data centric design***

Role of the programming model in future systems

IBM Research

- Recent programming models focus is on node level complexity
- Computation and control are the primary drivers
  - Data and communication are secondary considerations
  - Little progress on system wide programming models
    - MPI, SHMEM, PGAS . . .
- Future, data-centric systems will be workflow driven, with computation occurring at different levels of the memory and storage hierarchy
  - new and challenging problems for software and application developers
- Programming models must evolve to encompass all aspects of the data management, and computation requiring a data-centric programming abstraction where
  - Compute, data and communication are equal partners
  - Outer level abstraction will provide user means to reason about data and associated memory attributes
  - Will likely co-exist with lower level programming models

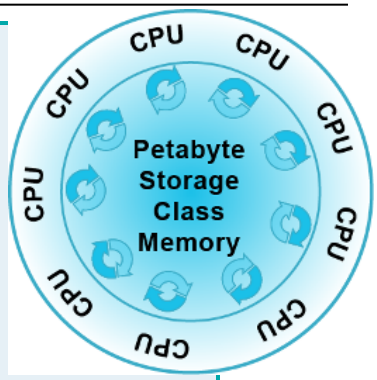


Data-Centric Model

Some strategic directions – but with pragmatic choices

IBM Research

- Refine the node-level model and extend to support system wide elements
- Separate programming model concerns from language concerns
- Revisit holistic language approach – build on the 'endurance' of HPCS languages and evolve the concepts developed there:
  - Places, Asynchrony ..
- Investigate programming model concepts that are applicable across mainstream and HPC
- Focus on delivery through existing languages:
  - Libraries, runtimes, compilers and tools
- Strive for cross vendor support



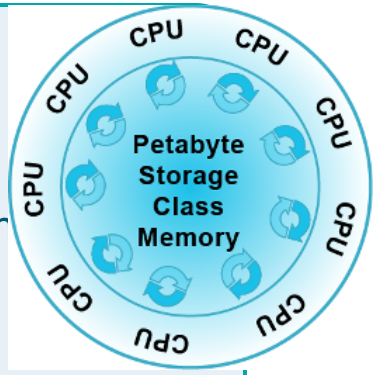
Data-Centric Model

Pursue open collaborative efforts – but in the context of **real** system development

Thoughts on a System Wide Programming Abstraction

IBM Research

- **Provide a uniform conceptual framework**
  - Build on global active objects
  - extend with elements of the APGAS programming model, to support data-centric computation on node types with differing compute, memory ar communication attributes
  - Provide abstractions for range of hardware elements.
- **Architectural elements surfaced as "places"**
  - Support user-defined computation
  - Have associated mutable meta-data at run-time
  - Support reflective access, so that tools such as schedulers and data mappers can operate on them.
- **Common place abstraction**
  - All places support the same fundamental set of properties
  - Can be thought of by the programmer in fundamentally the same way
  - Important practical differences are hidden (e.g. in intensity of compute vs memory).
- **Ease of movement between places**
  - Ability to move code (and associated working data) within a computation, and crucially, *between* computations.
- **Architectural element addressability**
  - The execution model lays out the lowest programmable layer of the system



Data-Centric Model

Conclusions

- Focus will continue to be on higher level, industry standard approaches targeting the high level programmer
  - Library support will be paramount
- Goal is to support multiple interoperable Programming Models/Languages across a range of hybrid system capabilities
- Common Open Runtime will be key to success of interoperability
  - Vendor participation essential
- Will need strong evidence of performance and productivity advances to effect a widespread adoption and enable a large shift in compilers and tools infrastructure

Work needs to start now …

Thank you