Adjusting to Exascale Computing

Do Domain-Specific Languages Stand a Chance?

Patrick McCormick

November 18, 2013
Overview

- Requirements
- Perspectives: Perceptions & Expectations
- Challenges
- Examples
- Conclusion

Work supported by: DOE Advanced Scientific Computing Research
Program Managers: Karen Pao, Lucy Nowell
Architecture Design Factors are Impacting Programming Models

The Old Model

- **Costs**: FLOPS
- **Parallelism**: By adding nodes
- **Memory**: maintain byte per flop capacity and bandwidth
- **Locality**: Uniform costs within node & between nodes
- **Uniformity**

The New Model

- **Costs**: data movement
- **Parallelism**: exponential growth within chips
- **Memory**: Compute growing 2x faster than capacity or bandwidth
- **Locality**: Must reason about data locality in increasingly complex memory hierarchies…
- **Heterogeneity**

Source: J. Shalf (2013)
The Exascale Paradox: Programming Model Requirements

- minimal impact on existing codes
- maintainability
- interoperability
- productivity
- performance portability

“I want you to find a bold and innovative way to do everything exactly the same way it’s been done for 25 years.”
The Exascale Paradox: Programming Model Realities

- **Data movement**
  - Must move away from bulk-synchronous
  - Prefer finding “data independence”

- **Parallelism + Concurrency**
  - Memory scaling, utilize core counts
  - Asynchronous data movement -- overlap compute and communication

- “Manual” task scheduling, asynchronous data movement, locality decisions will be difficult at best
  - Too complex, potential portability issues
The Exascale Paradox: Programming Model Requirements

- minimal impact on existing codes
- maintainability
- interoperability
- productivity
- performance

Thanks – this matches DSLs!!!
Where are DSLs Best Suited?

- Where complexity abounds and additional knowledge can be significantly leveraged to ease the burden, increase performance

- Small, limited domain within an application
  - Minimal code impact, reduced complexity and maintenance

- More significant / majority of code base
  - Most benefit at the cost of complexity and cost
  - High risk, high reward
What are we facing? What are the perceptions and expectations?
The Hype Curve...

- Given the history in HPC, it is often impossible to get expectations “high”
- It is very easy to follow the slope well past “disappointed”
- Reaching “enlightened” and maintaining “productive” are difficult ($$$)

“Mastering the Hype Cycle: How to Choose the Right Innovation at the Right Time” by J. Fenn and M. Raskino
Adoption and Achieving Productivity

- Long-term success needs much more attention and innovation
- Domain-nature must flow through the *entire* toolchain
  - Source-to-source is good for prototyping and "small" problems but loss of information/abstraction is painful
  - This means debuggers, profilers, etc.
- Much more complex than just a "new" language
  - Supporting runtime infrastructure(s) are significant for full-system/featured solutions
  - We need better tools to build DSLs
The Adoption (or Rejection) by Embarrassment Principle
Interoperability

- With *existing code base*
  - We can’t afford to rewrite legacy code and libraries

- With other DSLs
  - Complex (e.g. multi-physics) applications might have (need/benefit from) multiple abstractions/models
What to do about Legacy Codes?

- Grin and bear it… Regardless of the path taken, things will need to change…
- We need a **progressive** migration path
  - Allow gradual adoption and transformation (likely with an impact on performance)
- Long-term support
  - This often rests in the hands of the application…
Singe: A DSL Compiler for Combustion Chemistry

- Based on chemical mechanisms which consist of a set of reactions and the species involved (CHEMKIN Standard)
- Challenges: Traditional data-parallel approach in CUDA suffers from spilling, low occupancy, under-utilization of math units, large number of temporaries, memory divergence and shared memory bank conflicts.
- Warp specialization code-generation not directly supported by CUDA – inline PTX code had to be generated

Mike Bauer, Stanford University
Supporting Runtime Infrastructure

- Specify an abstract data representation, the code that operates on them, and their privileges (read-only, read-write, and reduce) and coherence (e.g., exclusive access and atomic)
- Separately implement how the data and tasks are placed and migrated within the system
  - “Mapping” can be done in an application and/or architecture centric fashion

Transforming S3D from Bulk-Synchronous to Data-Independence

- Total tasks/kernels: 781 (44,517 system-wide)
  - Max task tree depth: 4
  - Max task-level parallelism: 57 (widest the task dependence graph gets)
  - Total data fields: 1,140
Interoperability

Legion Runtime + Mapping Interface

GASNET+IBVERBS

IntFC

MEM

MPI

Fortran Application

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

November 2013 | 17
Domain-Centric Focus is Critical

Industry

Runtime & System Software

DSL (Prog. Models) Domain Science Tools Algorithms Architecture
Legion Web Site

- See [http://legion.stanford.edu](http://legion.stanford.edu) for documentation and open-source download (from github).
Thank you

Questions?