Quantifying the Overhead of Today’s Execution Models
Using Benchmarking to Understand Advanced Architecture and Execution Models
John Shalf, Robert Lucas, Pedro Diniz, Nicholas Wright, Jacqueline Chame, Gene Wagenbrenth
{jshire@lbl.gov, rflucas@isi.edu, pedro@isi.edu, NJWright@lbl.gov, jchame@isi.edu, genew@isi.edu}

Introduction / Motivation
Today we have a bulk synchronous, distributed memory, communicating sequential processes (CSP) based execution model
• We’ve evolved it over two decades
• It will require a lot of work to carry it forward to exascale
• The characteristics of today’s execution model are mis-aligned with emerging hardware trends of the coming decade

Emerging Hierarchical Machine Architectures
Need to examine alternative execution models for exascale
• Alternatives exist, and they look promising (e.g. dynamic / asynchronous Execution Models)
• We can use modeling and simulation to evaluate the alternatives using DOE applications
• This can guide our hardware/software trade-offs in the co-design process, and expands options for creating more effective machines

Examples of parallel execution models

CoDesign for Execution Models
• Execution Models are an integral part of the hardware CoDesign Cycle
• Different simulation methods have differing degrees of fidelity, coverage, and performance.
• Combined together, the the Top-Down (model-based) and Bottom-Up (simulator based) methodologies enables full coverage of design space
• Each tool is subjected to Verification and Validation to ensure confidence in its predictive fidelity

GTC
GTC (developed by PPPL) uses PIC method to simulate plasma microfluidic for fusion devices
• Scalable to thousands of processors
• Written in F90 with MPI
Example code provided by Stephane Ethier (PPPL)

• Particles are deposited on grid using multi-point gyro-averaging to represent path of charged particles in a magnetic field
• Accesses of grid point to deposit particle charge can create challenging (nearly random) access patterns for memory
• Parallel updates to mesh points create challenging data hazards

Analysis Tools
Integrated Performance Monitoring (IPM)
IPM is a tool for performing lightweight measurement of the MPI communication characteristics of an application.

Node Level Analysis
Source Code Analysis Tool using Open64 Infrastructure
• Translates code to Intermediate Representation (IR WHIRL)
• Leverages code transformations and architecture features
• Analyzes computation’s critical-path and performance bounds for specific architecture resources
• Relates metrics at source code level (not load/stores instructions)

Communication Analysis

• Load imbalances are natural consequence of static scheduling
• Introspective dynamic scheduling solves problem implicitly

Memory Trace Analysis

Cost of repacking data is significant fraction of the execution time
Waste of resources as well as detrimental to programmer productivity
Example: By using OpenMP tasking we can use spare resources to repack buffers while messages are being sent

Conclusion
In today’s CSP execution model
• There is overhead in accessing data structures
• The memory hierarchy is not well leveraged
• Users are burdened with management of communication and data movement
In practice
• Developers shufle data around
• Petascale programmers tend to be better at optimizing communication than computation
• Compiler inserts “busy” work

Next Steps
• Evaluate CoDesign Center Codes (LMC & NEK)