Deep Dive Meeting
February 3-4, 2015
Current Attendee List:

Bob Voigt    NNSA HQ    rvoigt@krellinst.org
Matt Bement   LANL    bement@lanl.gov
David Daniel    LANL    ddd@lanl.gov
Dave Nystrom   LANL    wdn@lanl.gov
Maya Gokhale   LLNL    maya@llnl.gov
Martin Schulz    LLNL    schulzm@llnl.gov
Jim Ang       SNL    jaang@sandia.gov
Arun Rodrigues   SNL    afrodir@sandia.gov
Jeremy Wilke   SNL    jjwilke@sandia.gov
S. Balachandar “Bala”   University of Florida    bala1s@ufl.edu
Alan George  University of Florida    george@hcs.ufl.edu
Rafi Hafika   University of Florida    hafika@ufl.edu
Herman Lam    University of Florida    hlam@ufl.edu
Sanjay Ranka   University of Florida    ranka@cise.ufl.edu
Greg Stitt    University of Florida    gstitt@ece.ufl.edu
Tom Jackson   University of Florida    tlj@ufl.edu
Tania Banerjee   University of Florida    tmishra@cise.ufl.edu

University of Florida Students:
Dylan Rudolph    rudolph@hcs.ufl.edu
Nalini Kumar    nkumar@hcs.ufl.edu
Carlo Pascoe    carlo.pascoe@gmail.com    (poppyc@ufl.edu)
Kasim AlliKasim   kasimall490@yahoo.com
Chris Hajas    chrishajas@ufl.edu
Mohammed Gadou
Michael Retherford

NOTE: There will be a $125.00 registration fee per person to cover all expenses associated with the meeting. In addition to other things, this will cover breakfast and lunch for two days, dinner Tuesday night, coffee breaks, etc. Please make checks payable to the University of Florida. A receipt will be available at the meeting.

NOTE: The meeting is all day Tuesday and ½ day on Wednesday. We will provide transportation to the airport as needed. Please make reservations at the University Hilton.
UF Deep dive agenda:

Tuesday, February 3, 2015

8:20   Van pickup at Hilton
8:30 – 9:00         Breakfast
9:00 – 9:30     Welcome and Deep-Dive Overview (3 Sessions)
                   1.  Behavioral emulation (BE): modeling & simulation/emulation methods
                   2.  CS issues (performance, energy, and thermal)
                   3.  Use of reconfigurable computing to accelerate behavioral emulation

* Each of the three deep-dive sessions is designed to be interactive: a combination of short presentations by UF and Tri-lab researchers, intermixed with discussion, demonstrations, etc.

9:30 – 11:30  Session 1: Behavioral Emulation: Modeling & Simulation/Emulation Methods
                   •  UF topics:
                       o  Behavioral characterization
                       o  Parameter estimation
                   •  Tri-lab topics:
                       o  Overview of  FastForward 2 and DesignForward 2 (Jim Ang, SNL)
                       o  Multi-scale architectural simulation with the Structural Simulation Toolkit (Arun Rodrigues, SNL)

11:30 – 12:30     Lunch
12:30 – 2:00  Session 1 (continued): Behavioral Emulation: Beyond Device Level
                   •  UF topics:
                       o  Synchronization for speed
                       o  Congestion modeling
                       o  Behavioral characterization & modeling beyond device level
                   •  Tri-lab topics:
                       o  Using discrete event simulation for programming model exploration at extreme-scale (Jeremy Wilke, SNL)
                       o  ASC next-generation code projects (David Daniel, LANL)

2:00 – 5:00  Session 2: CS Issues (Performance, Energy, and Thermal)
                   •  UF topics:
                       o  Performance and autotuning for hybrid architectures
                       o  Energy and thermal optimization
                       o  Dynamic load balancing
                   •  Tri-lab topics:
                       o  Performance, energy, and thermal benchmarking (Jim Ang, SNL)
                       o  Why power is a performance issue: utilizing overprovisioned systems (Martin Schulz, LLNL)

* There will be an afternoon coffee break in this time slot

6:30  Dinner (University Hilton)
Wednesday February 4, 2015

8:20   Van pickup
8:30 – 9:00  Breakfast
9:00 – 11:00  Session 3: Use of Reconfigurable Computing to Accelerate Behavioral Emulation
    • UF topics:
        o Efficient mapping of behavioral emulation objects (BEOs) onto a system of FPGAs
        o Demo of current single FPGA prototype
        o Transitioning to multiple FPGAs
        o Challenges associated with maximizing emulation speed while maintaining scalability/usability
    • Tri-lab topic:
        • FPGA-based emulation of processing near memory (Maya Gokhale, LLNL)
11:00 – 12:00  Open discussion and planning for action items
12:00   Box lunch; transportation to airport as needed.
UF Deep-Dive Agenda

Tuesday, February 3, 2015

8:20      Van pickup at Hilton
8:30 – 9:00  Breakfast
9:00 – 9:30  Welcome and Deep-Dive Overview (3 Sessions)
1. Behavioral emulation (BE): modeling & simulation/emulation methods
2. CS issues (performance, energy, and thermal)
3. Use of reconfigurable computing to accelerate behavioral emulation

* Each of the three deep-dive sessions is designed to be interactive: a combination of short presentations by UF and Tri-lab researchers, intermixed with discussion, demonstrations, etc.

UF Deep-Dive Agenda

9:30 – 11:30  Session 1: Behavioral Emulation: Modeling & Simulation/Emulation Methods

- UF topics:
  - Behavioral characterization
  - Parameter estimation

- Tri-lab topics:
  - Overview of FastForward 2 and DesignForward 2 (Jim Ang, SNL)
  - Multi-scale architectural simulation with the Structural Simulation Toolkit (Arun Rodrigues, SNL)

11:30 – 12:30  Lunch
12:30 – 2:00  **Session 1 (continued): Behavioral Emulation: Beyond Device Level**

- **UF topics:**
  - Synchronization for speed
  - Congestion modeling
  - Behavioral characterization & modeling beyond device level

- **Tri-lab topics:**
  - Using discrete event simulation for programming model exploration at extreme-scale (Jeremy Wilke, SNL)
  - ASC next-generation code projects (David Daniel, LANL)

2:00 – 5:00  **Session 2: CS Issues (Performance, Energy, and Thermal)**

- **UF topics:**
  - Performance and autotuning for hybrid architectures
  - Energy and thermal optimization
  - Dynamic load balancing

- **Tri-lab topics:**
  - Performance, energy, and thermal benchmarking (Jim Ang, SNL)
  - Why power is a performance issue: utilizing overprovisioned systems (Martin Schulz, LLNL)

* There will be an afternoon coffee break in this time slot

6:30  **Dinner (University Hilton)**
UF Deep-Dive Agenda

Wednesday February 4, 2015
9:00 – 11:00  Session 3: Use of Reconfigurable Computing to Accelerate Behavioral Emulation
- UF topics:
  0 Efficient mapping of behavioral emulation objects (BEOs) onto a system of FPGAs
  0 Demo of current single FPGA prototype
  0 Transitioning to multiple FPGAs
  0 Challenges associated with maximizing emulation speed while maintaining scalability/usability
- Tri-lab topic:
  0 FPGA-based emulation of processing near memory (Maya Gokhale, LLNL)
11:00 – 12:00 Open discussion and planning for action items
12:00          Box lunch; transportation to airport as needed.

Behavioral Emulation
for Design-Space Exploration of CCMT Apps

Principal Investigators:
Dr. Alan George, Dr. Herman Lam, Dr. Greg Stitt

Student Project Leaders:
Nalini Kumar, Carlo Pascoe, Dylan Rudolph

NSF Center for High-Performance Reconfigurable Computing (CHREC)
ECE Department, University of Florida
Outline

- Project context, scope, & focus
- Behavioral Emulation approach
- Research thrusts

Context: DOE Co-design

(One of) the difficulties of co-design

Co-design gets more difficult the further you get from open collaboration and the closer you get to the “truth” (particularly with competing vendors and national security applications)

- ASC concerns
  - National Security Applications
  - Unclassified, but not open applications
  - Open Co-design
    - Released Proxy Apps
    - Open vendor information
- Vendor concerns
  - Open Co-design
  - Standard NDA
  - Deep NDA

• ASC: Involve staff with clearances in co-design efforts and developing proxy apps
• Vendor: Limit number of lab staff engaging in multiple “deep NDA” discussions
Approach: Sandia Co-design Capabilities

- Proxy Applications (Mantevo):
  - Application source for architecture-centric optimization and analysis
  - [http://mantevo.org](http://mantevo.org)
- HPC Architectural Simulation Framework (SST):
  - Flexible to accommodate fidelity/speed tradeoffs
  - [http://SST-simulator.org](http://SST-simulator.org)
- ASC Advanced Architecture Test Beds:
  - Evolving examples of COTS “state-of-the-art”
- Abstract Machine Model (AMM) Definitions and associated Proxy Architectures
  - Supported by SC/ASCR Computer Architecture Lab

Sandia Exascale Architectures R&D Overview
James A. Ang, PhD, Manager and James Laros III, Scalable Computer Architecture Department
NNSA/ASC Program PSAAP II TST Site Visit
Univ. of Florida, Center for Compressible Multi-phase Turbulence
November 3-4, 2014

BEOs & Behavioral Emulation Flow
**Scope and Focus**

**Application** Design-Space Exploration (DSE)

- **Given** characteristics of promising exascale architectures (at device, node, and system level),
  - C/o vendor roadmaps & future technologies (e.g., FastForward 2 & DesignForward 2)
  - Explore different ways to parallelize exascale applications

- **Focused study**
  - Not intending to perform DSE on all types of exascale apps
  - Focus on exascale apps relevant to our CCMT Center

⇒ Along with Behavioral Emulation approach (discussed later)

⇒ This focus allows for optimizations & modeling techniques not available for general-purposed system simulators

---

**Approach: Behavioral Emulation**

- How may we study Exascale before the age of Exascale?
  - **Analytical studies** – *systems are too complicated*
  - **Software simulation** – *simulations are too slow at scale*
  - **Behavioral emulation** – *to be defined herein*
  - **Functional emulation** – *systems too massive and complex*
  - **Prototype device** – *future technology, does not exist*
  - **Prototype system** – *future technology, does not exist*

- Many pros and cons with various methods
  - We believe **behavioral emulation** is most promising in terms of balance of project goals (accuracy, speed, and scalability, as well as versatility)
Behavioral Emulation (BE)

- **Component-based, coarse-grained simulation**
  - Fundamental constructs called BE Objects (BEOs) act as surrogates
  - BEOs characterize & represent behavior of app, device, node, & system objects as fabrics of interconnected ArchBEOs (with AppBEOs) up to Exascale

- **Multi-scale simulation**
  - Hierarchical method based upon experimentation, abstraction, exploration

- **Multi-objective simulation**
  - Performance, power, reliability, and other environmental factors

---

<table>
<thead>
<tr>
<th>Apps</th>
<th>Arch</th>
<th>BEO Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro Level</td>
<td>Skeleton-apps</td>
<td>System BEO fabrics</td>
</tr>
<tr>
<td>Meso Level</td>
<td>Mini-apps</td>
<td>Node BEO fabrics</td>
</tr>
<tr>
<td>Micro Level</td>
<td>Kernels</td>
<td>Device BEO fabrics</td>
</tr>
</tbody>
</table>

---

Fundamental Design of an Arch BEO

Arch BEO: *Abstract model (surrogate) of an architecture object*
- Basic primitive in BE approach to studies of Exascale systems

**Emulation Plane**
- **Mimic** appropriate behavior of modeled object
- **Interact** with other BEOs via tokens to support emulation studies

**Management Plane**
- Measure, collect, and/or calculate metrics and statistics
- Support architectural exploration

**Metrics**
- **Performance factors** (execution time, speedup, latency, throughput, etc.)
- **Environmental factors** (power, energy, cooling, temperature)
- **Dependability factors** (reliability, availability, redundancy, overhead)
Conclusions: Research Thrusts

- **Behavioral Characterization** (Session 1 morning)
  - How do we build, calibrate, then validate performance?

- **Parameter Estimation** (Session 1 morning)
  - How do we efficiently capture behavior in surrogates?

- **Synchronization & Congestion** (Session 1 afternoon)
  - How do we handle sync and congestion at scale?

- **Management & Visualization**
  - How do we measure & analyze massive systems & apps?

- **Reconfigurable Architectures** (Session 3)
  - How do we exploit FPGA hardware for speed & scale?

- **Resilience & Energy** (starting after Y1)
  - How do we extend beyond performance attributes?
Exascale Hardware Challenges

- **Left to the *Invisible Hand***
  - Industry follows an evolutionary path focused on Peak Flops
- **In the Era of Dennard**
  - Scaling our *ad hoc* approach to integration of MPPs with COTS microprocessors was acceptable
- **With the end of Dennard scaling,** this is no longer able to meet DOE Mission Application Requirements

Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith, 2004
Exascale Hardware Challenges – cont.

- **We need to Motivate and Influence Architectural Changes**
  - Processor/node Architectures
  - System Architectures

- **Our Investments are not only in Architectures**
  - We cannot just develop new Exascale Architectures and *Throw it over the wall* to our application developers
  - We need Hardware/Software Co-design

- **The transition of the DOE Legacy Code base is another important challenge**
  - Challenge should influence future hardware thru Co-design

Industry Engagement is Vital

- **We need industry involvement**
  - Avoid one-off, stove-piped solutions
  - Continued “product” availability and upgrades beyond DOE support

- **Industry cannot and will not solve the problem alone**
  - Business model obligates industry to optimize for profit, beat competitors
  - Industry investments heavily weighted towards near-term, evolutionary improvements with small margin over competitors
  - Industry funding for long-term technology R&D is limited and constrained
  - Industry does not understand DOE Applications and Algorithms

- **How can we impact industry?**
  - Work with those that have strong advocate(s) within the company
  - Fund research, development and demonstration of long-term technologies that clearly show potential as future mass-market products (or product components)
  - Corollary: do not fund product development (as part of DOE R&D portfolio)
  - Industry will incorporate promising technologies into future product lines
NNSA/ASC and SC/ASCR are partnering to Influence Industry

- Aligned Hardware Architecture Efforts
  - April 2011 MOU signed between SC and NNSA
  - July 2011 Issued RFI on Critical Technologies for Exascale
  - July 2012 Established Fast Forward node-level Critical/Cross Cutting Technology R&D projects
  - October 2013 Established Design Forward interconnect R&D Projects
  - November 2014 Fast Forward 2: Exascale Node Designs
  - TBD: Design Forward 2: Conceptual Designs of Exascale Systems
  - Aligned joint Advanced Technology platform procurements
    - CORAL: Oak Ridge, Argonne, and Lawrence Livermore National Labs
    - APEX: Los Alamos, Lawrence Berkeley and Sandia National Labs

Fast Forward Program

- Objective: *Accelerate transition of innovative ideas from processor and memory architecture research into future products*
- Evaluate advanced research concepts and develop quantitative evidence of their benefit for DOE applications—using Proxy apps and collaborating on Co-design
  - Engage DOE application teams to understand technology trends/constraints (how it impacts their code development)
  - Understand how to *program* these new features
  - Quantitative evidence to lower risk to adoption of innovative ideas by product teams
- Critical Node Technologies and Designs for Extreme-scale Computing
Fast Forward Program

- **Fast Forward 1 (July 2012 – Sept. 2014)**
  - AMD: Heterogeneous processor, Processing-in-memory and 2-level Memory
  - IBM: Advanced Memory Concepts
  - Intel: Core energy efficiency and Processing-near-memory
  - Intel/Whamcloud: Storage reliability, I/O API, burst buffer management
  - Nvidia: Memory hierarchy, processor/packaging/programming

- **Fast Forward 2 (Nov 2014 – 2016)**
  - AMD: Near threshold voltage logic, other low-power computing technologies, and new standardized memory interface
  - Cray: alternative processor design points including ARM microprocessors
  - IBM: investigate next-generation standardized memory interface
  - Intel: energy efficient node and system architectures, including software targeted at developing extreme scale systems
  - Nvidia: focus on energy efficiency, programmability and resilience

Design Forward Program

- **Objective:** R&D of interconnect architectures and conceptual designs for future extreme-scale computers:
    - Overall Interconnect Architecture
    - Interconnect Integration with Processor and Memory
    - Multiple Communication Library Progression and Interaction
    - Interconnect Fabrics and Management
    - Protocol Support
    - Scalability
  - Start is imminent, Design Forward 2: System design and integration
    - Overall System Architecture
    - Energy Utilization
    - Resilience and Reliability
    - Data Movement through the System
    - Packaging Density
    - System Software
    - Programming Environment
**AMD**

- **Processor Research**
  - Heterogeneous nodes which blend CPU and GPU cores
  - Improved energy efficiency
  - Efficient communication and data movement across the die
  - Simplified programming models

- **Memory Research**
  - Investigating new memory technologies
  - Reduced data movement
  - Higher performance
  - Reduced energy consumption
  - *New Memory Interface*: standardized, robust interface to support integration of heterogeneous memory and cores

- **Software Tools**
  - HSA Foundation


**Intel**

- **Processor Research**
  - Lightweight processor cores
  - Fast synchronization
  - Specialized aspects of ISA and processor for data movement
  - Tapered access to memory

- **Interconnect Research**
  - Tapering bandwidth networks
  - Integration of NICs into processor
  - Intelligent data movement to reduce power

- **Software Tools**
  - Open Community Runtime (OCR)
  - Exploration of OpenMP and MPI as legacy environment

NVIDIA

- **Processor Research**
  - Temporal SIMT and Scalarization
    - Reduce effect of wide vectors
  - Coherency and consistency across system
  - Hierarchical memory systems

- **Interconnect Research**
  - Open standards for the data center
  - Support direct GPU messaging

- **Programmability**
  - Global address spaces (PGAS)
  - Efficient cross machine collectives
  - Fast synchronization
  - Active messages
  - Heterogeneous cores


---

IBM

- **Memory Research**
  - Novel Computation near memory
  - Reduction in data movement and associated overhead
  - Advances in programming models, compiler and runtime environment
  - Leverage of emerging memory technologies
  - Advances in memory efficiency
  - Advances in memory system integration, power and reliability management

- **Impact**
  - Large reduction of data movement
  - Significant improvement at system level performance, power efficiency, and reliability
  - Successful exploitation of novel architecture features while abstracting the hardware complexity, enables by evolutionary and revolutionary approaches

Source: IBM FastForward Project Overview (https://asc.llnl.gov/fastforward/IBM-FF.pdf)
ASC and ASCR are partnering on Joint Advanced Technology System Procurements

- The APEX (LANL, LBNL, and SNL) collaboration is intended to result in the procurement of two platforms in ~2020
  - NERSC/ASCR procurement of NERSC-9
  - ACES/ASC procurement of ATS-3 (Advanced Technology System)

- Both platforms will focus on meeting both mission needs and pursuing Advanced Technology concepts
  - We expect to use Non-Recurring Engineering investment to guide and improve system performance and productivity
High-level Design Philosophy for ATS3

- Delivered application performance is the primary driver in support of mission requirements
  - Peak FLOPS requirement will not appear in RFP
- Advanced technology development is assumed to be necessary to meet mission needs
  - Accelerate development of yet to be identified key technologies
  - 3rd round of NRE – (Trinity/NERSC-8, CORAL, APEX)
- APEX are pre-exascale platforms
  - MUST support path to exascale programming models
    - While supporting existing mission needs
  - Support MPI + OpenMP (threads)
    - Matured on Trinity/Cori and CORAL platforms
  - Additional support for other, yet to be identified, MPI+X programming models

APEX Capability Improvement

- An increase in predictive capability requires increases in the fidelity of both geometric and physics models
  - This implies usable large platform memory capacity
- APEX must demonstrate a significant capability improvement
  - Improvement measured relative to Trinity (ATS1) and Cori (NERSC-8)
  - Improvement as a function of performance (total time to solution), increased geometries, increased physics capabilities, power/energy efficiency, resilience and other factors
- Previous DOE investments assumed to be an integral part of production computing for APEX
  - Trinity/NERSC-8 NRE projects: Burst Buffer and Advanced Power management
  - Fast Forward and Design Forward Projects
    - Potential Path Forward project
    - NRE could take select technologies the final yards towards production
Fast Forward and Design Forward Impact

- APEX Team is performing Market Surveys
- Vendors visiting in phases starting in January 2015
  - IBM, Intel, Cray, Nvidia, AMD, SGI, HP, Micron, Broadcom, ARM, etc.
- Fast Forward and Design Forward Accomplishments and Progress have direct influence over the development of APEX technical requirements
- Developing NRE strategy
  - We started early to enable a richer range of NRE topics
Introduction

- Summary of topics that will be discussed in this session
Distributed Behavioral Emulation

**Goal:** Enable fast and scalable simulation of Exascale systems
- Require efficient simulation representation and synchronization mechanisms for PDES*

Different from other approaches in our definitions of processes, events, and event timings
- Three kinds of events: send event, receive event, internal event
- Relation between events are defined w.r.t logical time: correspondence, causality, concurrency

How do we define processes and generate events?

**PDES mapping of Matrix Multiply**

```plaintext
if (node==0) {
    broadcast (A,comm_grp);
    barrier ();
    scatter (B,B*,comm_grp);
    compute (dot_product,A,B*);
    gather (result,comm_grp);
} else {
    broadcast (A,comm_grp);
    barrier ();
    recv (B,B*,node_0);
    compute (dot_product,A,B*);
    send (result,node_0);
}
```

Pseudo-code for parallel matrix multiply (C=BxA)

Where do we get the timestamps from?

- **How do we define processes?**
  - Assuming one thread/processor, one logical process is generated for each processor core
    - ProcBEOs (logical processes) read events from AppBEOs (event queues)

- **How do we generate events?**
  - Partial ordering of events by assigning timestamps using integer timestamps (logical clock)
  - *Real clock timestamps* are used to estimate execution time

Timing diagram for matrix multiply distributed simulation (on 4-core CPU)
Performance Models (1)

How do we generate timestamps for internal events?

- Calibration data is used for developing interpolation models which are used to predict the execution time—**performance models**
  - Data have varying dimension (One-dimensional: Dot Product, Multi-dimensional: Matrix Multiply)
- We are using Kriging interpolation for multi-dimensional interpolation
  - More about performance models in the next talk of this session

What about send and receive events?

- Training/calibration data
- Experimental testbed, Cycle-accurate Device Simulator, Fast Forward 2 vendors, etc.
- In order to update the timestamps of sending and receiving processes, we have to account for:
  - Time taken by the message to reach its destination
  - If the receiving process is busy, the time spent by the message waiting in the queue
  - Time that receiving process spends in waiting for the message to arrive
- In BE, each Send event generates an *event token* with a timestamp equal to its local clock
- We use network performance models to estimate time taken by a token to reach its destination
  - Qualitative parameters are used to mimic movement of packets in network
  - Quantitative parameters help in estimating communication time
    - Some quantitative parameters are functions of independent variables (e.g., latency)
    - Others are fixed information about the network (e.g., hop time)
ProcBEO & CommBEO Calibration

How do we represent physical system with LPs in Behavioral Emulation?

*Each BEO represents a simulation LP*

**ProcBEOs** emulate processing units
- Initialization, computation etc. are internal events
- Interaction with other BEOs are send/receive events
- Update local clocks based on performance models for compute operations
  - A compute operation can be decomposed several ways
  - e.g., matrix multiply can be broken down into either *multiply and add operations*, *dot product*, or a smaller *matrix multiply*
  - Need to account for any non-negligible overheads

**CommBEOs** emulate network switches
- Mimic communication on the network by sending/receiving event tokens instead of real packets to other CommBEOs
- Update the timestamp of the token at each hop through the network

---

**Demo: AppBEO**

- AppBEO for the 3D matrix multiply kernel used in Nek5000
  - AppBEO instructions are compiled into events that ProcBEOs can understand
  - Timestamps, estimates of event execution time, are generated and compiled pre-simulation whenever possible
Demo: ProcBEO

- ProcBEO instance to model a Tile-Gx36 processing unit
  - AppBEO instructions are resolved by the ProcBEO
  - Local processor clock is updated based on the event being processed (computation or communication)

Demo: CommBEO

- CommBEO instance to model a Tile-Gx36 switch
  - Event token is forwarded based on the destination
  - A virtual machine block informs the local ProcBEO of the CommBEO clock
Demo: Simulation

- Using the App-Proc-Comm BEO stack we can define and simulate the behavior of many-core device
  - This simulation has been setup for a 81-core device of which only 36 cores are active in simulation
  - Connection between the device cores is described using a routing table

Behavioral Emulation Workflow

**Step 1: Calibration**
- Computation
- Communication

**Step 2: Validation**
- Microbenchmarks
  - Computation
  - Communication
- Kernels
  - 2D Matrix Multiply

**Step 3: Prediction**
- Kernels on next-gen Tile-Gx72
- Kernels on anticipated Intel Knight’s Landing (KNL)

More on performance models in the next talk in this session
Results: Communication Microbenchmarks

Simulation setup:
- Communication pattern: Tree Broadcast, Naïve Gather, Naïve Scatter
- BEOs modeled: Tilera iMesh network CommBEOs

Observation:
- Simulations under-predict execution time in most cases, can improve calibration to account for setup overhead
- Accuracy broadly improves with increase in number of cores and transfer size

Results: Parallel 2D Matrix Multiply

Prediction Error (Fine-grained Decomposition)

Prediction Error (Coarse-grained Decomposition)

Simulation setup: compute models for matrix multiply, loop overhead, & network parameters

Observations:
- Abstraction of compute details improves simulation accuracy at a one-time cost of training effort
- Accuracy of simulations is a function of domain, no. of samples, & other kriging parameters

Fewer cores means more share of work performed by each processor. For fine-grained decomposition, more error incurred.

Computation dominates communication, resulting in high total error
Error in dot-product model gets multiplied several times over

Abstraction improves simulation accuracy for all problem sizes
Performance Prediction

How do we simulate future or notional systems?
With some confidence in Behavioral Emulation approach we can proceed to study next-generation devices
- Ability to evaluate what-if scenarios by changing BEOs parameters

1. **Tile-Gx72**: Existing Tilera many-core processor
   - Largest device made by Tilera: 72 cores
   - Cores in Tile-Gx72 are identical to cores in Tile-Gx36
   - To simulate Tile-Gx72, we scale simulation to 72 Proc & CommBEOS

2. **Knight’s Landing (KNL)**: Anticipated Intel many-core processor
   - Rumored to have Xeon Phi-type cores with Mesh network
   - To simulate anticipated Knight’s Landing
     - Calibrate XeonPhi ProcBEOS based on existing XeonPhi processor cores
     - Use validated CommBEOS developed for iMesh network
   - 64-core device: similar in size to existing Xeon Phi
   - 100-core device: probable size; larger than existing devices

Interaction with Fast Forward 2 vendors will provide key information for carrying out useful simulations

Results: Prediction for Tile-Gx72 & Intel KNL

**Simulation Setup:**
- 72-Core Tile Device (Gx72)
- Twice as many cores, laid out in 9 by 8 mesh

**Simulation Setup:**
- BEOs modeled (For KNL): CommBEOS for Tilera iMesh, ProcBEOS for XeonPhi

**2D Matrix Multiply**

Larger matrix sizes utilize the Gx72 device better

Communication overshadows computation on larger KNL100 device, resulting in no speedup over KNL64

Different application algorithm (2D block decomposition) may scale better on KNL100
Identified Issues (1)

- How to systematically collect data?
  - Benchmarking automation is needed
    - For wide range of app and system parameters, on targeted platforms
    - Benchmark suite with basic set of prog. methods (e.g., OpenMP, OpenCL, & MPI)

- How do we automate modeling process?
  - Device-independent techniques

- How to easily repeat experiments?
  - Need automatic porting of benchmark suite
    - On new platforms
    - On upgrades of existing platform

- How do we select practical techniques for interpolation on multi-dimensional data for a given computation?
  - e.g., with matrix multiply: m, n, p, data type, memory affinity

- How do we determine appropriate granularity for compute decomposition?
  - Multiscale approach

Next talk in this session

Identified Issues (2)

- Code to skeleton apps
  - Transforming applications from C/C++/Fortran source to high-level instructions on AppBEO

- Simulation synchronization
  - Global vs. distributed simulator clock

- Scalability of software simulator
  - Message-passing simulator
  - Leverage and integrate SST Macro/Micro

- How do we model CommBEO congestion?
  - Event timing with CommBEO ingress
  - Exploring flit-level, packet-based, flow-based, & hybrid model

We will present our thoughts on these issues in the afternoon session
Questions?

References

System (macro-scale) Simulators


References

Device (micro-scale) & Node (meso-scale) Simulators


Object-oriented System Modeling


Hardware Emulation


Supercomputer-specific Modeling & Simulation


Analytical Modeling


**APPENDIX**

---

**AppBEO representation**

- Need representation of applications that simulator can understand
  - AppBEOS are list of instructions processed by ProcBEOS
  - Small and simple description allows easy development
    - Developer does not need to worry about creating working application code
  - Intermediate format can be compiled into format specific to simulation platform

**AppBEO (high-level description)**

```plaintext
// Define group as nodes 0-3
VAR commGrp=0:3
// Broadcast matrix A
(dataSize=64*64/2) to group
Bcast(int32,2048,0,commGrp)
// Barrier sync
Barrier(commGrp)
// Scatter 1/4 of matrix B
(dataSize=(64*64)/(4*2)) to each node
Scatter(int32,512,0,commGrp)
// Perform dot product of vector size 64
DotProduct(int32,64)
// Gather solutions from matrices
(dataSize=(64*64)/(4*2))
Gather(int32,512,commGrp)
Done
```

**Intermediate format (AppBEO for node0)**

```plaintext
send 1 1 129971 1
recv 4
send 2 2 129971 1
recv 8
send 13 1 381 1
recv 12
send 16 1 32420 1
recv 17
send 18 2 32420 1
recv 19
send 20 3 32420 1
recv 21
advt 5753856
```

**Human Readable Intermediate Format (debug mode)**

```plaintext
// Bcast(int32,2048,0,commGrp)
send 1 1 129971 1 Send broadcast to node 1
recv 4 Receive acknowledgement for broadcast from node 1
send 2 2 129971 1 Send broadcast to node 2
recv 8 Receive acknowledgement for broadcast from node 2
// Barrier(commGrp)
send 13 1 381 1 Send barrier to node 1
recv 12 Received barrier from node 0
// Scatter(int32,512,0,commGrp)
send 16 1 32420 1 Scatter from master to node 1
recv 17 Receive acknowledgement for scatter from 1
send 18 2 32420 1 Scatter from master to node 2
recv 19 Receive acknowledgement for scatter from 2
send 20 3 32420 1 Scatter from master to node 3
recv 21 Receive acknowledgement for scatter from 3
// DotProduct(int32,64)
advt 5753856 Advance timer for compute time in dot product
```
## Parallel 2D Matrix Multiply (fine-grained)

### Tile-Gx36

<table>
<thead>
<tr>
<th>Sim (ms) matrix size</th>
<th>2 threads</th>
<th>4 threads</th>
<th>8 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bcast</td>
<td>Scatter</td>
<td>Gather</td>
</tr>
<tr>
<td>64x64</td>
<td>0.1304</td>
<td>0.0655</td>
<td>11.5241</td>
</tr>
<tr>
<td>128x128</td>
<td>0.5182</td>
<td>0.2597</td>
<td>83.6615</td>
</tr>
<tr>
<td>256x256</td>
<td>2.0732</td>
<td>1.0367</td>
<td>637.7964</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testbed (ms) matrix size</th>
<th>2 threads</th>
<th>4 threads</th>
<th>8 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bcast</td>
<td>Scatter</td>
<td>Gather</td>
</tr>
<tr>
<td>64x64</td>
<td>0.1343</td>
<td>0.0661</td>
<td>1.7000</td>
</tr>
<tr>
<td>128x128</td>
<td>0.5338</td>
<td>0.2612</td>
<td>76.2128</td>
</tr>
<tr>
<td>256x256</td>
<td>2.1424</td>
<td>1.0479</td>
<td>606.9473</td>
</tr>
<tr>
<td>1024x1024</td>
<td>35.2284</td>
<td>17.6642</td>
<td>38378.2033</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>% error matrix size</th>
<th>2 threads</th>
<th>4 threads</th>
<th>8 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bcast</td>
<td>Scatter</td>
<td>Gather</td>
</tr>
<tr>
<td>64x64</td>
<td>-2.91</td>
<td>-0.94</td>
<td>18.79</td>
</tr>
<tr>
<td>128x128</td>
<td>-2.93</td>
<td>-0.58</td>
<td>10.04</td>
</tr>
<tr>
<td>256x256</td>
<td>-3.23</td>
<td>-1.07</td>
<td>5.08</td>
</tr>
<tr>
<td>512x512</td>
<td>-5.04</td>
<td>-6.22</td>
<td>2.47</td>
</tr>
<tr>
<td>1024x1024</td>
<td>-3.90</td>
<td>-5.75</td>
<td>1.32</td>
</tr>
</tbody>
</table>

---

**CCMT**
## Parallel 2D Matrix Multiply (coarse-grained)

### Tile-Gx36

#### Sim (ms)

<table>
<thead>
<tr>
<th>matrix size</th>
<th>2 threads</th>
<th>4 threads</th>
<th>8 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bcast</td>
<td>Scatter</td>
<td>Compute</td>
<td>Gather</td>
</tr>
<tr>
<td>64x64</td>
<td>0.1184</td>
<td>0.0655</td>
<td>9.7571</td>
</tr>
<tr>
<td>128x128</td>
<td>0.5182</td>
<td>0.2597</td>
<td>76.2505</td>
</tr>
<tr>
<td>256x256</td>
<td>2.0732</td>
<td>1.0387</td>
<td>652.5564</td>
</tr>
</tbody>
</table>

#### Festbed (ms)

<table>
<thead>
<tr>
<th>matrix size</th>
<th>2 threads</th>
<th>4 threads</th>
<th>8 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bcast</td>
<td>Scatter</td>
<td>Compute</td>
<td>Gather</td>
</tr>
<tr>
<td>64x64</td>
<td>0.1143</td>
<td>0.0661</td>
<td>9.7008</td>
</tr>
<tr>
<td>128x128</td>
<td>0.5338</td>
<td>0.2612</td>
<td>76.2128</td>
</tr>
<tr>
<td>256x256</td>
<td>2.1424</td>
<td>1.0479</td>
<td>606.9473</td>
</tr>
</tbody>
</table>

#### % error

<table>
<thead>
<tr>
<th>matrix size</th>
<th>2 threads</th>
<th>4 threads</th>
<th>8 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bcast</td>
<td>Scatter</td>
<td>Compute</td>
<td>Gather</td>
</tr>
<tr>
<td>64x64</td>
<td>-2.91</td>
<td>-0.94</td>
<td>0.52</td>
</tr>
<tr>
<td>128x128</td>
<td>-2.93</td>
<td>-0.58</td>
<td>0.05</td>
</tr>
<tr>
<td>256x256</td>
<td>-3.23</td>
<td>-1.07</td>
<td>7.51</td>
</tr>
<tr>
<td>1024x1024</td>
<td>-3.90</td>
<td>-5.75</td>
<td>-76.93</td>
</tr>
</tbody>
</table>

---

**CCMT**
## Parallel Sobel Filtering

### Tile-Gx36

#### Testbed (ms)

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Scatter</th>
<th>Compute_Gx</th>
<th>Compute_Gy</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>0.308</td>
<td>35.666</td>
<td>36.179</td>
<td>0.630</td>
<td>72.464</td>
</tr>
<tr>
<td>480x320</td>
<td>0.620</td>
<td>71.616</td>
<td>72.651</td>
<td>1.265</td>
<td>145.456</td>
</tr>
<tr>
<td>640x480</td>
<td>1.245</td>
<td>142.968</td>
<td>146.993</td>
<td>2.542</td>
<td>292.448</td>
</tr>
<tr>
<td>800x600</td>
<td>1.950</td>
<td>223.252</td>
<td>229.886</td>
<td>3.966</td>
<td>458.540</td>
</tr>
<tr>
<td>1024x768</td>
<td>3.277</td>
<td>365.944</td>
<td>377.464</td>
<td>6.698</td>
<td>752.560</td>
</tr>
<tr>
<td>1280x1024</td>
<td>5.419</td>
<td>611.279</td>
<td>661.535</td>
<td>10.860</td>
<td>1286.660</td>
</tr>
</tbody>
</table>

#### Simulation (ms)

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Scatter</th>
<th>Compute_Gx</th>
<th>Compute_Gy</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>0.306</td>
<td>35.750</td>
<td>36.557</td>
<td>0.604</td>
<td>73.233</td>
</tr>
<tr>
<td>480x320</td>
<td>0.610</td>
<td>71.501</td>
<td>73.114</td>
<td>1.210</td>
<td>146.440</td>
</tr>
<tr>
<td>640x480</td>
<td>1.219</td>
<td>143.002</td>
<td>146.227</td>
<td>2.422</td>
<td>292.875</td>
</tr>
<tr>
<td>800x600</td>
<td>1.903</td>
<td>223.440</td>
<td>228.480</td>
<td>3.784</td>
<td>457.613</td>
</tr>
<tr>
<td>1024x768</td>
<td>3.114</td>
<td>366.084</td>
<td>374.342</td>
<td>6.209</td>
<td>749.755</td>
</tr>
<tr>
<td>1280x1024</td>
<td>5.390</td>
<td>610.140</td>
<td>623.903</td>
<td>10.370</td>
<td>1249.609</td>
</tr>
</tbody>
</table>

#### Error %

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Scatter</th>
<th>Compute_Gx</th>
<th>Compute_Gy</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>0.306</td>
<td>35.750</td>
<td>36.557</td>
<td>0.604</td>
<td>73.233</td>
</tr>
<tr>
<td>480x320</td>
<td>0.610</td>
<td>71.501</td>
<td>73.114</td>
<td>1.210</td>
<td>146.440</td>
</tr>
<tr>
<td>640x480</td>
<td>1.219</td>
<td>143.002</td>
<td>146.227</td>
<td>2.422</td>
<td>292.875</td>
</tr>
<tr>
<td>800x600</td>
<td>1.903</td>
<td>223.440</td>
<td>228.480</td>
<td>3.784</td>
<td>457.613</td>
</tr>
<tr>
<td>1024x768</td>
<td>3.114</td>
<td>366.084</td>
<td>374.342</td>
<td>6.209</td>
<td>749.755</td>
</tr>
<tr>
<td>1280x1024</td>
<td>5.390</td>
<td>610.140</td>
<td>623.903</td>
<td>10.370</td>
<td>1249.609</td>
</tr>
</tbody>
</table>

#### Parallel Sobel Filtering

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Scatter</th>
<th>Compute_Gx</th>
<th>Compute_Gy</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>0.306</td>
<td>35.750</td>
<td>36.557</td>
<td>0.604</td>
<td>73.233</td>
</tr>
<tr>
<td>480x320</td>
<td>0.610</td>
<td>71.501</td>
<td>73.114</td>
<td>1.210</td>
<td>146.440</td>
</tr>
<tr>
<td>640x480</td>
<td>1.219</td>
<td>143.002</td>
<td>146.227</td>
<td>2.422</td>
<td>292.875</td>
</tr>
<tr>
<td>800x600</td>
<td>1.903</td>
<td>223.440</td>
<td>228.480</td>
<td>3.784</td>
<td>457.613</td>
</tr>
<tr>
<td>1024x768</td>
<td>3.114</td>
<td>366.084</td>
<td>374.342</td>
<td>6.209</td>
<td>749.755</td>
</tr>
<tr>
<td>1280x1024</td>
<td>5.390</td>
<td>610.140</td>
<td>623.903</td>
<td>10.370</td>
<td>1249.609</td>
</tr>
</tbody>
</table>
## Parallel 2D Matrix Multiply (fine-grained)

### Tile-Gx36

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Bcast</th>
<th>Scatter</th>
<th>Compute</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>64x64</td>
<td>0.26</td>
<td>0.16</td>
<td>0.64</td>
<td>0.17</td>
<td>2.32</td>
</tr>
<tr>
<td>128x128</td>
<td>1.04</td>
<td>0.59</td>
<td>4.66</td>
<td>0.60</td>
<td>11.08</td>
</tr>
<tr>
<td>256x256</td>
<td>4.15</td>
<td>2.29</td>
<td>35.43</td>
<td>2.30</td>
<td>60.81</td>
</tr>
<tr>
<td>512x512</td>
<td>16.60</td>
<td>9.09</td>
<td>275.92</td>
<td>9.10</td>
<td>377.14</td>
</tr>
<tr>
<td>1024x1024</td>
<td>67.71</td>
<td>36.28</td>
<td>2180.57</td>
<td>36.30</td>
<td>2591.75</td>
</tr>
</tbody>
</table>

### Tile-Gx72

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Bcast</th>
<th>Scatter</th>
<th>Compute</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>64x64</td>
<td>0.26</td>
<td>0.19</td>
<td>0.20</td>
<td>0.21</td>
<td>2.64</td>
</tr>
<tr>
<td>128x128</td>
<td>1.04</td>
<td>0.63</td>
<td>1.59</td>
<td>0.66</td>
<td>10.73</td>
</tr>
<tr>
<td>256x256</td>
<td>4.15</td>
<td>2.33</td>
<td>12.64</td>
<td>2.36</td>
<td>48.52</td>
</tr>
<tr>
<td>512x512</td>
<td>16.60</td>
<td>9.26</td>
<td>100.93</td>
<td>9.28</td>
<td>244.03</td>
</tr>
<tr>
<td>1024x1024</td>
<td>67.71</td>
<td>36.79</td>
<td>806.49</td>
<td>36.82</td>
<td>1388.01</td>
</tr>
</tbody>
</table>

## Parallel Sobel Filtering

### Tile-Gx72

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Scatter</th>
<th>Compute_Gx</th>
<th>Compute_Gy</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>0.74</td>
<td>2.01</td>
<td>2.06</td>
<td>1.21</td>
<td>6.13</td>
</tr>
<tr>
<td>480x320</td>
<td>1.38</td>
<td>4.17</td>
<td>4.26</td>
<td>2.40</td>
<td>12.32</td>
</tr>
<tr>
<td>640x480</td>
<td>2.66</td>
<td>8.04</td>
<td>8.23</td>
<td>4.77</td>
<td>23.80</td>
</tr>
<tr>
<td>800x600</td>
<td>4.05</td>
<td>12.85</td>
<td>13.14</td>
<td>7.42</td>
<td>37.57</td>
</tr>
<tr>
<td>1024x768</td>
<td>6.50</td>
<td>20.74</td>
<td>21.20</td>
<td>12.12</td>
<td>60.67</td>
</tr>
<tr>
<td>1280x1024</td>
<td>10.67</td>
<td>34.32</td>
<td>35.09</td>
<td>20.18</td>
<td>100.37</td>
</tr>
<tr>
<td>1600x1200</td>
<td>15.43</td>
<td>50.27</td>
<td>51.41</td>
<td>29.55</td>
<td>146.77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Scatter</th>
<th>Compute_Gx</th>
<th>Compute_Gy</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>0.92</td>
<td>1.12</td>
<td>1.14</td>
<td>1.27</td>
<td>4.67</td>
</tr>
<tr>
<td>480x320</td>
<td>1.60</td>
<td>2.09</td>
<td>2.13</td>
<td>2.46</td>
<td>8.49</td>
</tr>
<tr>
<td>640x480</td>
<td>2.97</td>
<td>4.02</td>
<td>4.11</td>
<td>4.87</td>
<td>16.20</td>
</tr>
<tr>
<td>800x600</td>
<td>4.46</td>
<td>6.70</td>
<td>6.85</td>
<td>7.57</td>
<td>25.81</td>
</tr>
<tr>
<td>1024x768</td>
<td>7.05</td>
<td>10.73</td>
<td>10.97</td>
<td>12.35</td>
<td>41.31</td>
</tr>
<tr>
<td>1280x1024</td>
<td>11.43</td>
<td>17.16</td>
<td>17.55</td>
<td>20.52</td>
<td>66.88</td>
</tr>
<tr>
<td>1600x1200</td>
<td>16.37</td>
<td>25.70</td>
<td>26.28</td>
<td>29.99</td>
<td>98.55</td>
</tr>
</tbody>
</table>

### Table of Speedups

<table>
<thead>
<tr>
<th>Image size</th>
<th>Tile-Gx36</th>
<th>Tile-Gx72</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>6.13</td>
<td>4.67</td>
<td>1.31</td>
</tr>
<tr>
<td>480x320</td>
<td>12.32</td>
<td>8.49</td>
<td>1.45</td>
</tr>
<tr>
<td>640x480</td>
<td>23.80</td>
<td>16.20</td>
<td>1.47</td>
</tr>
<tr>
<td>800x600</td>
<td>37.57</td>
<td>25.81</td>
<td>1.46</td>
</tr>
<tr>
<td>1024x768</td>
<td>60.67</td>
<td>41.31</td>
<td>1.47</td>
</tr>
<tr>
<td>1280x1024</td>
<td>100.37</td>
<td>66.88</td>
<td>1.50</td>
</tr>
<tr>
<td>1600x1200</td>
<td>146.77</td>
<td>98.55</td>
<td>1.49</td>
</tr>
</tbody>
</table>
## Parallel 2D Matrix Multiply (fine-grained)

<table>
<thead>
<tr>
<th>matrix size</th>
<th>Bcast</th>
<th>Scatter</th>
<th>Compute</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>128x128</td>
<td>1.04</td>
<td>0.56</td>
<td>0.07</td>
<td>0.58</td>
<td>8.54</td>
</tr>
<tr>
<td>256x256</td>
<td>4.15</td>
<td>2.07</td>
<td>0.38</td>
<td>2.09</td>
<td>33.60</td>
</tr>
<tr>
<td>512x512</td>
<td>16.60</td>
<td>8.21</td>
<td>2.28</td>
<td>8.24</td>
<td>135.00</td>
</tr>
<tr>
<td>1024x1024</td>
<td>67.70</td>
<td>32.60</td>
<td>15.10</td>
<td>32.70</td>
<td>555.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>matrix size</th>
<th>Tile-Gx36 (ms)</th>
<th>Gx72 (ms)</th>
<th>KNL 64 (ms)</th>
<th>KNL 100 (ms)</th>
<th>Speedup (Gx36 vs KNL64)</th>
<th>Speedup (Gx36 vs KNL100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128x128</td>
<td>11.08</td>
<td>10.73</td>
<td>8.54</td>
<td>10.50</td>
<td>1.30</td>
<td>1.05</td>
</tr>
<tr>
<td>256x256</td>
<td>60.81</td>
<td>48.52</td>
<td>33.60</td>
<td>41.20</td>
<td>1.81</td>
<td>1.48</td>
</tr>
<tr>
<td>512x512</td>
<td>377.14</td>
<td>244.03</td>
<td>135.00</td>
<td>165.00</td>
<td>2.79</td>
<td>2.29</td>
</tr>
<tr>
<td>1024x1024</td>
<td>2591.75</td>
<td>1388.01</td>
<td>555.00</td>
<td>673.00</td>
<td>4.67</td>
<td>3.85</td>
</tr>
<tr>
<td>2048x2048</td>
<td>18000.00</td>
<td>--</td>
<td>2270.00</td>
<td>2721.73</td>
<td>7.93</td>
<td>6.61</td>
</tr>
</tbody>
</table>

## Parallel Sobel Filtering

<table>
<thead>
<tr>
<th>Image size</th>
<th>Scatter</th>
<th>Compute_Gx</th>
<th>Compute_Gy</th>
<th>Gather</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>0.76</td>
<td>0.12</td>
<td>1.02</td>
<td>2.22</td>
<td></td>
</tr>
<tr>
<td>480x320</td>
<td>1.55</td>
<td>0.26</td>
<td>2.44</td>
<td>4.72</td>
<td></td>
</tr>
<tr>
<td>640x480</td>
<td>2.66</td>
<td>0.49</td>
<td>4.37</td>
<td>8.22</td>
<td></td>
</tr>
<tr>
<td>800x600</td>
<td>4.38</td>
<td>0.80</td>
<td>7.56</td>
<td>13.70</td>
<td></td>
</tr>
<tr>
<td>1024x768</td>
<td>6.55</td>
<td>1.27</td>
<td>11.50</td>
<td>20.80</td>
<td></td>
</tr>
<tr>
<td>1280x1024</td>
<td>10.80</td>
<td>2.11</td>
<td>19.40</td>
<td>34.60</td>
<td></td>
</tr>
<tr>
<td>1600x1200</td>
<td>15.60</td>
<td>3.09</td>
<td>28.70</td>
<td>50.70</td>
<td></td>
</tr>
<tr>
<td>1920x1080</td>
<td>16.70</td>
<td>3.34</td>
<td>31.20</td>
<td>54.80</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image size</th>
<th>Tile-Gx36</th>
<th>KNL 64</th>
<th>KNL 100</th>
<th>Speedup (Gx36 vs KNL64)</th>
<th>Speedup (Gx36 vs KNL100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>320x240</td>
<td>6.13</td>
<td>2.22</td>
<td>2.86</td>
<td>2.76</td>
<td>2.15</td>
</tr>
<tr>
<td>480x320</td>
<td>12.32</td>
<td>4.72</td>
<td>4.9</td>
<td>2.61</td>
<td>2.51</td>
</tr>
<tr>
<td>640x480</td>
<td>23.80</td>
<td>8.22</td>
<td>9.11</td>
<td>2.90</td>
<td>2.61</td>
</tr>
<tr>
<td>800x600</td>
<td>37.57</td>
<td>13.7</td>
<td>12.3</td>
<td>2.74</td>
<td>3.05</td>
</tr>
<tr>
<td>1024x768</td>
<td>60.67</td>
<td>20.8</td>
<td>21.9</td>
<td>2.92</td>
<td>2.77</td>
</tr>
<tr>
<td>1280x1024</td>
<td>100.37</td>
<td>34.6</td>
<td>35.6</td>
<td>2.90</td>
<td>2.82</td>
</tr>
<tr>
<td>1600x1200</td>
<td>146.77</td>
<td>50.7</td>
<td>48.6</td>
<td>2.89</td>
<td>3.02</td>
</tr>
<tr>
<td>1920x1080</td>
<td>158.00</td>
<td>54.8</td>
<td>55.3</td>
<td>2.88</td>
<td>2.86</td>
</tr>
</tbody>
</table>
Performance Modeling

How do we generate timestamps for internal events?

- Calibration data is used for developing interpolation models which are used to predict the execution time
  - Performance models
    - Data have varying dimension (One-dimensional: Dot Product, Multi-dimensional: Matrix Multiply)
- We are using Kriging interpolation for multi-dimensional interpolation
  - More about performance models in the next talk of this session

What about send and receive events?

Motivation:

- Behavioral emulation necessitates that simulators do not perform cycle-accurate (or otherwise complex) operations
- However, simulators must still know time required for an operation (e.g., Matrix Multiply) based on input sizes (e.g., \([M, N, P] = [256, 100, 42]\))

Goals:

- Estimate values of specific non-integral numerical parameters (e.g., computation time) before or during simulation without having access to real generators (e.g., the target CPU) of those parameters
  - Methods which produce these parameters are surrogates for the real generators
  - Any access to the target platforms is assumed to be strictly in advance of the simulation
- Determine efficacy of possible models in multi-dimensional domains
- Perform uncertainty analysis to determine degree to which estimation is introducing error into the simulator
Performance Modeling

Approach:
- On the target platform or simulator, take a number of representative samples (of the parameter of interest) within the expected domain
- Using these samples, interpolate any other needed values just prior to, or during, the simulation

Kriging:
- A product of the geostatistics community, used for many-dimensional sparse interpolation
- Other (likely less accurate) options:
  - Radial basis functions
  - Nearest-neighbor

Universal Kriging:
- Inputs: Variogram (spatial relationship of data), Polynomial Degree, Samples
- Internal: Computes weights for each of the samples
- Outputs: Interpolated values, estimate of variance at interpolation points

Aside: What do our data look like? (1 of 3: Easy)

- Matrix Multiplication is relatively easy, but still non-trivial, to interpolate
  - Banding near numbers which are additive combinations of powers of two
  - Bands can have times of 2-10 times longer than their neighbors
  - Likely due to cache particularities in the system
- Example details:
  - Platform: x86 Ivy-Bridge Quad
  - Single Core Used
  - Triple-loop textbook method
  - C, GCC, -O2
Performance Modeling

Aside: What do our data look like? (2 of 3: Hard)

- FFTs (FFTW) are the most difficult benchmarks in the set
  - Computation time strongly related to how composite the input size is
  - Adjacent samples can jump by more than an order of magnitude
  - Interpolating to obtain an average error fraction of just below 1 is difficult
- Example details:
  - Platform: x86 Ivy-Bridge Quad
  - Single Core Used
  - FFTW
  - C, GCC, -O2

Performance Modeling

Aside: What do our data look like? (3 of 3: Other)

- CUDA BLAS DGEMM is different
  - Divided into blocks
  - Not symmetrical about diagonal
  - Partitions are differently sized
- Example details:
  - Platform: Quadro K600 GPU
Performance Modeling

Approach: Interpolation Process: Part 1 of 4

**Step One:** obtain ordered, but slightly randomized samples within the domain

- Randomization prevents aliasing along areas of unusual time
- Determining the number of samples to use is one of the research goals
- Even sampling may not be ideal, but to do otherwise requires unavailable a priori knowledge

- Example details:
  - These samples cover 0.17% of the domain, a relatively small amount
  - Notably: they mostly missed the bands of high computation time

Performance Modeling

Approach: Interpolation Process: Part 2 of 4

**Step Two:** using those samples, construct an interpolation model

- After entering these parameters, we have a model which can substituted for the real data in the simulator (the surrogate)
- It will not be perfect, as it is subject to both the samples and the parameters
- For example: this model loses most of the banding of the original data
Performance Modeling

Approach: Interpolation Process: Part 3 of 4

Step Three: evaluate the model with another dataset and an error metric

- Test data are completely random within the domain
- Our Error Metric: average mean-squared fractional error

Performance Modeling

Approach: Interpolation Process: Part 4 of 4

Step Four: If the error was too great, go back to step one, but use different parameters

- Obtaining good parameters for Kriging (without much knowledge of the underlying processes) is difficult, so we let computers do it for us
  - Specifically: the Step Four to Step One Transition (revising parameters if needed) is done by Genetic Algorithms (GA)
    - GAs simulate the evolutionary process by only allowing the best Genomes (solutions) to pass on their genes to the next generation
    - Process is computationally intensive, as many Kriging evaluations are required, but an optimal solution is likely to result
  - The GAs stop only when a good-enough solution is found, which may take up to several days
- After all iterations, we can incorporate the model into the simulator
- It is necessary to obtain good Kriging parameters for many different sample-fractions, so we can know how many samples are needed
There are a number of tunable parameters which must be selected for each stage:

- **Sampling**
  - Domain of interest
  - Sampling strategy (linear, random, logarithmic, etc.)
  - Number of points
  - Noise reduction strategy

- **Model construction**
  - Interpolation method (Kriging, RBF, etc.)
  - Method parameters

- **Error quantification**
  - Number of evaluation points
  - Evaluation sampling method
  - Error metric

### Performance Modeling (Results)

**Single-Dimensional Benchmarks**  
(One Input Parameter)

**Multi-Dimensional Benchmarks**  
(Two and Three Input Parameters)
Performance Modeling (Results)

Kriging versus Nearest-Neighbor

- Kriging outperforms nearest-neighbor interpolation in (most) all cases (values greater than unity)
- There is little or no improvement for FFT
- For the high-algorithmic-complexity algorithms, Kriging is much better
- Kriging has a better improvement for more sparse sampling

Identified Issues

- Kriging requires the selection of a number of interpolation parameters, the choice of which is not obvious
- We must select a sampling strategy (or set of strategies), each with a number of parameters which must be picked
- In order to effectively model functions like the FFTs, an alternative to Kriging must be used
  - This may be a type of domain partitioning which is used in conjunction with Kriging, or something entirely different
- Performing Kriging can be cumbersome on some platforms, so it may be excluded from use at run-time (which may not be necessary)
**Research Thrust**

**Synchronization & Congestion**

---

**Motivation:**
- Current BE infrastructure ensures causality but isn’t optimized for large-scale simulation.
- Communication events in BE abstract away details of network packet transfers making robust congestion modeling necessary.

**Goal:** Adapt synchronization and congestion-modeling techniques to support simulation experiments with *millions of behavioral objects*.

---

**Simulator Landscape**

Disclaimer: This is first attempt at classifying different types of simulators. Our goal is to understand the unique features and associated advantages of each simulator for adoption choice.

---

*Can we find events that can be executed in parallel?*
Expressing Causality

- Maintaining causality ensures correctness and exposing concurrency helps in determining the maximum amount of parallelism in the simulation
- **Causal history** – all events in an event’s past that can affect its outcome
  - Causal history is enough to determine the ordering of events
- Lamport clocks, not consistent with real time, are consistent with causality
  - Partial ordering of events by assigning timestamps using logical clock
- Vector Time is consistent and can characterize causality

Do we need complete causal history of an event?

Search for Efficient Representation

Can we sacrifice on accuracy to speed up simulations?

- Size of time vectors is a limiting factor for scalable simulation
  - Several methods exist for compressing message timestamps which tradeoff simulation speed, storage, and complete ordering of events
  - Utilize only direct dependencies or use causal distributed breakpoints
- Creating **concurrent regions** may be cheaper than causality
  - A dependence blocks can be regarded as a single, atomic event
  - This approach offers some scope for abstracting execution details
- There may also be merit in focusing only on the state transitions

What are our options for synchronizing execution of these events?
Causality & Event Synchronization

- Non-aggressive vs Aggressive
  - Conservative vs Optimistic
  - Hybrids

- “Limited” Optimistic
  - \textit{Window-based}: events with timestamps within some agreed upon window are executed between synchronizations
  - \textit{Space-based}: LPs divided into clusters, each cluster executes optimistically, interaction between clusters is conservative.
  - \textit{Penalty-based}: LPs are either penalized and blocked or favored and not blocked depending on rollback behavior
  - \textit{Knowledge-based}: optimistic execution, broadcast message is sent out if error is detected
  - \textit{Probabilistic}: periodic probabilistic synchronization of LPs
  - \textit{State-based}: optimism continuously adjusted based on local state information
- Application dependent

How is event synchronization handled in existing simulators?

Synchronization Schemes

- Majority coarse-grained simulators use conservative schemes *
  - BigSim is trace-driven, ParSim is time-driven, and Fsim lacks timing model
  - SST uses component-based back-end with global queue and MPI barriers
- ROSS uses optimistic time-warp protocol for event ordering

- We envision adopting one or combination of multiple approaches tuned for BE
  - Sync. scheme fixed or selected by user as per their need
  - Schemes for different scales of simulation or simulation platform
  - Multi-pass simulation

How can we specialize and tune these approaches for BE?

* List of references at end
Parameters for Scalable PDES Design

How can we specialize and tune these approaches for BE?

- Various options available to explore
  - **Partitioning** - how to cluster LPs?
  - **Adaptability** - how does simulator change based on state?
  - **Aggressiveness** - how much should conditional knowledge be processed?
  - **Accuracy** - how much error can be tolerated?
  - **Risk** - how far should a potentially incorrect message be propagated?
  - **Synchrony** - what is the degree of temporal binding or coupling of LPs?
  - **Knowledge embedding** - knowledge of an LPs behavioral attributes that is embedded in the simulation
  - **Knowledge dissemination/acquisition** - how much does LP initiate transmission/request of information to/from other LPs?

But choice of synchronization mechanism will dictate our methods and ability to model congestion on the network.

Congestion Modeling

- Many recent simulators and frameworks have *low-level network models*
  - SST-Micro uses high-fidelity component models to simulate CMP and SMP systems
  - SST/Macro uses packet-level, flow-based, & hybrid train models for system networks
  - FSim and xSim use fine-grained network models
  - FSim, xSim, and BigSim allows high-level latency models and detailed model of communication fabric

- Explore existing congestion models for use in Behavioral Emulation
  - Fine-grained vs. coarse-grained packet-level models
  - Analytical flow-based or train-based models
  - Queuing-theory models
  - System-specific models derived from experiments

How can we specialize and tune these approaches for BE?
Congestion Modeling

- Based on tradeoff studies devise congestion models for BE
  - Adopt one or combination of multiple approaches tuned for BE
  - For most promising approaches, study variations in congestion behavior and scalability with choice of synchronization schemes

- Tuning for Behavioral Emulation
  - Congestion model fixed or selected by user as per their need
  - Different models for different simulation levels (on-chip, inter-node, inter-rack)
  - Models for different levels of simulation platform (exploit different levels of parallelism)

- Choice of synchronization mechanisms and congestion models can significantly affect the design and speed of emulation in hardware

Questions?
References

System (macro-scale) Simulators

Synchronization in PDES
- R. Schwarz “Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail”
References

**Device (micro-scale) & Node (meso-scale) Simulators**

**Object-oriented System Modeling**

**Hardware Emulation**

**Supercomputer-specific Modeling & Simulation**

**Analytical Modeling**
Scalability Methods

Goal: Reduce simulator execution time by exploiting domain-specific simplifying assumptions

- Intended to supplement other simulation efforts by providing techniques and methods which can be used (primarily) prior to simulation-time to speed-up the simulators

- Here, we focus on the simplifying assumptions which are a result of behavioral emulation (as opposed to general DES problems)
  - “We need not be concerned with the result, just the time it took to get there”
  - “We need not be concerned with handling application-specified non-determinism (RNGs)”
  - “We need not be concerned with handling run-time conditional operations”
    - Acceptable: if (my_rank != 0) { do_something(); }
    - Not acceptable: if (current_time >= 1.5) { do_something(); }

- There may be other sets of simplifying assumptions as a result of being concerned mostly with one application (CMT-related), and one machine size (Exascale)

Current methods of interest:
- Global task graph manipulation (done at compile time, see compilation overview below)
  - Generate a global task graph
  - Manipulate this task graph to reduce the number of total tasks required
- Micro-scale symmetry exploitation
  - Find code blocks (within a simulation process) which are isomorphic to blocks in other simulation processes
  - Publish these code blocks to a “cache” (or model them in advance) to avoid repetition at the micro scale

Application Description

Machine Description

Parser

Abstract Syntax Tree

Code Generator

Process Code

Task-Graph Generator

Task Graph

Task-Graph Modifier

Code Generator

Abstract Process Code

To simulator
To be able to statically generate a global task graph for all processes, a limited language is necessary (example AppBEO right)

Language can support:
- Unconditional looping
- Compile-time evaluated conditionals
- Function calls without side effects
- Basic macros

Language must avoid:
- Variable manipulation and assignment (all expressions must be evaluated at compile time)
- Run-time conditional statements
- Random number generators
- Anything which give the program a state which is not just the location in the instruction stream

These are acceptable compromises, as we require only a description language, not a Turing-complete one

High level process:
- Produce this global graph (has about $10^{12}$ entries for a substantial application on an exascale machine) or produce a highly-symmetric compressed task graph (if permitted by the application)
- Section off the graph at a natural hardware level (e.g., per node)
- Apply combination and simplification rules to reduce the number of graph nodes
Micro-Scale Symmetry Exploitation

- Task graph simplification will produce multi-operation blocks which require new performance models
- We generate (at compile time, or run time) a model for this block, and then use the model, but do not re-simulate the internals of the block
  - Block is modeled based on the relative timing of its input signals
  - Block is limited to roughly a maximum of 4-6 inputs before this modeling becomes untenable
- This method is only helpful if there is enough symmetry to find many isomorphic blocks in the simulation (likely)
  - This is likely because each node in a large application is likely to do very similar things as other nodes

Issues and Conclusions

- Known plausible limitations to task graph simplification:
  - Must be able to generate global task graph (may be not feasible due to time constraints)
  - Must be able to effectively manipulate this graph in a timely manner
  - Application must be written such that the task graph can be simplified

- Known plausible limitations to micro-scale symmetry exploitation:
  - Application must be written such that there is enough symmetry to exploit across processes (likely)
  - It must be possible to generate models fairly quickly

- Conclusions:
  - If they are feasible, the use of these methods could permit huge simulator speedups for large machines
  - They also may permit less accurate, but very fast, analytical solutions to determining the run-time of a simulated machine-application pair
Why Power is a Performance Issue: Utilizing Overprovisioned Systems (plus: Visual Performance Analysis)

Martin Schulz  
Lawrence Livermore National Laboratory

PSAAP-II Deep Dive, University of Florida • February 3rd/4th, 2015

The Scalability Team  
http://scalability.llnl.gov/

- Part of the Center for Applied Scientific Computing (CASC)
- Main topics
  - Performance analysis tools and optimization
  - Correctness and debugging (incl. STAT, AutomaDeD, MUST)
  - Tool infrastructures (incl. P*MPI, GREMLINs)
  - Resilience and Checkpoint/Restart (incl. SCR)
  - Power-aware and power-limited computing (incl. Adagio)

- Crosscut: Scalability

Abhinav Bhatele  David Boehme  Todd Gamblin  Tanzima Islam  Ignacio Laguna  Kathryn Mohror  Barry Rountree  Martin Schulz
The Scalability Team
http://scalability.llnl.gov/

- Part of the Center for Applied Scientific Computing (CASC)
- Main topics
  - Performance analysis tools and optimization
  - Correctness and debugging (incl. STAT, AutomaDeD, MUST)
  - Tool infrastructures (incl. P*MPI, GREMLINs)
  - Resilience and Checkpoint/Restart (incl. SCR)
  - Power-aware and power-limited computing (incl. Adagio)
- Crosscut: Scalability

Performance Visualization Across Domains

- Example: Performance data of a 256 core CFD run
  - Dense matrix on 8x32 cores
  - Floating point operations
The Boxfish Tool Embodies This Approach

- Input data from multiple measurements
- Drag selection to map data to visualization
- Available Visualization domains
- Selected visualization: 3D Torus
- Choice of mappings: Present data on nodes or links?

Boxfish Usage: Network Visualization

- **Study contention in networks**
  - Map link counters to 3D torus on BG/P
  - Example: AMG2000
- **Processor Neighborhoods**
  - Visualize job allocations around own job
  - Understand performance impact
- **Visualization of fat tree networks**
  - Dominant topology in cluster systems
  - Study impact of routing algorithms
- **Power optimization for networks**
  - Turn off unused links in torus networks
  - Boxfish used to show simulation data
- **Dragonfly Visualization**
MemAxes: Visualizing Memory Traffic

- Shows data mapped to code and machine characteristics
  - Hardware topology
  - Location within the mesh
  - Code locations

Ravel: Showing Lateness on Logical Timelines
Ravel: Trace Visualization Using Logical Time

The Scalability Team
http://scalability.llnl.gov/

- Part of the Center for Applied Scientific Computing (CASC)
- Main topics
  - Performance analysis tools and optimization
  - Correctness and debugging (incl. STAT, AutomaDeD, MUST)
  - Tool infrastructures (incl. P*MPI, GREMLINs)
  - Resilience and Checkpoint/Restart (incl. SCR)
  - Power-aware and power-limited computing (incl. Adagio)
- Crosscut: Scalability
Power Consumption of Current Systems (e.g., BG/Q)

Provisioned Power

Goal: Exascale @ 20MW

“Wasted” Power
Goal: Exascale @ 20MW

- Power as a constraint, not an optimization goal
- Overprovisioned systems
  - More hardware that can be powered at once
  - Need mechanisms to control and cap power
  - Need runtime systems to maximize power usage within cap
  - Need Cross-job scheduling control

Importance of Configurations

- Experiment on 32 nodes on LLNL’s TLCC system with a global power bound
- Naïve configuration in red
  - Running all cores
  - This limits number of nodes
- Best configuration in blue
  - Moderate per node power bound
  - Reduced number of cores
- Difference: > 2x
- Consequences:
  - Determining the right configuration is critical
  - Intuition is insufficient -> need new models
Processors are Heterogeneous Under a Power Bound

Dual socket 8-core
mg.C.8
64 processor
34 power bounds

Power lead to unintended noise
- Power variations turn load imbalance
- Power limitations turn back into a performance issue

Large Scale Power Capping Experiments: 90W

- **Census across 2386 processors**
  - mg (multigrid)
    - Runs at 105W
  - ep (embarrassingly parallel)
    - Runs at 90W

- **Chart showing one point for each processor in the system**
  - Performance normalized to fastest unbounded run
  - X-Axis: Slowdown
  - Y-Axis: CPU clock
  - Slowest processors circled
Large Scale Power Capping Experiments: 80W / 65W

Large Scale Power Capping Experiments: < 51W
- **Power capping makes systems heterogeneous**
  - Need more flexible task scheduling and ability to absorb slack
  - Needs to be taken into account during load balancing

- **Slowdown under power caps application specific**
  - Can't use a single knob “processor/silicon efficiency”
  - Working on black and white box models to predict performance

- **Runtime systems can help with more efficient scheduling**

---

**Runnights with Adaptive Power Control**

- **Adagio = Optimizing Energy**
  - Identifies MPI critical path and slows the rest of the system down
  - Up to 20% reduction in energy for real applications

- **Working on power maximizing runtime (Adagio++)**
  - Under a power bound direct power to critical path
  - Scale rest of the computation accordingly
  - Requires accurate models for performance under a power bound
Power-aware Resource Management

- **Power is a global resource**
  - The system power cap must be divided among jobs
  - Static division results in power fragmentation
  - Dynamic management can utilize open resources

- **Direction 1: Power-aware Resource Management**
  - Power as a controlled resource that is allocated
  - FLUX project

- **Direction 2: Runtime Adaptation**
  - Part of a global operating system
  - Detection and reallocation of unused power
  - Transparent to application
  - Need to maintain fairness

Prototype Power Scheduler for the Exascale

- **8 Enclaves with different job mixes**
  - Static vs. dynamic scheduling under same power bound
  - Dynamic power measurement and control
Conclusions

- **Overprovisioning is a way to approach the power bound**
  - Treat power as a system/facility constraint
  - More hardware than can be run at maximal power
  - Optimizing on where to direct the resource power

- **Overprovisioning turns power into a performance problem**
  - Optimal configurations (number of nodes vs. per node cap)
  - Dynamic runtime optimization to steer power to the critical path
  - Adaptive job scheduling to optimize system utilization

- **Challenges going forward**
  - Finer grain control beyond sockets (per core, per sub-component)
  - Accurate modeling under a power bound
  - Integration into next generation resource managers
Long Term Goals

- Parallelization and UQ of Rocflu and CMT-Nek beyond a million cores
- Parallel Performance and Load Balancing
- **Single Processor (Hybrid) Performance**
- Energy Management and Thermal Issues
Hybrid Multicores: Performance, Energy and Thermal Management

- Code Generation for hybrid cores
  - Support for multiple types of cores
  - Support for Vectorization
- Multi-objective optimization
  - Energy
  - Performance
- Thermal Constraints

Rocfun and CMT-Nek → BEOs
- Algorithmic Parameters
- Architectural Parameters
For Energy and Performance

NGEE
- Emulate Processor
- Network
- Energy Requirements for Exascale Computation

Hybrid Multicores, Performance, Energy and Thermal Management

- Multiple Elements can be optimized for Energy
  - Processor (Dynamic Voltage Scaling)
  - Caches (Dynamic Cache Reconfiguration)
  - Buses
  - Memory
- Multi-objective optimization
  - Energy
  - Performance
- Multiple Constraints
  - Thermal Issues
  - Packaging Issues
Multi-core processors

Number of cores are growing

8 cores

AMD Opteron 16-Core Processor

Intel 48-core Processor

Nvidia Fermi GPU

Nvidia Kepler Processor

Nvidia Maxwell GPU

Single Processor Performance (Hybrid Multicores)

<table>
<thead>
<tr>
<th>Multiple Cores</th>
<th>GPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common</strong></td>
<td><strong>Common</strong></td>
</tr>
<tr>
<td>- Multiple flows of Control</td>
<td>- Single/Multiple flows of Control</td>
</tr>
<tr>
<td>- Multiple Local Memories</td>
<td>- Multiple Local Memories</td>
</tr>
<tr>
<td><strong>Differences</strong></td>
<td><strong>Differences</strong></td>
</tr>
<tr>
<td>- Synchronization</td>
<td>- Amount of Local memory</td>
</tr>
<tr>
<td>- Communication</td>
<td>- Communication</td>
</tr>
</tbody>
</table>

2012 AMD A-SERIES OVERVIEW

"Trinity" Details
- 32nm
- ~1.5B transistors
- 246mm²
- Up to 4 x86 Cores
Our Work

**Performance**
- Execution time
- Throughput
- Makespan

**Energy**
- Total energy consumed
- Energy due to Leakage power

**Temperature**
- Maximum temperature
- Average temperature
- Spatial and Temporal gradients

**Environments**
- Single-core
- Multi-core

Optimization Goals
- Execution time
- Throughput
- Makespan
- Total energy consumed
- Energy due to Leakage power
- Maximum temperature
- Average temperature
- Spatial and Temporal gradients

Optimization Techniques
- Scheduling
- DCR
- DVS

Performance, Energy and Thermal Levers

- DVS of Cores
- L1 Cache Reconfiguration
- L2 Cache Reconfiguration
- L2 Cache
- DVS of Buses
- 8 ways in one cache set
Our work

Develop an integrated framework for multicore machines that address:
1. Computation
2. Energy
3. Temperature

Challenging Multi-objective optimization and system issues.

Spectral Element Method

\[
\begin{align*}
\frac{\partial U}{\partial r} (i, j, k) &= \sum_{l=1}^{N_1} A_{il} u_{ijk} \\
\frac{\partial U}{\partial s} (i, j, k) &= \sum_{l=1}^{N_2} B_{il} u_{ilk} \\
\frac{\partial U}{\partial t} (i, j, k) &= \sum_{l=1}^{N_3} C_{il} u_{ijl}
\end{align*}
\]

- If \( N_x = N_y = N_z = N \)
  - Then \( B = C = A^T \)
- Complexity: \( O(N^4) \)
- \( N \) is typically between 5-25
  - A large number of small matrix multiplications

The derivative computing kernel requires 75-80% of the total execution time of CMT-Nek.
Spectral Elements: Derivatives and codes

Algorithm: dudr-4loop
\[
\begin{align*}
\text{do} & \quad k = 1, N_z \\
\text{do} & \quad j = 1, N_y \\
\text{do} & \quad i = 1, N_x \\
\text{do} & \quad l = 1, N_x \\
& \quad dudr(l, j, k) = dudr(l, j, k) + a(i, l) * u(l, j, k, ie)
\end{align*}
\]

Similarly, 5loop versions and 5loop-fused versions were considered.

Algorithm: dudr-4loop-fused
\[
\begin{align*}
\text{do} & \quad k = 1, N_z * N_y \\
\text{do} & \quad i = 1, N_x \\
\text{do} & \quad l = 1, N_x \\
& \quad dudr(l, k) = dudr(l, k) + a(i, l) * u(l, k, ie)
\end{align*}
\]

Optimizations

- **Autotuning**
  - Apply loop transformations
    - Loop permutation
    - Loop unroll
  - CHiLL applies loop transformation automatically on the target code

### Loop permutation

- Simple when you have perfect nesting

### Loop unroll

- Duplicate loop body
- Adjust loop header and data indexes
- May be applied to outer level loops too.
- Unroll factors are preferably divisors of the iteration space
- Reduces the number of limit checks for iterator
- Exposes possibility of vectorization to the back end compiler
  - \( c(i:i+4, j, k) = a(j, i:i+4) \times b(i:i+4, k) \)
- Code size increases, may result in higher L-cache miss rates
Possible Combinations

Algorithm: dudr-4loop

\[
\begin{align*}
&\text{do } k = 1, N_z \\
&\text{do } j = 1, N_y \\
&\text{do } i = 1, N_x \\
&\text{do } l = 1, N_x \\
&\quad \text{dudr}(l, j, k) = \text{dudr}(l, j, k) + a(i, l) \times u(l, j, k, i) \\
&\quad \text{enddo} \\
&\quad \text{enddo} \\
&\quad \text{enddo} \\
&\quad \text{enddo}
\end{align*}
\]

Number of implementations for \(N_x = N_y = N_z = 10\)

\[= 4! \times 4^4 \]

\[= 24 \times 256 = 6144 \text{ variants} \]

Total number of variants = 98240

CHiLL example

Algorithm: dudr-4loop-fused

\[
\begin{align*}
&\text{do } k = 1, N_z \times N_y \\
&\text{do } i = 1, N_x \\
&\text{do } l = 1, N_x \\
&\quad \text{dudr}(l, k) = \text{dudr}(l, k) + a(i, l) \times u(l, k, i) \\
&\quad \text{enddo} \\
&\quad \text{enddo} \\
&\quad \text{enddo}
\end{align*}
\]

\[
\begin{align*}
&\text{permute([2, 1, 3])} \\
&\text{unroll(0, 1, 1)} \\
&\text{unroll(0, 2, 2)} \\
&\text{unroll(0,3,5)}
\end{align*}
\]

CHiLL script

Input code

\[
\begin{align*}
&\text{DO } t2=1,10 \\
&\text{DO } t4=1,99,2 \\
&\text{DO } t6=1,6,5 \\
&\quad \text{dudr}(t2,t4) = \text{dudr}(t2,t4) + u(t6,t4,ie) \times a(t2,t6) \\
&\quad \text{dudr}(t2,t4+1) = \text{dudr}(t2,t4+1) + u(t6+t4+1,ie) \times a(t2,t6+1) \\
&\quad \text{dudr}(t2,t4) = \text{dudr}(t2,t4) + u(t6+1,ie) \times a(t2,t6+1) \\
&\quad \text{dudr}(t2,t4+1) = \text{dudr}(t2,t4+1) + u(t6+1+1,ie) \times a(t2,t6+2) \\
&\quad \text{dudr}(t2,t4) = \text{dudr}(t2,t4) + u(t6+2,ie) \times a(t2,t6+2) \\
&\quad \text{dudr}(t2,t4+1) = \text{dudr}(t2,t4+1) + u(t6+2+1,ie) \times a(t2,t6+3) \\
&\quad \text{dudr}(t2,t4) = \text{dudr}(t2,t4) + u(t6+3,ie) \times a(t2,t6+3) \\
&\quad \text{dudr}(t2,t4+1) = \text{dudr}(t2,t4+1) + u(t6+3+1,ie) \times a(t2,t6+3) \\
&\quad \text{dudr}(t2,t4) = \text{dudr}(t2,t4) + u(t6+4,ie) \times a(t2,t6+4) \\
&\quad \text{dudr}(t2,t4+1) = \text{dudr}(t2,t4+1) + u(t6+4+1,ie) \times a(t2,t6+4) \\
&\quad \text{ENDDO} \\
&\quad \text{ENDDO} \\
&\quad \text{ENDDO}
\end{align*}
\]

Output code
Genetic Algorithm

- We use genetic algorithms to search the exploration space efficiently.

- “Hello world!” example
  - Start with an arbitrary 12 character string
  - Goal is to generate “Hello world!”
  - Probability of coming up with target string in one try:
    - 1/95^12

A "Hello World!" Genetic Algorithm Example by James Matthews at http://www.generation5.org/content/2003/gahelloworld.asp

GA Characteristics

- Individuals
  - Each individual consists of a 12-letter string
  - Each individual has a fitness value

- Fitness function:
  - Sum of distance of each letter from target letter

- Population
  - Individuals make a population
  - Population size = 2048 for this problem
GA Operators

- Operators are used to create new individuals from existing ones.
  - Creates a new generation

Possible operations

- Crossover
  - $GWTc'kv2\%8@g$
  - $4K)\vM^pE`Yp$
  - **Result:** $GK)\vM^pE`Yp$

- Mutation
  - Random point mutation
    - $GK)\vM^pE`Zp$

Genetic Algorithm

**Algorithm:**

1. Initialize population
2. Do $i=1$, $max\_iter$
   a. calculate fitness of population
   b. sort population based on fitness
   c. print the member $p$ with the best fitness
   d. if $p$.fitness is 0 then break;
   e. apply crossover and mutation operations on pairs of members to create new population
3. Enddo
Genetic Algorithm

- **Output:**
  - Best: RV[S ‘yujj]p! (188)
  - Best: 8mKCrjvrsT& (153)
  - Best: Yakor7yilvg (132)
  - Best: GvthH$vrYU” (106)
  - Best: BiXpb wqwXg& (82)
  - Best: Sdmul0wqwXe’ (75)
  - Best: J]ndm"wqwvl% (53)
  - Best: J]ndm"wqwacl (43)
  - Best: J]ndm"wquag& (42)
  - Best: Chkmo"vtuq& (34)
  - Best: Gilho wpuig! (22)
  - Best: Hdmul wqaqmf” (21)
  - Best: Hdmul wqaqmf” (21)
  - Best: Ldnlp wqaqmf” (15)
  - Best: Ldnlp wqaqmf” (15)
  - Best: Jckop wqrc! (14)
  - Best: Hejip”wqlqg” (11)
  - Best: Ifklp wqrc! (9)
  - Best: Hejlm wprnc! (8)
  - Best: Jenlo wprld! (5)
  - Best: Genlo wprld! (4)
  - Best: Genlo wprld! (4)
  - Best: Hellp wormd” (3)
  - Best: Helko wprrld! (2)
  - Best: Helko wprrld! (2)
  - Best: Helko wprrld! (2)
  - Best: Helko wprrld! (2)
  - Best: Helko wprrld! (2)
  - Best: Helko wprrld! (2)
  - Best: Helko wprrld! (2)

- **Iterations:** 30
- **Total number of individuals studied:**
  \[ 30 \times 2048 = 61440 \]
GA (Our application)

- Individuals
  - Base code
  - Permutation sequence
  - Maximum of 5 unroll factors

| 5loop | 1 | 10 | "21435" | 5 | 2 | 10 |

Figure 9. Example of an individual

GA (Our application)

- Fitness function
  - Time
  - Energy
- Population size
  - 100 individuals
- Operators
  - Mutation
  - Crossover
**GA (Our application)**

- **Mutation**

  ![](mutation.png)

  **Figure 11. Mutation applied to individual P**

- **Crossover**

  ![](crossover.png)

  **Figure 12. Crossover operator applied to individuals R and S**
As a result of the mutation and crossover, certain incompatibilities may arise.

Example: Base code of individual P mutation to 4loop

Knowledge based GA: not 100% random
- Inclusion of target individual in the initial population
  - Target individual = CMT-Nek multiplication algorithm
- Crossover: Loop permutation has a greater role in performance, so inherit loop permutation sequence from the better performing parent
- Fixing incompatibilities in crossover operation
GA (Our application)

- Stopping criteria
  - The last three generations result in the same best individual
  - A pre-defined maximum number of iterations is reached
  - Improvement in performance of best individual compared to target individual is x%  
    - x is set dynamically
    - First 5 iterations, x = 60%
    - Next 5 iterations, x = 50%
    - ...
    - Next 5 iterations, performance of best individual is better than that of target individual
GA Output

- Matrix size 12:
  - Itr: 1 Best Algo: 4loop Unroll factors: 3 1 2 4 Permute: 4235 Fitness: 4.57044
  - Itr: 2 Best Algo: 4loopfused Unroll factors: 6 6 3 12 Permute: 423 Fitness: 3.33914
  - Itr: 3 Best Algo: 4loop Unroll factors: 3 1 1 2 Permute: 4235 Fitness: 3.15236
  - Itr: 4 Best Algo: 4loop Unroll factors: 3 1 1 2 Permute: 4235 Fitness: 3.15236
  - Itr: 5 Best Algo: 4loop Unroll factors: 3 1 1 2 Permute: 4235 Fitness: 3.15236

Performance And Energy

- Software Implementation:
  - CMT-Nek
  - 4loop version
  - 4loop-fused version
  - 5loop-version
  - 5loop-fused version

- CPU Platforms:
  - IBM Blue Gene/Q
  - AMD Opteron 6378

Performance and Energy Benchmarking of Spectral Element Solvers, Tania Banerjee and Sanjay Ranka (under preparation)
Architectures

- **BG/Q node**
  - Cores: 16
  - Each core:
    - 4-way SMT
    - 1.6 GHz
  - 204.8 GFLOPS peak performance
  - 55W peak power

- **Dell 6145 node**
  - 4 AMD Opteron CPU
  - Each CPU:
    - 16 cores
    - 2.4 GHz
  - 614.4 GFLOPS peak performance
  - 115 W peak power

---

**IBM BG/Q (Performance)**

- Matrix size 10x10x10, 100 elements
- 51% improvement versus CMT-Nek (~ 2 times)
- 34 GFLOPS average

- Comparable number of total nodes, but performance with 10x10x10 matrix size is 20% better than with 16x16x16

- Matrix size: 16x16x16, 25 elements
- 61% improvement versus CMT-Nek (~ 2.53 times)
- 12.7 GFLOPS average
AMD Opteron (Performance)

- Matrix size: 10x10x10, 100 elements
- GNU compilers
- 43% improvement versus CMT-Nek (~1.72 times)
- 209 GFLOPS
- Comparable number of total nodes, but performance with 10x10x10 matrix size is 10% better than with 16x16x16

- Matrix size: 16x16x16, 25 elements
- GNU compilers
- 42% improvement versus CMT-Nek (~1.73 times)
- 80 GFLOPS

Energy Measurements on IBM BG/Q

- Environmental Monitoring (EMON) APIs
- MonEQ wrapper library
  - Reports power consumption by domains
  - Utilizes interrupts for more frequent current/voltage readings
  - APIs:
    - MonEQ_Initilaize
    - MonEQ_Finalize,
    - MonEQ_StartPowerTag,
    - MonEQ_EndPowerTag

BG/Q Power Domains

- Power is measured on a node board basis
- Power Domains:
  - Core Logic power
  - Chip Memory Interface and SDRAM-DDR3
  - Optical module power
  - Optical module power + PCIExpress
  - HSS Network Transceiver power for Compute+Link Chip
  - Link Chip Core power
  - Core array power

Monitoring Power

- Power consumed by the basic dudt-4loop implementation for matrix size 10x10x10
- After the initial start, power consumption is constant for all domains
**IBM BG/Q (Energy)**

- **Energy Consumption**
  - Energy (Joules) vs. Runtime (seconds)

- **Performance**
  - Runtime (seconds) vs. Derivatives

**Observations:**
- Matrix size 10x10x10, 100 elements
- 55% reduction in energy versus CMT-Nek
**IBMBG/Q (Energy)**

- **Observations:**
  - matrix size 16x16x16, 25 elements
  - 56.8% reduction in energy versus CMT-Nek
  - Consumes 40% more energy compared to the 10x10x10 case

**GA Results**

- **Hipergator (Performance)**
- **Teller@Sandia (Energy)**
  - 104 nodes cluster
  - AMD-Fusion A10-5800K
  - 4 cores operating at 3.8GHz
  - Used PowerInsight to measure power

Results (Hipergator)

- Between 9.7% to 38.6% improvement, average improvement of 28.2%
- Maximum improvement for $N=12$

![Figure 22. Comparison of various compute times](image-url)
Results (SNL)

- Between 27% to 45% improvement average improvement of 37%
- Maximum improvement for $N=8$ and $N=12$

![Energy Comparison](image1.png)

- Between 23% to 45% improvement, average improvement of 34%
- Average power consumption is about the same for the various implementations across different matrix sizes
- Improvement in energy consumption heavily reflects improvement in runtime

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Time (s)</th>
<th>CPU Power (W)</th>
<th>Average Memory</th>
<th>Average Runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.235932</td>
<td>6.865028607</td>
<td>5.126008022</td>
<td>7.148349665</td>
</tr>
<tr>
<td>6</td>
<td>0.230774</td>
<td>7.420116651</td>
<td>5.336086388</td>
<td>7.148349665</td>
</tr>
<tr>
<td>7</td>
<td>0.261767</td>
<td>7.564760052</td>
<td>5.463265024</td>
<td>7.148349665</td>
</tr>
<tr>
<td>8</td>
<td>0.179687</td>
<td>7.620807293</td>
<td>5.12724905</td>
<td>7.148349665</td>
</tr>
<tr>
<td>9</td>
<td>0.34436</td>
<td>7.419096209</td>
<td>6.155995535</td>
<td>7.148349665</td>
</tr>
<tr>
<td>10</td>
<td>0.384240</td>
<td>7.088240727</td>
<td>5.716928264</td>
<td>7.148349665</td>
</tr>
<tr>
<td>11</td>
<td>0.418853</td>
<td>7.148349665</td>
<td>5.373871401</td>
<td>7.148349665</td>
</tr>
<tr>
<td>12</td>
<td>0.256378</td>
<td>7.113597891</td>
<td>5.18219192</td>
<td>7.148349665</td>
</tr>
<tr>
<td>13</td>
<td>0.463592</td>
<td>7.673234273</td>
<td>5.193316537</td>
<td>7.148349665</td>
</tr>
<tr>
<td>14</td>
<td>0.429746</td>
<td>6.986398774</td>
<td>5.354824051</td>
<td>7.148349665</td>
</tr>
<tr>
<td>15</td>
<td>0.437998</td>
<td>7.253166753</td>
<td>5.341765542</td>
<td>7.148349665</td>
</tr>
<tr>
<td>16</td>
<td>0.318481</td>
<td>7.122340108</td>
<td>5.399831701</td>
<td>7.148349665</td>
</tr>
</tbody>
</table>
Integration with CMT-Nek

Algorithm:
For all spectral elements do
- Compute dudr
- Compute duds
- Compute dudt
- Populate du for (x, y, z) coordinate system using dudr, duds and dudt above
Enddo

File: navier1.f
Subroutine: conv1
Inputs: du, u: where du represents a derivative matrix populated in the subroutine and u is the function matrix

GPU Architecture

Tesla K20c:
- 13 Processors
- 192 Cores
- 48k shared memory
- 64k registers
- 1170 GFLOP/s Peak
Optimizations:

– The derivative operator matrices $D$ and $D^T$ matrices are only brought once per block from the device memory to shared memory.
– The derivative operator matrices $D$ and $D^T$ are stored in registers instead of shared memory.


GPU Performance

Compared with

– CUGEMM
– ‘Combined’
  • Computation of $dudr$, $duds$ and $dudt$ happen in one kernel reusing function data as much as is possible.
– ‘Separate’
  • Computation of $dudr$, $duds$ and $dudt$ happen in separate kernels independent of each other.

Platform: Tesla K20c
**GPU (Performance)**

- **Observations**
  - Performance increases nearly linearly with matrix size
  - Over 180 GFLOPS for matrix size 16x16x16
  - 39% improvement versus CUGEMM for matrix size 16x16x16

**Energy Modeling**

- Nvidia-smi gives instantaneous power
- Run kernel thousands of times
- Measure every second
- Divide power consumed into GFLOPs
Observations:
- Power consumed was nearly similar for each kernel
- Hence performance/watt is dominated by performance results.

Hybrid processors: Performance and energy

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
<th>CPU Time (ms)</th>
<th>GPU Time (ms)</th>
<th>CPU Energy (mJ)</th>
<th>GPU Energy (mJ)</th>
<th>Total Time (ms)</th>
<th>Total energy (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000</td>
<td>0.0000</td>
<td>0.8333</td>
<td>0.0000</td>
<td>0.8333</td>
<td>0.8333</td>
<td>0.8333</td>
</tr>
<tr>
<td>10</td>
<td>9990</td>
<td>0.0333</td>
<td>0.8325</td>
<td>0.4167</td>
<td>0.8325</td>
<td>0.8325</td>
<td>1.2492</td>
</tr>
<tr>
<td>20</td>
<td>9980</td>
<td>0.0667</td>
<td>0.8317</td>
<td>0.8333</td>
<td>0.8317</td>
<td>0.8317</td>
<td>1.6650</td>
</tr>
<tr>
<td>40</td>
<td>9960</td>
<td>0.1333</td>
<td>0.8300</td>
<td>1.6667</td>
<td>0.8300</td>
<td>0.8300</td>
<td>2.4967</td>
</tr>
<tr>
<td>80</td>
<td>9920</td>
<td>0.2667</td>
<td>0.8267</td>
<td>3.3333</td>
<td>0.8267</td>
<td>0.8267</td>
<td>4.1600</td>
</tr>
<tr>
<td>160</td>
<td>9840</td>
<td>0.5333</td>
<td>0.8200</td>
<td>6.6667</td>
<td>0.8200</td>
<td>0.8200</td>
<td>7.4867</td>
</tr>
<tr>
<td>320</td>
<td>9680</td>
<td>1.0667</td>
<td>0.8067</td>
<td>13.3333</td>
<td>0.8067</td>
<td>1.0667</td>
<td>14.1400</td>
</tr>
<tr>
<td>640</td>
<td>9360</td>
<td>2.1333</td>
<td>0.7800</td>
<td>26.6667</td>
<td>0.7800</td>
<td>2.1333</td>
<td>27.4467</td>
</tr>
<tr>
<td>1280</td>
<td>8720</td>
<td>4.2667</td>
<td>0.7267</td>
<td>53.3333</td>
<td>0.7267</td>
<td>4.2667</td>
<td>54.0600</td>
</tr>
<tr>
<td>2560</td>
<td>7440</td>
<td>8.5333</td>
<td>0.6200</td>
<td>106.6667</td>
<td>0.6200</td>
<td>8.5333</td>
<td>107.2867</td>
</tr>
</tbody>
</table>
Interpolation

- **Optimizations:**
  - Matrix multiplication
  - Ordering of operations
    - Expensive operations can be done earlier while matrix size is still small
    - \(xyz, xzy, yxz, yzx, zxy, zyx\)

---

**Lightweight Distributed Metric Service**

- **LDMS:** Data collection tool at LANL
- **Gives temperature numbers.**
- **After logging in,**
  
  ```
  startJobNodes -m procsensors -s /turquoise/users/banerjee
  ```

- **Output:**
  
  ```
  #Time, Time_usec, Compld, temp1_1b.3, temp1_1a.3, temp1_19.3, temp1_18.3
  1421325154.002116, 2116, 1600, 30, 30, 30, 32
  ```

- **Units of temperature - degree Celsius**
- **Sensors are placed inside core.**
Conclusions

- We benchmarked the most compute intensive kernel of CMT-Nek for performance and energy.
- Our work highlights autotuning as an important strategy for improving both performance and energy, over different architectures
  - We got between 42-61% improvement in performance and about 55% improvement in energy requirement
- We coupled genetic algorithms with autotuning for performing smart search
- Currently working on spectral interpolation and temperature measurements

Dynamic Voltage Scaling

- \( P_{\text{dyn}} \propto V_{dd}^2 \cdot f \)
- \( P_{\text{sta}} \sim \propto V_{dd} \)
- \( f \propto (V_{dd} - V_{th})^\alpha, \alpha = 1.5 \)
Reconfigurable Cache

Capacity Tuning

Zhang et al., ACM TECS 2005

Our Research

System Modeling

DCR
Scheduling-aware Cache Reconfiguration

Multi-level Cache Tuning

Cache Reconfiguration and Partitioning in Multi-Core Systems

DVS
Static Slack Allocation

Dynamic Slack Reclamation

Energy- and Temperature-Constrained Scheduling

Energy Optimization

DCR + DVS

System-wide Energy Optimization

Zhang et al., ACM TECS 2005
Optimal Cache Configurations

### CJPEG

<table>
<thead>
<tr>
<th></th>
<th>I-Cache</th>
<th>D-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>4KB, 2W, 16B</td>
<td>4KB, 4W, 16B</td>
</tr>
<tr>
<td>C1</td>
<td>4KB, 2W, 16B</td>
<td>4KB, 4W, 32B</td>
</tr>
<tr>
<td>C2</td>
<td>2KB, 2W, 16B</td>
<td>4KB, 4W, 16B</td>
</tr>
<tr>
<td>C3</td>
<td>2KB, 2W, 16B</td>
<td>4KB, 4W, 16B</td>
</tr>
</tbody>
</table>

### RAWCAUDIO

<table>
<thead>
<tr>
<th></th>
<th>I-Cache</th>
<th>D-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>1KB, 1W, 16B</td>
<td>4KB, 2W, 64B</td>
</tr>
<tr>
<td>C1</td>
<td>1KB, 1W, 16B</td>
<td>2KB, 2W, 16B</td>
</tr>
<tr>
<td>C2</td>
<td>1KB, 1W, 16B</td>
<td>4KB, 4W, 16B</td>
</tr>
<tr>
<td>C3</td>
<td>1KB, 1W, 16B</td>
<td>4KB, 2W, 16B</td>
</tr>
</tbody>
</table>

### A2TIME01

<table>
<thead>
<tr>
<th></th>
<th>I-Cache</th>
<th>D-Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>4KB, 4W, 16B</td>
<td>4KB, 4W, 16B</td>
</tr>
<tr>
<td>C1</td>
<td>4KB, 4W, 16B</td>
<td>4KB, 4W, 16B</td>
</tr>
<tr>
<td>C2</td>
<td>4KB, 4W, 16B</td>
<td>4KB, 4W, 16B</td>
</tr>
<tr>
<td>C3</td>
<td>4KB, 4W, 16B</td>
<td>4KB, 4W, 16B</td>
</tr>
</tbody>
</table>

---

Cache Reconfiguration in Multi-core Systems

Dynamic cache reconfiguration in L1 caches and partitioning in L2 cache are highly correlated.

**Algorithm**

- We assume task mapping is given
- We statically profile each independent task for all L1 cache configurations and L2 cache partitioning factors
  - Greatly reduced design space size
- Step 1: We employ a dynamic programming based algorithm to find the optimal L1 cache configurations for each core (with multiple tasks) separately under all L2 partition factors
- Step 2: We then find the optimal L2 cache partition scheme
Our approach can achieve 29% energy savings compared with CP and up to 14% savings compared with DCR + UCP.
Thermal Issues of Multi-core Processors

- The power density of multi-core processors has doubled every three years and this rate is expected to increase as frequencies scale faster than operating voltages.
- A small increase of 10 C in temperature may result in 2 × reductions in the lifespan of the device.
- The cost of cooling system increases super-linearly in power consumption.

Managing Temperature: Motivation

Temperature varies on multiple cores.

Tilera Processor
Thermal RC model

\[ T(t) = P \times R + T_A - (P \times R + T_A - T_i)e^{-\frac{t}{RC}} \]

- **P**: power consumption
- **T_i**: initial temperature
- **t**: execution time

There are three major factors affecting the on-chip transient temperature: average power of the processor, initial temperature and execution time

Thermal Management

- **Sensor based**
  - [Mukherjee06, Sharifi08, Cochran09]
  - **Pros**: low computation overhead
  - **Cons**: imprecision and noise from the raw data of temperature sensor; fixed position of the sensor

- **Model based**
  - [Skadron03, Huang04, Liu06, Rao07]
  - **Pros**: estimate the temperature accurately
  - **Cons**: high computation overhead
Assume $P$ is the power of a thermal element, $T_i$ is the initial temperature and $T_A$ is the ambient temperature, then the transient temperature at time $t$ is:

$$T_i = P \times R + T_A - (P \times R + T_A - T_i) \cdot e^{-t/RC}$$

Assume $t \to \infty$, we get steady state temperature:

$$T_s = P \times R + T_A$$

Time to reach steady state temperature: 20ms – 20s
- $G$: thermal conductance between thermal elements (cores and heatsinks)
- $A$: thermal conductance between thermal elements and outside environment
- Thermal conductance is the quantity of heat transferred with temperature difference of one kelvin, measured in W/K

Matrix Model

- Suppose there are $u$ cores and $v$ heatsinks, $T_m$ is the steady-state temperature of thermal element $m$, $T_A$ is the ambient temperature.

\[
\sum_{m \in M} (T_m - T_n) \cdot G(m,n) + (T_m - T_A) \cdot A(m) = P(m), \quad \forall m \in M
\]

\[
\Rightarrow R \cdot T = T_A \cdot A + P \quad (1)
\]

where $R_{ij} = \begin{cases} 
-G(i, j), & \text{if } i \neq j \\
\sum_{k=0, k \neq i}^{u+v-1} G(i, k) + A(i), & \text{if } i = j 
\end{cases}$
Matrix Model

- In a multi-core processor with \( u \) cores

\[
T_{\text{core}} = T_A \cdot I + C \cdot P
\]

- \( T_{\text{core}} \), \( 1 \times u \) vector of steady-state temperature of cores
- \( T_A \), ambient temperature, \( I = [1, 1, \ldots, 1]^T \)
- \( C \), \( u \times u \) matrix
  - The amount of change in temperature of \( m^{th} \) core caused by \( n^{th} \) core is given by the \( C_{m,n} \) times the change in thermal power of \( n^{th} \) core
- \( P \), \( 1 \times u \) vector of power of cores

Matrix Model

- Generation of matrix \( C \)

\[
\begin{bmatrix}
P_0, P_1, \ldots, P_{u-1} \\
P_0 + \alpha, P_1, \ldots, P_{u-1}
\end{bmatrix}
\rightarrow\text{HotSpot} \rightarrow
\begin{bmatrix}
T_0, T_1, \ldots, T_{u-1} \\
T'_0, T'_1, \ldots, T'_{u-1}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
C_{00}, C_{01}, \ldots, C_{0u-1} \\
C_{10}, C_{11}, \ldots, C_{1u-1} \\
\vdots \\
C_{u0}, C_{u1}, \ldots, C_{uu-1}
\end{bmatrix}
\]

\[
\begin{pmatrix}
C_{00} & C_{01} & C_{02} & \ldots & C_{0u-1} \\
C_{10} & C_{11} & C_{12} & \ldots & C_{1u-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
C_{u0} & C_{u1} & C_{u2} & \ldots & C_{uu-1}
\end{pmatrix}
\]
Matrix Model

- Generation of matrix $C$

\[
\begin{align*}
[P_0, P_1, \ldots, P_{u-1}] & \rightarrow \text{HotSpot} \rightarrow [T_0, T_1, \ldots, T_{u-1}] \\
[P_0, P_1+\alpha, \ldots, P_{u-1}] & \rightarrow \text{HotSpot} \rightarrow [T'_0, T'_1, \ldots, T'_{u-1}] \\
\text{subtract} & \\
\end{align*}
\]

\[
\begin{bmatrix}
C_{00} & C_{01} & C_{02} & \cdots & C_{0u-1} \\
C_{10} & C_{11} & C_{12} & \cdots & C_{1u-1} \\
C_{20} & C_{21} & C_{22} & \cdots & C_{2u-1}
\end{bmatrix}
\]
Matrix Model

- Generation of matrix $C$

\[
\begin{align*}
[P_0, P_1, ..., P_{u-1}] & \quad \xrightarrow{\text{HotSpot}} \quad [T_0, T_1, ..., T_{u-1}] \\
[P_0, P_1, ..., P_{u-1}+\alpha] & \quad \xrightarrow{\text{HotSpot}} \quad [T'_0, T'_1, ..., T'_{u-1}]
\end{align*}
\]

\[
\begin{bmatrix}
C_{00} & C_{01} & C_{02} & \cdots & C_{0u-1} \\
C_{10} & C_{11} & C_{12} & \cdots & C_{1u-1} \\
C_{20} & C_{21} & C_{22} & \cdots & C_{2u-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
C_{u-10} & C_{u-11} & C_{u-12} & \cdots & C_{u-1u-1}
\end{bmatrix}
\]

\[\text{subtract} \quad u\]

C-Matrix of a 4 core processor

<table>
<thead>
<tr>
<th></th>
<th>Core0</th>
<th>Core1</th>
<th>Core2</th>
<th>Core3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core0</td>
<td>3.66</td>
<td>1.56</td>
<td>1.56</td>
<td>1.31</td>
</tr>
<tr>
<td>Core1</td>
<td>1.56</td>
<td>3.66</td>
<td>1.31</td>
<td>1.56</td>
</tr>
<tr>
<td>Core2</td>
<td>1.56</td>
<td>1.31</td>
<td>3.66</td>
<td>1.56</td>
</tr>
<tr>
<td>Core3</td>
<td>1.31</td>
<td>1.56</td>
<td>1.56</td>
<td>3.66</td>
</tr>
</tbody>
</table>

Fig. 2. $C$ matrix of a 4-core processor
Matrix Model

- Compact
  - Quite simple equations to compute the steady-state temperature of cores, much easier than HotSpot
- Accurate
  - Achieve very close steady-state temperature to HotSpot (Experiments)
- Limitations
  - Only steady-state temperature
  - Still need HotSpot to generate C matrix

Evaluation

- Matrix C generation
  - Use 20 different power configurations to generate 20 different matrix Cs and average them
- DAG generation:
  - 32, 64, 128, 256, 512 tasks
  - execution time, 20 – 60 time units
  - probability of any two node having an edge between them is set to 0.1

- Multicore parameters:
  - 4 cores and 16 cores processors
  - Each core is abstracted as a 8mm × 8mm square chip
  - For 4 cores processors, each core dissipates 100W power at maximum voltage. For 16 cores processor, 50W each core
  - The default thermal configuration in HotSpot.
Evaluation

- Matrix Model vs HotSpot --- Peak Temperature

![Graph showing peak temperature vs number of slacks]

- Matrix Model vs HotSpot -- Computation Time

![Graph showing computation time vs number of tasks]
Temperature Aware Scheduling

- Uniform voltage without throttling
  - Each core can work at full voltage or has to be shut off
- Uniform voltage with throttling
  - The voltage of each core has to be the same but can be varied between a maximum and minimum
- Non-uniform voltage with throttling
  - The voltage of each core can be varied independently

Problem: For each of above three types of processors, determine the workload distribution for each core, so that the total throughput across all cores is maximized and the maximum temperature for any core is bounded by a given threshold

Uniform voltage without throttling

- Determine data parallel workloads distribution on multicore processor, so that the total throughput across all cores is maximized and the maximum temperature for any core is bounded by a given threshold

\[
\max \sum_{i=1}^{m} \frac{w_i}{\max t_i} \\
\text{s.t. } T_i \leq T_{th}, \ i \in \{1,2,\ldots,m\}
\]

- \(w_i\) denotes workloads assigned on core \(i\);
- \(t_i\) denotes the running time of the workload on core \(i\);
- \(T_i\) denotes the peak temperature on core \(i\);
- \(T_{th}\) denotes the temperature threshold for all cores
Uniform voltage without throttling

\[
\begin{align*}
\text{max} \quad & z = \sum_{i \in M} x_i \\
\text{s.t.} \quad & T_A + \sum_{j \in M} C_{ij} \cdot x_j \cdot P \leq T_{th}, \quad \forall i \in M \\
& x_i = 0 \text{ or } 1, \quad \forall i \in M
\end{align*}
\]

Uniform voltage with throttling

Optimization Problem

\[
\begin{align*}
\text{max} \quad & x \\
\text{s.t.} \quad & T_A + \sum_{j \in M} C_{ij} \cdot x_j \cdot P \leq T_{th} \quad (*) \\
& 0 \leq x \leq 1 \\
& \text{Simplify (*) as follows:}
\end{align*}
\]

\[
\begin{align*}
& x \leq \frac{D}{\sum_j C_{ij}}, \quad \forall i \in M
\end{align*}
\]

where \(D\) is equal to \(\frac{T_{th} - T_A}{P}\)

\[
\begin{align*}
& x \text{ is given by:}
\end{align*}
\]

\[
\begin{align*}
& x = \min \left\{ \frac{\min_i \left( \frac{D_i}{\sum_j C_{ij}} \right)}{1}, \quad \forall i \in M
\end{align*}
\]
Non-uniform voltage with throttling

Optimization Problem
\[
\text{max} \quad \sum_{i \in M} x_i \\
\text{s.t.} \quad T_A + \sum_{j \in M} C_{ij} \cdot x_j \cdot P \leq T_{th}, \quad \forall i \in M \quad (**)
\]
\[
0 \leq x_i \leq 1, \quad \forall i \in M \quad (***)
\]

Three possible cases:

- Case 1: The threshold temperature is high and all cores can execute at its maximum voltage without exceed the threshold. In this case all $x_i$s are set to 1s.
- Case 2: The threshold temperature is low and requires all $x_i$s to be less than 1. In this case these $x_i$s are all bounded by Equation (**).
- Case 3: The threshold temperature is such that $x_i$ values of some of the cores is limited by the constraint given by equation (***)
Non-uniform voltage with throttling

- **Case 2**

\[
\begin{align*}
\text{max} & \quad \sum_{i \in M} x_i \\
\text{s.t.} & \quad T_A + \sum_{j \in M} C_{ij} \cdot x_j \cdot P \leq T_{th}, \quad \forall i \in M \quad (**)
\end{align*}
\]

\[
0 \leq x_i \leq 1, \quad \forall i \in M \quad (***)
\]

\[
\Rightarrow \quad \text{max} \quad \sum_{i \in M} x_i
\]

\[
\text{s.t.} \quad C \cdot X \leq D
\]

\[
0 \leq X \leq 1
\]

\[
\Rightarrow \quad X = C^{-1}D
\]

Non-uniform voltage with throttling

- **Case 3:**

  - Check Case 1 followed by Case 2. If both fails, than an approximation is used as in Case 3 by assuming that all \(x_i\) values are the same and the algorithm for uniform voltage with throttling
Evaluation

- CPU: multi-core processors with 4, 16, 32 and 64 cores
- The ambient temperature used was 45.5°C. The maximum allowable temperature was set to 70°C

Effective Number of Cores =

Metric: \[
\begin{array}{|c|c|c|c|}
\hline
\text{Number of cores} & \text{Floorplan} & \text{Area per core} & \text{Maximum power per core} \\
\hline
4 & 2\times2 \text{ grid} & 8\text{mm}\times8\text{mm} & 40\text{Watt} \\
16 & 4\times4 \text{ grid} & 4\text{mm}\times4\text{mm} & 10\text{Watt} \\
32 & 8\times4 \text{ grid} & 3\text{mm}\times3\text{mm} & 5\text{Watt} \\
64 & 8\times8 \text{ grid} & 2\text{mm}\times2\text{mm} & 2.5\text{Watt} \\
\hline
\end{array}
\]

Evaluation

- Uniform voltage without throttling
  - MIP (Mixed Integer Programming): The solution derived by our algorithm that minimizes the maximum temperature for the optimal value of P
  - BestP: Consider all subsets of size P. Find the subset that corresponds to the lowest maximum temperature
  - BestP+1: Consider all subsets of size P+1. Find the subset that corresponds to the lowest maximum temperature
  - WorstP: Consider all subsets of size P. Find the subset that corresponds to the highest maximum temperature
Evaluation

- **Uniform voltage without throttling**

  ![Graph showing uniform voltage without throttling]

- **Uniform voltage with throttling**
  
  - Throughput comparison

  ![Graph showing throughput comparison]
Evaluation

- **Uniform voltage with throttling**
  - Computation time comparison

- **Non-uniform voltage with throttling**
  - Throughput comparison
Evaluation

- Non-uniform voltage with throttling
  - Computation time

![Graph showing computation time vs. number of cores]

Decomposing Hot Tasks

- Partition the “hot” tasks into multiple subtasks and interleave these subtasks with “cool” tasks to reduce the overall maximum temperature
  - To the best of our knowledge, our work is the first attempt to develop efficient task partitioning algorithms to demonstrate significant temperature reduction.

- Several heuristic task partitioning algorithms using “cool” tasks to interleave “hot” tasks
  - 1) for a periodic set of tasks with common period
  - 2) for a periodic set of tasks with individual period

1We define “hot” tasks as tasks with higher average power consumption, and “cool” tasks as tasks with lower average power consumption.
Related Work

- Dynamic Voltage and Frequency Scaling (Reduce power)
  - Dynamic Voltage and Frequency Scaling (DVFS) can be used to reduce the power consumption by lowering the supply voltage and operating frequency, thereby reduce the on-chip temperature [Brooks2001, Rao2008, Kadin2008, Ebi2009, ...]
  - **Cons:** faces a serious problem in time-constrained applications

- Temperature aware task sequencing algorithm (Reduce initial temperature)
  - Reduce peak temperature compared to a random sequence [Jayaseelan2008]
  - **Cons:** fails to reduce temperature in cases when one or more of the “hot” tasks are long

Temperature-aware task partitioning algorithm

Illustrative example

![Illustrative example diagram]
Illustrative example

Temperature-aware task partitioning algorithm
Temperature-aware task partitioning algorithm

Illustrative example

Task Sequencing

Task Partitioning

Transient Temperature (Celsius)

Time (ms)
Temperature-aware task partitioning algorithm

Illustrative example

Task Partitioning Algorithm can achieve lower peak temperature than Task Sequencing Algorithm.
Experiments

- **Platform:**
- **CPU:**
  - ARM Cortex A8 (Simplescalar)
  - 2-width in-order issue, 32KB instruction cache
  - 1.5GHz clock speed
- **Power simulator:**
  - Wattch
- **Temperature evaluation:**
  - HotSpot
  - Ambient temperature:
- **Tasks:** Synthetic tasks and real benchmarks are used
- **Algorithms:** Compare with task sequencing algorithm and EDF algorithm.

Experiments

```
T(t) = P \times R + T_A - (P \times R + T_A - T) e^{-\frac{t}{RC}}
```

Temperature-aware task partitioning algorithms

Peak temperature
Experiments: Periodic tasks with common period

Real benchmarks:

<table>
<thead>
<tr>
<th>Set</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 1</td>
<td>patricia, adpcm, rijndael, susan, crc, FFT, dijkstra, epic</td>
</tr>
<tr>
<td>Set 2</td>
<td>patricia, djpeg, adpcm, sha, FFT, rijndael, susan, rijndael</td>
</tr>
<tr>
<td>Set 3</td>
<td>sha, djpeg, FFT, rijndael, dijkstra, epic, rijndael, susan</td>
</tr>
<tr>
<td>Set 4</td>
<td>rijndael, dijkstra, FFT, gsm, sha, patricia, pegwit, djpeg</td>
</tr>
</tbody>
</table>

Temperature comparison:

Temperature comparison: Task partitioning algorithm (TPA) can reduce the peak temperature by up to 5.8°C compared with task sequencing algorithm (TSA)
Periodic tasks with individual period

EDF scheduling with task partitioning (EDFp) can also reduce the peak temperature by up to 6°C.

Experiments

- **Overhead:**

Average context switch per task for TPA is as low as 2 (left figure).
Average context switch per task for EDFp is also lower than 2 (right figure). They are tolerable in many practical scenarios.

\(^1\) Context switch time on ARM cpu can be less than 10us [SEGGER]
Conclusion

- We propose to partition the “hot” tasks into multiple subtasks and interleave these subtasks with “cool” tasks to reduce the overall maximum temperature.

- We propose two heuristic task partitioning algorithms using “cool” tasks to interleave “hot” tasks:
  - 1) for a periodic set of tasks with common period
  - 2) for a periodic set of tasks with individual period

Temperature-aware Scheduling for Multicores

- Multicore Processors:
  - Multiple heating sources
  - Heat interaction between neighboring cores

Legend:
- Thermal Resistance
- Thermal Capacitance
- Power Source

Diagram showing the interconnection of cores with thermal resistors and capacitors.
Inter Core Scheduling

Experiments

- **Platform:**
- **CPU:**
  - Simplescalar, ARM Cortex A9 (multicore)
  - 2-width out-of-order issue, 32KB instruction cache
  - 1.2GHz clock speed.
- **Power simulator:**
  - Wattch
- **Temperature evaluation:**
  - Temperature simulator: HotSpot
  - Ambient temperature: 45.15°C
- **Tasks:** Synthetic tasks and real benchmarks are used
- **Algorithms:** Min-Min, PDTM [Yeo2008], TPS-1($\delta=0.33$ms), TPS-2($\delta=0.66$ms), TPS-3($\delta=1.32$ms), TPS-3($\delta=2.64$ms)
Experiments

- Multicore:
  - Real benchmarks

**The Four Sets of Benchmark Tasks**

<table>
<thead>
<tr>
<th>set</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>set1</td>
<td>dijkstra, patricia, adpcm, crc32, fft</td>
</tr>
<tr>
<td></td>
<td>gsm, ghostscript, ejpeg, blowfish, stringsearch, sha</td>
</tr>
<tr>
<td>set2</td>
<td>epic, unepic, rasta, epegwit, ifft</td>
</tr>
<tr>
<td></td>
<td>dpegwit, ghostscript, dadpcm, ungsm</td>
</tr>
<tr>
<td></td>
<td>blowfish, susan, basicmath</td>
</tr>
<tr>
<td>set3</td>
<td>dijkstra, adpcm, crc32, fft, gsm, rasta</td>
</tr>
<tr>
<td></td>
<td>ghostscript, jpeg, patricia, rijndael</td>
</tr>
<tr>
<td></td>
<td>blowfish, stringsearch, susan, sha, epic, pegwit</td>
</tr>
<tr>
<td>set4</td>
<td>stringsearch, rasta, unepic, dadpcm, ifft</td>
</tr>
<tr>
<td></td>
<td>susan, c, ghostscript, dpegwit, blowfish</td>
</tr>
<tr>
<td></td>
<td>crc32, dijkstra, djpeg, sha</td>
</tr>
</tbody>
</table>
Experiments

- Multicore: Synthetic tasks

TPS algorithm reduces the peak temperature by up to 11.68°C compared with Min-Min algorithm.
PDTM can achieve similar peak temperature reduction, but requires 33% more makespan.

Experiments

- Multicore:
  - Real benchmarks

TPS algorithm reduces the peak temperature by up to 9.92°C compared with Min-Min algorithm, 4.52°C compared with PDTM algorithm.
PDTM can achieve similar peak temperature reduction, but requires 44% more makespan.
Experiments

- Multicore:
  - TPS vs PDTM: peak temperature undersame makespan

TPS-1 relaxed algorithm reduce the peak temperature by up to 20°C compared with PDTM algorithm.

- Multicore:
  - TPS vs PDTM: scalability
Conclusions

Thermal model addressed core throttling can highly improve performance

Heuristics with transient thermal model have better improvements than methods with steady-state model

Computing time cost: transient is larger than steady-state
  – It is worthwhile because it is calculated offline.

Different initial configurations on f and t may result in small difference of achievements
  – Practically it will be better by using the steady-state solution as initial configuration

Conclusion

- We propose to partition the “hot” tasks into multiple subtasks and interleave these subtasks with “cool” tasks to reduce the overall maximum temperature

- We propose heuristic task partitioning algorithms using “cool” tasks to interleave “hot” tasks on both single core and multicore processors

- Experimental results show that our algorithm outperforms existing state-of-art thermal-aware scheduling algorithm in terms of peak temperature and makespan.
Transient Thermal models

Steady-state thermal model
- \( T(t) = T_A + G^{-1}P \)
- Efficient but does not capture transient effects (worst case scenario)

Transient-state thermal model:
- If the average power of core is \( P \) over a time period \( t \), then the temperature at the end of this period \( T(t) \) is given by:
  \[
  T(t) = T_A + e^{-G^{-1}Ct}(T_i - T_A) + G^{-1}(I - e^{-G^{-1}Ct})P
  \]

\( G \) is the thermal conductance matrix
\( C \) is the thermal capacitance matrix
\( T_A \) is the ambient temperature
\( T_i \) is the initial temperature

Approach

1. We propose a solution to the convex optimization problem with the simple thermal model to solve the problem of maximizing throughput under the temperature constraint.
2. We also propose a heuristic algorithm with the transient thermal model to solve the problem with higher accuracy.
Evaluation – Matrix Multiplication

General scheme
- High throughput improvement than w/o HLB
- Around 10% throughput improvement than base solution
- With very large workload, solutions of heuristic and base will converge

Homogeneous-scaling scheme

Hengxing Tan, and Sanjay Ranka, Thermal-aware Scheduling for Data Parallel Workloads on Multi-Core Processors, ISCC 2014 (Work partially supported by NSF)

Future Work: Energy and Thermal Management

- Varying Architectural Elements
  - Processor (Dynamic Voltage Scaling)
  - Caches (Dynamic Cache Reconfiguration)
  - Buses
  - Memory
- Developing Optimized Libraries
  - Energy
  - Performance
  - Temperature
Managing Temperature: Approaches

High temperature leads to performance loss.

Temperature Thresholding -

- Lower workloads at $T_{max}$
- Increase workloads at $T_1$

Zigzag throttling processor speeds

- Zigzag effects cause more loss
- Processor will put “hot” cores into low power state.

[Figure: Zigzag temperature thresholding]

[Rajan2008]
Modeling Thermal Behavior (HotSpot)

Legend:
- dotted lines: shorting connected resistances
- 
  - node
- 
  - thermal resistance
- 
  - thermal capacitance

Energy Levers

Core 1
IL1
DL1

Core 2
IL1
DL1

Core 3
IL1
DL1

Core m
IL1
DL1

L2 Cache
To Memory

8 ways in one cache set
Relaxation

- Most real architectures only support discrete frequency settings to scale
  - E.g. Given options $\{1.0 \text{ GHz}, 1.5 \text{ GHz}, 2.0 \text{ GHz}\}$ for dual cores
  - Results: $\langle 1.4 \text{ GHz}, 1.99 \text{ GHz} \rangle$
- Relaxation
  - Native: downward
  - relaxation
    - $\langle 1.0 \text{ GHz}, 1.5 \text{ GHz} \rangle$
    - Our method: relaxing result frequency to the neighboring discrete value
      - $\langle 1.0 \text{ GHz}, 2.0 \text{ GHz} \rangle$
      - Practically choose among 2 or 4 neighbors

Solution with the steady state model

Assumption:

- Applications runs for a long time with constant frequency
- Each core completes work simultaneously

Each core will arrive at its steady-state temperature

\[
\begin{align*}
\max & \quad \beta \sum_{i=1}^{m} f_i \\
\text{s.t.} & \quad \sum_{i=1}^{m} G^{-1} i f_i^3 \leq T_m - T_i, \\
 & \quad F_{\min} \leq f_i \leq F_{\max}, \quad i \in \{1,2,\ldots,m\}
\end{align*}
\]

Use convex solver
Related Dual Objective

Minimize the Makespan across all cores with given workloads and temperature threshold

\[
\begin{align*}
\min & \quad \max t_i \\
\text{s.t.} & \quad \beta f_i t_i \leq W \\
                & \quad T_i \leq T_{\text{th}} \quad , \quad i \in \{1,2,\ldots,m\}
\end{align*}
\]

A general solution uses non-linear solver as SQP

Iterative Refinement Process

1) Derive the core with the maximum peak temperature. Let this core be given by \(L\).
2) Compute the new peak temperature on every core if workload of amount \(\delta w\) is moved from core \(L\) to one of the other core \(J\). This is done while ensuring that the execution time on any core is not reduced. Only the frequency of the destination core is potentially modified. Among all possible choices of \(J\), the one that minimizes the peak temperature among all cores is chosen. Computation of this step is facilitated by using a matrix \(G^f\), where \(G^f_{i,j} = \partial T_i / \partial f_j\), \(i,j \in \{1,2,\ldots,m\}\).
3) The workload is moved from core \(L\) to core \(J\). The temperature vector \(T\) is updated using \(G^f\).
Additional constraints

Homogeneous-scaling: cores run at same frequency

Objective:
\[
\max \beta \sum_{i=1}^{m} f t_i / \max t_i
\]

Non-scaling: cores run at fixed frequency but some cores could be turned off

Objective:
\[
\max \beta F \sum_{i=1}^{m} t_i / \max t_i
\]

Additional Constraints (with simple model)

Homogeneous-scaling: cores run at same frequency

\[
\begin{align*}
\max & \quad \beta f \\
\text{s.t.} & \quad \sum_{i=1}^{m} G^{-1} f i^3 \leq T_m - T_i, \\
& \quad F_{\min} \leq f \leq F_{\max}, \quad i \in \{1,2,\ldots,m\}
\end{align*}
\]

– Convex problem

Non-scaling: cores run at fixed frequency

\[
\begin{align*}
\max & \quad \beta F \sum_{i=1}^{m} x_i \\
\text{s.t.} & \quad F^{-1} \sum_{i=1}^{m} G^{-1} x_i \leq T_{\text{th}} - T_i, \\
& \quad x_i = 0 \text{ or } 1, \quad i \in \{1,2,\ldots,m\}
\end{align*}
\]

– Mixed Integer linear problem
Homogeneous-scaling: cores run at same frequency

\[
\begin{align*}
\min & \quad \max t_i \\
\text{s.t.} & \quad \beta f \sum_{i=1}^{m} t_i = W, \\
& \quad T_i \leq T_{th}, \\
& \quad F_{\min} \leq f \leq F_{\max}, \quad i \in \{1,2,\ldots,m\}
\end{align*}
\]

Non-scaling: cores run at fixed frequency but some cores could be turned off

\[
\begin{align*}
\min & \quad \max t_i \\
\text{s.t.} & \quad \sum_{i=1}^{m} t_i = \frac{W}{\beta F_{\max}}, \\
& \quad T_i \leq T_{th}, \\
& \quad i \in \{1,2,\ldots,m\}
\end{align*}
\]

The heuristic can work on both problems

---

**Heuristic – local search**

Based on Heat Load Balance (HLB)

Slicing workloads and move then from hot cores to cool cores

1: Start from an initial configuration including f and t
   - Distribute total workloads evenly to cores with \( F_{\max} \)

2: Move a workload unit
   - From: the core decided by peak temperature \( T_p \)
   - To: the core decided by max gradient \( \partial T / \partial f \), in term of frequency slice

3: Repeat step 2 until same peak temperature \( T_p \) on all cores

4: If \( T_p < T_{th} \)
   - Continue move workload unit until \( T_p = T_{th} \) (From: decided by gradient \( \partial T / \partial t \))

Else if \( T_p > T_{th} \)
   - Backward move workload unit until \( T_p = T_{th} \) (reducing f by increase t)
Temperature-aware task partitioning algorithm

- Two major challenges:
  - 1) Number of Partitions
  - 2) Sequencing of Subtasks

- Two broad scenarios:
  - 1) A periodic set of tasks with common period. All the tasks have the same arrival time and deadline.
  - 2) A set of periodic tasks with individual period. Each task may have different arrival time and deadline.
Scenario 1: Periodic tasks with common period

- A periodic set of $N$ heterogeneous tasks $L$, let $P_i$ be the average power consumption during the execution time $c_i$ of task $\tau_i$.
- The goal is to find a sequence of these tasks using task partitioning to minimize the peak temperature.

Algorithm: Periodic tasks with common period

- Sort the tasks based on the power profile from coolest to hottest.
- Group the sorted tasks into $k$ categories with equal number of tasks.
- Partition tasks in category $j$, $2 \leq j \leq k$, into $2^{i-1}$ equal subtasks. Partition tasks in category 1 into 2 equal subtasks.
- **for** $i = 1$ to $k - 1$ **do**
  - Interleave tasks of $i$th category with tasks of $(i+1)$th category to form the new $(i+1)$th category
- **end for**
Periodic tasks with common period

3: T2 : power = 25
2: T1: power = 20
1: T3: power = 15

Periodic tasks with common period

3: T2  T2  T2  T2
2: T1  T1
1: T3  T3
Periodic tasks with common period

3:

1:

Periodic tasks with common period

1:
Scenario 2: Periodic tasks with individual period

- A set of periodic $N$ heterogeneous tasks in a set $L$ where each task has its own period $p_i$. The arrival time $a_i$ is equal to the start time of its period and the deadline $d_i$ is equal to the end time of its period.

- The goal is to find a sequence of these tasks using task partitioning to minimize the peak temperature.

Algorithm: Periodic tasks with individual period

Use EDF scheduler to get the initial schedule of these tasks

while loop for $M$ times do
  Calculate the thermal profile of task sequence, find the “hot” task instance $\tau_h$ where peak temperature occurs.
  Partition the task instances whose execution period overlap with the arrival time or deadline of the “hot” task instance.
  In the hot interval, remove all the subparts of $\tau_h$ and calculate the available slack for each “cool” task instance.
  while there are parts of $\tau_h$ unassigned and some “cool” task instance has available slack do
    for each “cool” task instance $\tau_c$ in the hot interval do
      if $\text{slack}_i > 0$ then
        Append one unit of $\tau_h$ into $\tau_c$ and update the slack for all “cool” task instances
      end if
    end for
  end while
  If there is still some subparts of $\tau_h$ unassigned, scan the hot interval and assign them uniformly into the idle time.
end while
Periodic tasks with individual period

- **Partition the task instances whose execution period overlaps with the arrival time and/or deadline of the “hot” task instance**
  
  **Slack allocation:**
  
  \[ EST_i = \max(a_{r_h}, a_{r_i}, EST_{pred_i} + c_{pred_i}) \]
  
  \[ LST_i = \min(d_{r_h}, d_{r_i}, LST_{succ_i}) - c_i \]
  
  \[ slack_i = LST_i - EST_i \]
Periodic tasks with individual period
Load Balancing: Particles and Mesh

Computation Partitioning: Types of Parallelism

- **Independent Parallelism**
  - Bundled simulations using DAKOTA (e.g. UQ, parametric variations)

- **Task Parallelism**
  - Independent models that are simulated concurrently
  - (e.g. Fluid, particle coupling at microscale)

- **Data Parallelism**
  - Parallelization of Eulerian grid
  - Parallelization of Lagrangian particles
Communication Mapping: Types of Interactions

- Across cells and elements
- Across immersed interfaces

Levels of AMR

Load Balancing: Types of Adaptivity

- Extreme event UQ-driven
- Computational steering

- Preferential particle clustering
- Lagrangian remap

Adaptive mesh refinement

Computational power focusing
CUDA Memory model

- A Thread block has R/W access to Shared memory
- A block grid has R/W access to Global memory
- A block grid has R/only to Constant memory
- Global memory (of order 4G) resides in DRAM and has very high access latency than the shared memory

Global memory access

- When accessing global memory, peak performance is achieved when all threads in a half warp access continuous memory locations.
Shared memory access

- Shared memory is divided into banks
- Multiple simultaneous accesses to a bank result in a bank conflict
  - Conflicting accesses are serialized

---

4 Phases of PIC algorithm

1. Charge Deposition Phase
   - Compute the forces (Poisson equations) needed for particle motion from the accumulated particle charges

2. Field Solve Phase

3. Force Gathering Phase

4. Particle push Phase
PIC algorithm on a triangular mesh

- Irregular structure makes partitioning complex.
- Each particle requires a search to find the enclosing triangle
  - This step forms an additional **Search Phase** in the PIC algorithm flow
- Search phase forms one of the time consuming steps in the PIC flow

---

GPU Parallelization using Mesh coloring

- Triangles in mesh are considered as nodes of a graph
- Triangles with at least one common vertex are of different colors
- Every GPU kernel works only on one color
- **Pros**
  - No conflicting access
- **Cons**
  - Needs multiple kernel invocations.
  - For efficient indexing, we need to maintain color sorted order of triangles and particles which involves additional computation
GPU Parallelization using Mesh Partitioning

- Particles, triangles and vertices of triangles form the partitioning entities in the PIC problem
- Mesh is partitioned into regions using a virtual rectangular grid
- Each region is mapped to a GPU block

Triangular Mesh Partitioning

- Triangles that cross region boundaries are referred as shadow triangles. The vertices of shadow triangles are termed as shadow vertices
- Shadow vertices and triangles are replicated
- Particles, triangles and vertices are represented using linear arrays
Replication of shadow entities

- Replication ensures that each block can compute independently of the other blocks
- After the computation, an aggregation step merges the values from all shadow vertices

<table>
<thead>
<tr>
<th>Vertices in Region 1</th>
<th>Shadow Vertices in Region 1</th>
<th>Vertices in Region 2</th>
<th>Shadow Vertices in Region 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangles in Region 1</td>
<td>Shadow Triangles in Region 1</td>
<td>Triangles in Region 2</td>
<td>Shadow Triangles in Region 2</td>
</tr>
<tr>
<td>Particles in Region 1</td>
<td></td>
<td>Particles in Region 2</td>
<td></td>
</tr>
</tbody>
</table>

GPU kernels

- Bucket sort for triangles, vertices and particles
- For each simulation iteration
  - Triangle search
  - Field solve phase
  - Force aggregation of shadow vertices
  - Particle push phase
  - Re-sorting of moved particles
Triangle density based partitioning

- Ensures effective load balancing across regions
- Expensive pre-processing step
- Need to use a spatial indexing data structure like KD-tree to partition triangles
- During search phase particles traverse the KD-tree
- KD-tree is not very well suited for GPU

Partitioning using Level 1 grid

- The virtual rectangular grid partitions the mesh into regions
- Pre-processing step is very fast
- Load imbalance due to difference in triangle density
- The linear search for triangles can be a bottleneck
Level 2 Partitioning (Partition Region into Sub-Regions)

- Used only for the search phase
- No replication of shadow vertices across sub-regions as the vertices are read-only while searching.
- Requires sorting of triangles, and vertices in sub-region order which is again part of pre-processing
- Sub-region order sorting of particles has to be performed after each iteration

Incremental Sort – Uniform Partitioning

- After each simulation iteration, particles have to be re-sorted
- For efficient bucket sort all the sub-regions should be present in shared memory which stores the particle count in the sub-region
- As number of sub-regions increase, it won't fit in shared memory of GPU
- In reality most of the particles will move only to adjacent sub-regions.
- Keep only adjacent regions (Y) to region (X) in shared memory
Non-uniform Partitioning using Level 2 grid

- Level 2 grid is not uniform
- Dense in regions where triangle density is more.
- Non-uniformity creates asymmetry and requires more complex pre-processing and indexing methods.
- Incremental sort becomes very complex

Field solve phase

- Most of the flops are executed in this phase
- Each region is mapped to a GPU block.
- GPU block loads the vertices and shadow vertices in a region to shared memory
- Each thread operates on a set of particles in the region
- The force is updated on vertices/shadow vertices in shared memory.
  - Different particles can update the same vertex. Hence atomic update is used. Doesn’t consume much cycles as atomic updates in shared memory are very fast.
- Once the block completed execution the values are written back to global memory
Experimental Results

- Mesh from ORNL used for XGC1 benchmarks
  - 1.8 Million triangles
  - Randomly distributed 18 Million particles
  - Level 1 partitioning uses 32 X 32 rectangular grid (regions)
  - NVIDIA Tesla T10 GPU with 4GB global memory, 16k shared memory and 240 computing cores

Comparison of triangle search time (10 simulation iterations)

- Uniform and non-uniform partitioning gives similar performance when there are sufficient number of GPU blocks
- Simpler uniform partitioning would be a better choice

<table>
<thead>
<tr>
<th>GPU blocks</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>12561.06</td>
</tr>
<tr>
<td>2779</td>
<td>7235.16</td>
</tr>
<tr>
<td>22471</td>
<td>989.88</td>
</tr>
<tr>
<td>33464</td>
<td>428.51</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU blocks</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>3111.11</td>
</tr>
<tr>
<td>9216</td>
<td>1366.21</td>
</tr>
<tr>
<td>16384</td>
<td>877.23</td>
</tr>
<tr>
<td>25600</td>
<td>609</td>
</tr>
<tr>
<td>36864</td>
<td>500.92</td>
</tr>
<tr>
<td>50176</td>
<td>427</td>
</tr>
</tbody>
</table>
Particle sorting time (Non-uniform partitioning)

- ~20X speedup when using shared memory for sorting

<table>
<thead>
<tr>
<th>Order of sub-region (Max # of triangles in sub-region)</th>
<th>Sorting using shared memory (Time in ms)</th>
<th>Sorting without using shared memory (Time in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>185.68</td>
<td>1492.55</td>
</tr>
<tr>
<td>200</td>
<td>183.88</td>
<td>1881.56</td>
</tr>
<tr>
<td>1000</td>
<td>183.55</td>
<td>2441.08</td>
</tr>
<tr>
<td>5000</td>
<td>183.99</td>
<td>3773.92</td>
</tr>
</tbody>
</table>

Particle Incremental Sorting time (Uniform Partitioning)

- Relatively independent of the number of blocks used

<table>
<thead>
<tr>
<th>Number of GPU blocks</th>
<th>Particles per thread</th>
<th>Time in (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>35157</td>
<td>2</td>
<td>61.49</td>
</tr>
<tr>
<td>17579</td>
<td>4</td>
<td>60.49</td>
</tr>
<tr>
<td>4395</td>
<td>16</td>
<td>60.73</td>
</tr>
<tr>
<td>1099</td>
<td>64</td>
<td>61.13</td>
</tr>
<tr>
<td>550</td>
<td>128</td>
<td>65.79</td>
</tr>
<tr>
<td>314</td>
<td>224</td>
<td>63.89</td>
</tr>
<tr>
<td>276</td>
<td>255</td>
<td>65.07</td>
</tr>
</tbody>
</table>
Conclusion

- Methodologies to Parallelize PIC on triangular mesh using GPUs
- Shadow entities (replication) provides a simpler and efficient solution
- Algorithms discussed are scalable with the size of mesh, number of particles and can be easily ported to a multi-GPU framework
Introduction

**Background:**
- Behavioral emulation (BE) approach: manages Exascale complexity via
  - BEOs (abstraction of object behavior; not cycle accurate)
  - Multi-scale (abstraction at micro, meso, and macro levels)

**Goal:** Research & develop toolset to scale BE approach of system simulation up to Exascale while maintaining required performance (speed)
- Software PDES behavioral simulator
- Hardware-accelerated behavioral emulator

**Approach:** (for behavior emulator)
- Explore methods of mapping BEOs onto systems of reconfigurable processors
- Investigate use of large-scale reconfigurable supercomputing, RSC (e.g., Novo-G#, next-gen RSC) in emulation of Exa/extreme-scale systems

**Related research:**
- Multi-FPGA systems (Novo-G, Catapult, BEE3-based cluster, Bluehive)
- Multi-FPGA system interconnect (Novo-G#, BEE3-based cluster)
- FPGA-accelerated architectural emulation (RAMP)
- Recent interest in FPGA-based heterogeneous computing for big data and data centers: Microsoft, IBM, Intel, Oracle, Google, Baidu
Session 3 Outline

- Introduction: motivation, goal, & approach
- Mapping BEOs onto reconfigurable (RC) platform
  - Novo-G# reconfigurable supercomputer
  - Basic appBEO, procBEO, & commBEO designs
- Current single-FPGA prototype (NGEEv1)
  - Current single-device prototype & demo
  - Additional results & SMP performance comparison
- Transitioning to multiple FPGAs
  - Questions, identified issues, & possible directions
  - Direct FPGA-to-FPGA communication via 3D interconnect in Novo-G#
  - New potential NGEE target architecture
  - Proposed scalability measure
- NGEEv2 single-FPGA design
  - Effect of BE V2 methodology improvements on our FPGA acceleration efforts
  - Multiple-FPGA considerations

Conclusions

Novo-G Reconfigurable Supercomputer

- Developed and deployed at CHREC
  - Most powerful reconfigurable computer in (academic) world
  - 2012 Alexander Schwarzkopf Prize for Technology Innovation @ NSF Center
- Apps acceleration
  - In key science domains: bioinformatics, finance, image & video processing
- Hardware emulation
  - Behavioral emulation of future-gen systems, up to Exascale
- 2014 upgrade
  - 64 GiDEL ProceV (Stratix V D8)
  - 4x4x4 3D-torus or 6D-hypercube
  - 6 Rx-Tx links per FPGA
  - 4x 10 Gbps per link

Novo-G Annual Growth

<table>
<thead>
<tr>
<th>Year</th>
<th>Cards</th>
<th>FPGAs</th>
<th>SDRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>24</td>
<td>Stratix-III</td>
<td>4.25GB</td>
</tr>
<tr>
<td>2010</td>
<td>24</td>
<td>Stratix-III</td>
<td>4.25GB</td>
</tr>
<tr>
<td>2011</td>
<td>24</td>
<td>Stratix-IV</td>
<td>8.50GB</td>
</tr>
<tr>
<td>2012</td>
<td>24</td>
<td>Stratix-IV</td>
<td>8.50GB</td>
</tr>
<tr>
<td>2014</td>
<td>64</td>
<td>Stratix-V</td>
<td>8.50GB</td>
</tr>
</tbody>
</table>

with high-speed 4x4x4 torus or 6D-hypercube
Novo-G ProceV Upgrade w/ 3D Torus

**Upgraded Novo-G**
- 4x4x2 Torus (soon to be 4x4x4)
- 8 ProceV nodes
- 32 GiDEL ProceV (Stratix V D8)
- 4x4x2 3d-torus or 5d-hypercube
- 6 Rx-Tx links per FPGA
- 4x 10 Gbps per link
- Data-link layer: SerialLite III protocol
  - Full-duplex, CRC32 protection, in-band or out-of-band flow control
- Physical layer: Interlaken protocol
  - 64B/67B encoding, multi-lane sync.

**Novo-G# (Novo-jee-sharp)**
- 32 GiDEL ProceV (Stratix V D8)
- 4x4x2 3d-torus or 5d-hypercube
- 6 Rx-Tx links per FPGA
- 4x 10 Gbps per link
- Data-link layer: SerialLite III protocol
  - Full-duplex, CRC32 protection, in-band or out-of-band flow control
- Physical layer: Interlaken protocol
  - 64B/67B encoding, multi-lane sync.

**Novo-G# 3D Torus Protocol Stack**

- **Network layer**
  - Dimension order routing
  - Collective routing
  - Source data buffering

- **Data-link layer**
  - Data framing
  - Error detection (CRC)

- **Physical layer**
  - Clock recovery
  - Line coding
  - Multi-lane sync.

- **Services available** through IP
- **Services available** through RTL design

- **3-layer 3D torus protocol stack** (shown above) based on IP from Altera and GiDEL
- Basic point-to-point services provided by Interlaken and SerialLite-III, network oriented services provided by RTL code
- Direct FPGA interconnect crucial to the scalability of NGEE

Special contributions by Abhijeet Lawande via cost-share from CHREC
Mapping AppBEO onto Single FPGA

- High-level appBEO script (abstraction of target app) mapped to custom machine code (MIF file)
- Stream of instructions for procBEOs
- Instruction delivery options
  - Pulled from on-chip ROM
  - Pushed from CPU (Instr. stream from CPUs through external memories)

```
appBEO (high-level description)

/* Define group as nodes 0-3 */
VAR commGrp=0,3
/* Broadcast variable A */
dataSize=54*34/2 to group:
Bcast(int32,2048,0,commGrp)
/*Barrier sync*/
Barrier(commGrp)
/* Scatter 1/4 of matrix R */
dataSize=(54*4/4) to each node:
Scatter(int32,112,0,commGrp)
/* Perform dot products of vector size 64 */
DotProduct(int32,64)
/* Gather solutions from matrices */
(dataSize=(54*4/4)*4)
Gather(int32,512,commGrp)
Done
```

Mapping ProcBEO onto Single FPGA

- Mimics “real” processor under study
  - Instruction decoding, timekeeping
  - No real computation: interpolation of compute operations
  - Generates tokens to emulate comm packets
- Lightweight processing elements
- Initial prototype
  - One-to-one mapping of procBEOs to interpolation & comm resources
Mapping CommBEO onto Single FPGA

- **System-specific**: fabric is explicit emulation of target architecture
- Consists of token buffers, arbiter, router, network timer
- Packets transferred contain characteristics of (not real) data

Session 3 Outline

- **Introduction**: motivation, goal, & approach
- Mapping BEOs onto reconfigurable (RC) platform
  - Novo-G# reconfigurable supercomputer
  - Basic appBEO, procBEO, & commBEO designs
- **Current single-FPGA prototype (NGEEv1)**
  - Current single-device prototype & demo
  - Additional results & SMP performance comparison
- Transitioning to multiple FPGAs
  - Questions, identified issues, & possible directions
  - Direct FPGA-to-FPGA communication via 3D interconnect in Novo-G#
  - New potential NGEE target architecture
  - Proposed scalability measure
- **NGEEv2 single-FPGA design**
  - Effect of BE V2 methodology improvements on our FPGA acceleration efforts
  - Multiple-FPGA considerations
- **Conclusions**
Functioning prototype running on single FPGA of Novo-G

- **No optimization** (i.e., max-resource implementation)
- Current core density of **90** for Stratix IV, **256** for Stratix V
  - Each core contains one each of appBEO, procBEO, and commBEO
  - Stratix IV currently limited by FPGA block RAM, not logic
    - 9x10 mesh on Stratix IV: logic 19%, block memory 100%
  - Higher core density on Stratix V
    - 16x16 mesh on Stratix V: logic 94%, block memory 100%
- appBEO scripts stored in on-chip block RAMs as memory initialization files (MIFs)
- Proc interpolation resources replaced with MIF *pre-processing*
- Explicit emulation of target communication fabric *without congestion modeling*
- Separate management plane fabric collecting management tokens for postmortem analysis (e.g., simulation visualization)

**DEMO: FPGA-specific appBEOs**

- Generate memory initialization files (mif) to configure FPGA simulator
  - R script to convert appBEO instructions into custom NGEE-specific machine code
  - Generates core-level instruction streams for configuring simulators FPGA bit file
DEMO: Simulator Setup & Execution

- mif files assembled into FPGA bit file & loaded into FPGA
- Custom c driver initiates simulator and collects results from FPGA management plane
- Management plane logs core-level events & streams to host

Additional Results & SMP Performance Comparison

Experimental Setup

Experiments with TileGX36, next-gen TileGX72, & anticipated Intel Xeon Phi Knights Landing architecture

- Single Stratix IV E530 vs SMP on single quad-core Xeon E5520 CPU @ 2.27GHz
- Proc/comm configurations: Tile 6x6, Tile 9x8, Knights Landing 9x8
- App configuration: work equally distributed to all available cores for each proc/comm configuration
- Apps: 2D MM & Sobel filtering
- App kernels executed 250 times to amortize simulator overheads
- Compare management results to SMP for equivalency/correctness
- Compare execution times to SMP for performance improvement
Performance Comparison: 3 Data Points

<table>
<thead>
<tr>
<th>Tile Size</th>
<th>Simulated Time</th>
<th>Prediction Error</th>
<th>FPGA Simulation Time</th>
<th>SMP Simulation Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>6x6</td>
<td>2.82 × 10⁹ ns</td>
<td>-0.35%</td>
<td>35.7 μs</td>
<td>4.82 ms</td>
<td>~135x</td>
</tr>
<tr>
<td>6x6</td>
<td>9.27 × 10⁷ ns</td>
<td>-2.61%</td>
<td>54.2 μs</td>
<td>8.08 ms</td>
<td>~149x</td>
</tr>
</tbody>
</table>

Next-gen

<table>
<thead>
<tr>
<th>Tile Size</th>
<th>Simulated Time</th>
<th>Prediction Error</th>
<th>FPGA Simulation Time</th>
<th>SMP Simulation Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>9x8</td>
<td>1.66 × 10⁹ ns</td>
<td>To be determined</td>
<td>81.1 μs</td>
<td>10.24 ms</td>
<td>~126x</td>
</tr>
<tr>
<td>9x8</td>
<td>2.58 × 10⁷ ns</td>
<td>To be determined</td>
<td>102.0 μs</td>
<td>17.94 ms</td>
<td>~176x</td>
</tr>
</tbody>
</table>

Anticipated

<table>
<thead>
<tr>
<th>Tile Size</th>
<th>Simulated Time</th>
<th>Prediction Error</th>
<th>FPGA Simulation Time</th>
<th>SMP Simulation Time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>9x8</td>
<td>5.87 × 10⁸ ns</td>
<td>To be determined</td>
<td>81.1 μs</td>
<td>10.24 ms</td>
<td>~126x</td>
</tr>
<tr>
<td>9x8</td>
<td>1.37 × 10⁷ ns</td>
<td>To be determined</td>
<td>102.0 μs</td>
<td>17.94 ms</td>
<td>~176x</td>
</tr>
</tbody>
</table>

Session 3 Outline

- Introduction: motivation, goal, & approach
- Mapping BEOs onto reconfigurable (RC) platform
  - Novo-G# reconfigurable supercomputer
  - Basic appBEO, procBEO, & commBEO designs
- Current single-FPGA prototype (NGEEv1)
  - Current single-device prototype & demo
  - Additional results & SMP performance comparison
- Transitioning to multiple FPGAs
  - Questions, identified issues, & possible directions
  - Direct FPGA-to-FPGA communication via 3D interconnect in Novo-G#
  - New potential NGEE target architecture
  - Proposed scalability measure
- NGEEv2 single-FPGA design
  - Effect of BE V2 methodology improvements on our FPGA acceleration efforts
  - Multiple-FPGA considerations
- Conclusions
Transitioning to Multiple FPGAs

With Novo-G# as target system architecture, requires modification of current design to allow communication between commBEOs instantiated on different FPGAs over direct FPGA interconnect

Q1. Effect on current single-FPGA design?
Q2. Implications on speed/scalability?
Q3. Would the use of other system architectures be more advantageous given new BE requirements?

Q1. Effect on current single-FPGA design?
- Added communication infrastructure & overhead
- Multi-layer communication protocol & virtual network fabric
- Modified ISA, packet structure, management tokens, inter-device bandwidth allocation
- General design considerations (e.g., arbitrary no. of resources vs. hardcoded limits of single FPGA)
- Likely reduced BEO density

Q2. Implications on speed/scalability?

Q3. Would the use of other system architectures be more advantageous given new BE requirements?
Direct FPGA-to-FPGA Communication via 3D Interconnect

Transitioning to Multiple FPGAs

Q1. Effect on current single-FPGA design?
- Added communication infrastructure & overhead
- Multi-layer communication protocol & virtual network fabric
- Modified ISA, packet structure, management tokens, inter-device bandwidth allocation
- General design description (i.e., arbitrary No. of resources vs. hardcoded limits of single FPGA)
- Likely reduced BEO density

Q2. Implications on speed/scalability?

Q3. Would the use of other system architectures be more advantageous given new BE requirements?
Transitioning to Multiple FPGAs

Q1. Effect on current single-FPGA design?

Q2. Implications on speed/scalability?
   - BEO wait times
   - Inter-FPGA event queuing, flow controls, queuing size, event reordering
   - Sharing of BEO resources
   - Proposed scalability measurement & its use to inform multi-FPGA design decisions

Q3. Would the use of other system architectures be more advantageous given new BE requirements?

Scalability Studies & Projections

Definitions:
- **Emulation system**: Behavioral emulation platform such as *Novo-G#
- **Emulated system**: *appBEOS* (e.g., modeling CMT app) stimulating *archBEOS* (e.g., modeling Blue Gene/L)

Open questions to be answered in the future:
- For a given emulation system architecture (e.g., #FPGAs, BEO core density, core design, interconnect arch, etc.), **What are the limits of an emulated system?**
  - Including size (e.g., #BEOs) and emulation performance
- For given requirements of an emulated system (e.g., macro-scale emulation with Blue Gene/L), **What emulation system resources are necessary?**
  - Including #FPGAs, core density, interconnect arch, etc.
Potential Scalability Measure

Objective:
- Compare scalability (HW) vs. scalability (SW)
  HW: hardware approach; SW: software SMP approach

Potential Scalability Measure for HW
- Ideally entire simulation system is on single large FPGA; thus, communication between BEOs is at on-chip rate
- **Baseline:**
  - Validated BE model for single-FPGA performance (PfS) of NGEE (i.e., BE model of FPGA running other BE models)
- **Scalability issues** arise when BEOs communicate across FPGAs
  - Off-chip communication much more costly
- **Approach**
  - Validated BE model for multiple-FPGA performance (PfM) of NGEE (possible after multi-FPGA experiments)

**Potential Scalability Measure** SM(HW) = PfS/PfM

Transitioning to Multiple FPGAs

Q1. Effect on current single-FPGA design?

Q2. Implications on speed/scalability?
- BEO wait times
- Inter-FPGA event queuing, flow controls, queuing size, event reordering
- Sharing of BEO resources
- Proposed scalability measurement & its use to inform multi-FPGA design decisions

Q3. Would the use of other system architectures be more advantageous given new BE requirements?
Transitioning to Multiple FPGAs

Q1. Effect on current single-FPGA design?
Q2. Implications on speed/scalability?
Q3. Would the use of other system architectures be more advantageous given new BE requirements?

New Potential NGEE Target Architecture

Node architecture:
- 2 CPUs
- 2 CAPI attached accelerators
- 4 accelerator 1D torus

System configuration:
- 4 POWER8 servers
- 16 Nallatech boards
- 16 board 2D torus
- Up to 32 FPGAs w/ dual chip boards
- CAPI enables a hardware kernel bypass

Resource pool functionality & OpenCL support

OpenCL Resource Pool Interconnect manager

OpenCL Kernel 0

OpenCL Kernel 1

OpenCL Kernel N

OpenCL POWER Service Layer

CCMT

FPGA

Compute Node
Session 3 Outline

- Introduction: motivation, goal, & approach
- Mapping BEOs onto reconfigurable (RC) platform
  - Novo-G# reconfigurable supercomputer
  - Basic appBEO, procBEO, & commBEO designs
- Current single-FPGA prototype (NGEEv1)
  - Current single-device prototype & demo
  - Additional results & SMP performance comparison
- Transitioning to multiple FPGAs
  - Questions, identified issues, & possible directions
  - Direct FPGA-to-FPGA communication via 3D interconnect in Novo-G#
  - New potential NGEE target architecture
  - Proposed scalability measure
- NGEEv2 single-FPGA design
  - Effect of BE V2 methodology improvements on our FPGA acceleration efforts
  - Multiple-FPGA considerations

Conclusions

NGEEv2 single-FPGA design

*Updated design based on current and possible future developments in fundamental BE methodology, emulation system architecture, ...*

- Effect on FPGA acceleration efforts going forward?
  - Both single & multi FPGA considerations

- Identified issues
  - BEv2 modifications: e.g., congestion modeling, global task graph manipulation, micro-scale symmetry exploitation, multi-pass simulation ...

- Possible approaches and directions
  - Alternative acceleration approaches?
  - Alternate target system architectures?
Conclusions

Progress:
- **Working single-FPGA prototype** (micro-scale) with max-resource implementation & management plane (*no optimization*)
- Beginning stages of performance optimization & scalability evaluation
- New design (NGEEv2) ideation

Plans for March:
- **Prototype** NGEE platform operating on *multiple* FPGAs
- Showcase results from optimization studies
  - Increased BEO density per FPGA
- **Performance comparison** with software-based SMP simulator for multiple appBEO scripts
- Upgraded Novo-G# (4x4x4 torus) supporting BE
- New performance/scalability predictions for fully upgraded Novo-G#