Languages, Compilation, and Models of Computation for Exascale Systems

PASCAL: PArallel Systems and Computer Architecture Lab.

Jean-Luc Gaudiot
Motivation

- Issues at hand
- Some solutions:
  - Heterogeneous architectures
  - Multi-level programming

Our framework: Exxen

- Definition
- Exxen design issues:
  - Memory model: value prediction, speculation
  - Scheduling and synchronization
  - Mapping and fault-tolerance
  - Other issues and past work

Putting it all together
Exascale computing is a highly challenging goal

- Not an extension of Petascale
  - Physical dimension and power consumption will require a completely different design philosophy
  - New hardware models (GPUs, etc.) may force significant design changes
  - New languages, compilers, and algorithms may be required to reap the benefits of this new hardware
  - As a consequence, the overall development will be more layered and complex, which means we will have to develop tools to manage development tasks

- Important issues:
  - Programmability
  - Compiler/Language Tools
  - Performance/Power consumption estimation/optimization
  - Reliability and Fault Tolerance
  - Application definition
Some challenges in future exascale designs

- Q1. How can we achieve power efficiency while achieving our performance/reliability goals?

- Q2. How can we achieve higher computational density and make the system manageable?

- Q3. How can we specify synchronization to express the enormous parallelism required? Can we deliver even more parallelism than available intrinsically?
Hardware issues

- Extrapolating on current technology, exascale is “physically” challenging (e.g., LBNL report, 2010):
  - 10K ~ 1000K nodes are required to build an exascale computer systems
    - E.g., 24,000 square meters housing 400 containers (the Microsoft million server datacenter)
  - The estimated power consumption is unacceptable
    - By 2018, 125 MW is expected for such large systems (Target: 20MW)

- Current abstract memory models may not be sufficiently expressive for the expected complexity (e.g., varying granularity of data movement, many different kinds of communication links, etc.)
Heterogeneous architectures

Heterogeneous architectures are good candidates to:

- **Build “manageable” systems**
  - Higher computational density $\rightarrow$ smaller number of nodes
  - Tesla C1060 GPU: 933 Gflops/unit vs. Intel Xeon W5590 CPU: 106 Gflops/unit

- **Enhance power efficiency**
  - GPGPU, and FPGA have better FLOPS/Watt
  - Tesla C1060 GPU: 0.20 W/Gflop vs. Intel Xeon W5590 CPU: 1.22 W/Gflop
FPGA vs. GPP vs. GPU

From “High Performance Quasi-Monte Carlo Financial Simulation: FPGA vs. GPP vs. GPU” by X. Tian et al., ACM Transactions on Reconfigurable Technology and Systems, 2010

<table>
<thead>
<tr>
<th>CPU</th>
<th>FPGA</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed-26bits</td>
<td>Fixed-29bits</td>
</tr>
<tr>
<td>1x</td>
<td>544x</td>
<td>448x</td>
</tr>
</tbody>
</table>

- FPGA: Vertex 4
- GPGPU: 8800 GTX
- CPU: Pentium 4 generation Xeon
Caution!

- Off chip memory operations are power hungry!
  - With standard technology roadmap, 48 MW for memory access, 12 MW for interconnect: only 10.6 MW for computation (LBNL Report, 2010)
  - Solve with advanced memory technology (the report expects we can reduce memory access overhead to 6.4 MW), but also we need to consider new computational models to reduce the overall frequency of data movement

- HPC applications generally need access to large arrays, lists, or trees of data, often larger than the memory available in a single node.
  - Careful mapping to maximize locality (recalculation)
  - Nomadic threads
Software issues / New Programming Paradigms

- How can we better specify parallelism?
  - HW/SW boundary
  - OS/Application boundary

- How can we improve programmability?
  - High multi-level language paradigms: e.g., YML
  - Functional Language/Dataflow
  - Flux Parallelism
  - Graphical Programming
  - Data Parallelism: CUDA, Open-CL
  - Use of Pragmas/end user input

- Re-calculation sometimes wiser when we consider power consumption of off-chip data movement
Outline

- Motivation
  - Issues at hand
  - Some solutions:
    - Heterogeneous architectures
    - Multi-level programming

- Our framework: Exxen
  - Definition
  - Exxen design issues:
    - Memory model: value prediction, speculation
    - Scheduling and synchronization
    - Mapping and fault-tolerance
    - Other issues and past work

- Putting it all together
Exxen: **EXtreme(-Scale) eXecution Environment**

- Our design framework for heterogeneous exascale computer system is one answer to the need for aggressive innovation.

- We will promote an integrated hardware/software framework which includes:
  - Fine-grain event-driven environment with macro data-driven origins
  - The integration of many-core CPU (e.g., Intel SCC), many-core GPU (e.g., NVIDIA), ASIPs, FPGAs, etc.
  - Compiler design (static assignment of workloads)
  - A runtime system (dynamic workload migrate)
  - Based on workload characterization
  - Founded on a flexible programming paradigm
Exxen as a framework for exascale computing

Compiler/Language

- Dataflow / functional languages
  - To map tasks more easily across heterogeneous computing devices
  - To reveal points of synchronization or communication for performance/power optimization

Runtime model for dynamic issues

- For performance issues of memory system,
  - Value prediction/Memory system
- For power efficiency of memory system and performance tweak during runtime,
  - Nomadic Thread

Fault-Tolerance

- Migrating work to available resources,
  - Nomadic thread
- FPGA: more vulnerable to failure,
  - Fault-Tolerant logic design
Outline

- Motivation of the problem
  - Issues at hand
  - Some solutions:
    - Heterogeneous architectures
    - Multi-level programming

- Our framework: Exxen
  - Definition
  - Exxen design issues
    - Memory model: value prediction, speculation
    - Scheduling and synchronization
    - Compilation, mapping and fault-tolerance
    - Other issues and past work

- Putting it all together
Rethinking Synchronization/Memory model

- Generally, we hide latency by implementing
  - Re-ordering
  - Asynchronous transactions
  - Relaxed memory model

- With massive parallelism on a multi-layered exascale computer, the following is quite difficult to achieve:
  - Minimizing synchronization overhead
  - Keeping correctness

- Some possible solutions include:
  - Potential Impact of Value Prediction on Communication in Many-Core Architectures (*IEEE Transactions on Computers, 2009*)
  - Performance of value prediction framework on GPGPU
Exploratory Study

- **LL6**: high scalable, demonstrates the efficiency of SSB
- **LL4 - LL20**: Speedup/scalability strictly restricted by data dependencies
Exploratory Study

- Communication latency: the number of cycles the consumer thread spends at the synchronization point waiting for the producer to reach completion
**Exploratory Study**

- As shown in the cases of LL4 and LL20, parallelism is limited by data dependencies

- Implementation with a “virtual” value predictor:
  - Make prediction at synchronization points where the consumers have to wait for the producers
  - Prediction accuracy is a control parameter
  - Assume no rollback/re-execution overhead
  - Assume a 2-cycle overhead in making each prediction
**Exploratory Study**

- LL6 is highly parallel; using value prediction does not bring much benefit
- LL4 is less parallel; value prediction is beneficial only after saturation
- LL20 is sequential; value prediction is always beneficial
A Theoretical Framework of Value Prediction in Parallel Systems

- Characterization of inherent data redundancy on PARSEC and SPLASH-2 benchmarks
  - Use the binary instrumentation tool on PARSEC and SPLASH-2 benchmarks to collect runtime statistics
  - Apply analysis tools (based on information theory) to the runtime statistic data to extract measures such as information entropy, redundancy, and minimum misprediction rate

- Verification of the Potential of Fine-Grained Value Prediction
  - LVP: Last Value Predictor
  - FCP: Finite Context Predictor
  - SVP: Stride Value Predictor
## Regular and Stride Data Predictability

- (R) Maximum regular value prediction accuracy (exploited by LVP & FCP)
- (S) Maximum stride value prediction accuracy (exploited by SVP)

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>S</th>
<th></th>
<th>R</th>
<th>S</th>
<th></th>
<th>S</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barnes</td>
<td>0.78</td>
<td>0.90</td>
<td>Radix</td>
<td>0.63</td>
<td>0.65</td>
<td>Canneal</td>
<td>0.69</td>
<td>0.85</td>
</tr>
<tr>
<td>Radiosity</td>
<td>0.75</td>
<td>0.69</td>
<td>Raytrace</td>
<td>0.77</td>
<td>0.77</td>
<td>Dedup</td>
<td>0.52</td>
<td>0.51</td>
</tr>
<tr>
<td>FFT</td>
<td>0.63</td>
<td>0.62</td>
<td>Volrend</td>
<td>0.63</td>
<td>0.58</td>
<td>Facesim</td>
<td>0.60</td>
<td>0.83</td>
</tr>
<tr>
<td>FMM</td>
<td>0.70</td>
<td>0.69</td>
<td>Bodytrack</td>
<td>0.71</td>
<td>0.63</td>
<td>Fluidanimate</td>
<td>0.62</td>
<td>0.86</td>
</tr>
<tr>
<td>LU</td>
<td>0.75</td>
<td>0.62</td>
<td>Swapitions</td>
<td>0.66</td>
<td>0.57</td>
<td>Freqmine</td>
<td>0.66</td>
<td>0.78</td>
</tr>
<tr>
<td>Ocean</td>
<td>0.41</td>
<td>0.72</td>
<td>Blackscholes</td>
<td>0.71</td>
<td>0.77</td>
<td>Streamcluster</td>
<td>0.67</td>
<td>0.64</td>
</tr>
</tbody>
</table>
Thread-level parallelism

- Routines that contain tens of instructions and basic blocks that contain five to ten instructions are good candidates for fine-grained threads
- Characterize the predictability of routine return values and basic-block live-in values with the theoretical model that we developed

There exists a strong data redundancy in routine return values and basic block live-in values, implying the feasibility of utilizing value prediction to enhance TLP

<table>
<thead>
<tr>
<th>Routine</th>
<th>BBL</th>
<th>Routine</th>
<th>BBL</th>
<th>Routine</th>
<th>BBL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barnes</td>
<td>0.95</td>
<td>Radix</td>
<td>0.97</td>
<td>Canneal</td>
<td>0.63</td>
</tr>
<tr>
<td>Radiosity</td>
<td>0.91</td>
<td>Raytrace</td>
<td>0.98</td>
<td>Dedup</td>
<td>0.73</td>
</tr>
<tr>
<td>FFT</td>
<td>0.88</td>
<td>Volrend</td>
<td>0.84</td>
<td>Facesim</td>
<td>0.68</td>
</tr>
<tr>
<td>FMM</td>
<td>0.88</td>
<td>Bodytrack</td>
<td>0.82</td>
<td>Fluidanimate</td>
<td>0.83</td>
</tr>
<tr>
<td>LU</td>
<td>0.96</td>
<td>Swaptions</td>
<td>0.73</td>
<td>Freqmine</td>
<td>0.70</td>
</tr>
<tr>
<td>Ocean</td>
<td>0.96</td>
<td>Blackscholes</td>
<td>0.89</td>
<td>Streamcluster</td>
<td>0.29</td>
</tr>
</tbody>
</table>
Value Prediction and Speculation with GPUs

- Strong *data redundancy* in application programs.
- GPU is good for SIMD/data parallel applications
  - Many datapaths: simultaneously handle many speculative streams
  - No need for a decision circuit, map one predictor to each speculative stream, if one is correct, the execution can move on
  - Aggressive techniques: speculate a few steps ahead
  - Low synchronization overhead: low rollback cost
- Example:

```c
for( int i=0; i<N; i++)
{
    k[i] = Op(k[i-1]);
}
```
Value Prediction on GPU

- **Stage 1:** generates live-in values for speculative (slave) and non-speculative (master) threads
- **Stage 2:** both speculative and non-speculative threads start simultaneously
- **Stage 3:** update value predictors and commit results to memory

![Diagram showing stages of value prediction on GPU]
Speculation on GPU

Cont_SPEC: this is the concentrer routine taken from streamcluster of PARSEC. This loop is hard to parallelize because the data production in the loop body goes through a level of indirection such that data consumption cannot be performed until the source data address has been resolved.
Value prediction and speculation (lessons learned)

- Value prediction can efficiently amortize the communication overhead without the complexity of re-ordering.

- With data parallelism, re-calculation or re-execution of wrong speculation becomes simple because:
  - No special speculative storage
  - Simple re-injection of the value to the input of a kernel
Value prediction and speculation (leverage)

- We will have a lot of hardware resources and thus for high-performance we can try value prediction and speculative execution on the idle cores.
- Expand previous results to largely increased system size
Outline

- Motivation
  - Issues at hand
  - Some solutions:
    - Heterogeneous architectures
    - Multi-level programming

- Our framework: Exxen
  - Definition
  - Exxen design issues:
    - Memory model: value prediction, speculation
    - Scheduling and synchronization
    - Compilation, mapping and fault-tolerance
    - Other issues and past work

- Putting it all together
Mapping to computing nodes: compiler issues

- Flexibility: DD/codelets can be mapped from
  - Functional language
  - Hardware description languages: Bluespec, LAVA

- Makes speculation easier to handle
  - Power efficient
    - No need for running all the threads
  - No need for saving state for re-execution
    - No need for complex speculative storage
  - Handle synchronization issue better
    - In case of re-execution, just re-injecting non-speculative token
  - Low memory overhead for history
    - No need for global searching
Speculation/Prediction design with EXXEN

- With DFG, we can spot the points for “easy” speculation/prediction
  - By detecting delays from incoming tokens
- When delay on the delivery of specific token
  - FPGA: add predictor
  - Codelet or function: transactional memory
  - GPGPU: our framework
Speculation/Prediction

- Source of difficulties for optimization in the development phase
  - Task mapping onto various heterogeneous hardware
  - Task mapping to minimize communication overhead
  - Task mapping to minimize synchronization overhead
  - Also, time complexity of synthesizing and optimization onto FPGA target

- How can we significantly increase performance and maintain high programmability?
  - Speculation/Prediction technology can be a solution
  - However, speculation often trades performance off for power consumption
  - With careful use of speculation, power consumption with speculation/value prediction can be traded off for enhanced performance

Value Prediction on GPGPU

- Our framework to enhance programmability on GPGPU by handling loop carried dependencies speculatively

- Limitations:
  - Assume strong *data redundancy* in application programs
    - Power hungry
    - Performance penalty under “no dependencies” and “too many dependencies with weak redundancy”
  - We can gain more if we decide when to use this framework smartly
Speculation/Prediction design with EXXEN

- GPU Board
  - Local Memory
- Main Memory
- FPGA Board
  - Local Memory

Speculatively send data with predicted value

Dependency causing delay in transferring data

Other Nodes

Many core CPU
Outline

- Motivation
  - Issues at hand
  - Some solutions:
    - Heterogeneous architectures
    - Multi-level programming

- Our framework: Exxen
  - Definition
  - Exxen design issues:
    - Memory model: value prediction, speculation
    - Scheduling and synchronization
    - Compilation, mapping and fault-tolerance
    - Other issues and past work

- Putting it all together
Different programming style/Different tools
Mapping/scheduling to optimizing performance becomes more complex
Optimization (2)

- Using functional language or dataflow for restricted regions
  - Enhancing programmability from the more flexible mapping
    - Codelet (unit of code): Model from Dr. Gao, UDEL
    - Module (unit of circuit which will be mapped to FPGA)
      - Efficiently map to hardware description language: Bluespec, LAVA

- Can we make a library to estimate power/performance?
“Pre-compilation”

- **Performance/Power estimation**
- **Library**: contains information on power/performance for similar function/codelet/module
- **With this estimation, real compile for specific target followed**

<table>
<thead>
<tr>
<th></th>
<th>Power</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codelet</td>
<td>1.2 W</td>
<td>2 Gflops</td>
</tr>
<tr>
<td>Module</td>
<td>0.2 W</td>
<td>1 Gflops</td>
</tr>
</tbody>
</table>

Estimate on various target architectures
Another benefit of Pre-compilation

- Communication/Locality
  - Lots of data movement to the local memory of GPGPU or FPGA
  - Can decide compile target by considering data movement and performance/power estimation
Other system design benefits and challenges with EXXEN: Fault tolerance and resiliency

FPGA

First order function or task

Upon failure, remap to another computational element, and re-execute

Threads on many-core architecture
Outline

- Motivation
  - Issues at hand
  - Some solutions:
    - Heterogeneous architectures
    - Multi-level programming

- Our framework: Exxen
  - Definition
  - Exxen design issues:
    - Memory model: value prediction, speculation
    - Scheduling and synchronization
    - Compilation, mapping and fault-tolerance
    - Other issues and past work

- Putting it all together
Others

- Speculation by compiler
- PIM design
- SMT vs. CMP tradeoffs
- Power Efficiency in Cache Design
- I-Structures / State Buffer Synchronization
- Architecture approaches for fault tolerance
- Nomadic thread synchronization
Locality in EXXEN

I-Structure Software Cache

- Non-Blocking Multi-threaded execution + I-Structure memory system
  - Split-Phase transaction tolerates communication latency
  - Presence bit provides asynchronous memory operations.
  - Single assignment rule eliminates side effects.
  - Non-strict data access provides fine grain parallelism

- But, data locality is not exploited and introduces potential high network traffic problem.
  - Exploiting locality is very important to reduce amount of network traffic
I-Structure cache design

Split-Phase transaction caching scheme.
Architectures for soft error detection and recovery

- Soft errors have become an increasingly important problem
- From an architecture perspective, examine radiation-induced transient faults issues:
  - \( \text{SER} = \text{function (prog, ISA, µarchitecture, VLSI, device, fab, particle flux)} \)
- Use *architecture techniques* to harden microprocessors in the presence of radiation-induced transient faults:
  - Control flow checking
  - Redundant execution on SMT
  - Thread selection policies on SMT
Control Flow Checking Algorithm

V1
D(1) = 1
S(1) = 1

V2
D(2) = 2
S(2) = 3

V3
D(3) = 3
S(3) = 1

V4
D(4) = 4
S(4) = 6

V1 -> V4:
G = G XOR S(4) = D(1) XOR D(2) XOR D(4)
= 1 XOR 6
= 7
!= D(4) (CFE!)

V2 -> V4:
G = G XOR S(4) = D(2) XOR D(2) XOR D(4)
= D(4)

V1
D(1) = 1
S(1) = 1
V2
D(2) = 2
S(2) = 3
V3
D(3) = 3
S(3) = 1
V4
D(4) = 4
S(4) = 6
V1 -> V4:
G = G XOR S(4) = D(1) XOR D(2) XOR D(4)
= 1 XOR 6
= 7
!= D(4) (CFE!)
V2 -> V4:
G = G XOR S(4) = D(2) XOR D(2) XOR D(4)
= D(4)
Nomadic Threads: Thread & Process Migration

- Move Thread of Control To PE Where Data Resides
  - Exploits Spatial Locality
  - Makes Local Access Range Very Large
- Particularly Good For Linked Data Structures (Olden)
- Load Balancing (Cilk)
- Supports Many Granularities
  - Fine Grain – Olden, Gao & Hum, SMP
  - Medium Grain – Nomadic Threads
  - Large Grain – Mobile Agents, MIT
- Drawbacks
  - Migration Can Be Expensive
  - Multiple Migrations May Be Needed
  - Minimal Exploitation of Temporal Locality

Prior Research
Nomadic Threads

- **Goal:** Reduce Impact and Number of RMA Messages
  - **Idea:** Threads Migrate To Fetch Data (1 Message vs. 2)
  - **Approach:** Compare Migration to Caching
    - Remote Memory Access Operation Costs
    - Suitable Access Patterns and Program Structures
    - Compatible Thread Generation Approaches

- **Migration Testbed**
  - **Multithreading Runtime System (C)**
    - Equal Support for Migration and Caching
    - Architecture Independent
    - Vary PE Count, Including Virtual PEs
    - Detailed Statistics Gathering
  - **NT-SISAL Compiler (Java)**
    - Automatic Parallel Code Generation
    - Access Pattern Analysis & Heuristics for RMA Choice
    - Graphical Thread Building Monitor
Performance of Nomadic Threads

- Evaluated Array Locality Exploitation Effectiveness
  - NT Does Exploit Spatial Locality
  - Very Effective For Some Affine Access Patterns
  - NT Does Not Exploit well Temporal Locality
  - Caching is Better For Stencil Access Patterns
  - Can Greatly Aid Sequential Algorithms on Distributed Memory Machines

- Developed Parallelizing Compiler That Generates Migratory and Caching Remote Memory Access Code
  - Automatic Access Mechanism Choice Heuristics

- Built Architecture-Independent Runtime System That Supports Migration and Caching Array Access
Outline

- Motivation of the problem
  - Issues at hand
  - Some solutions:
    - Heterogeneous architectures
    - Multi-level programming

- Our framework: Exxen
  - Definition
  - Exxen design issues
    - Memory model: value prediction, speculation
    - Scheduling and synchronization
    - Compilation, mapping and fault-tolerance
    - Other issues and past work

- Putting it all together
Design of EXXEN

**Language/compiler /runtime support**
- Speculative execution
- SISAL
- Codelet execution model

**Novel Memory systems**
- Value Prediction
- Speculation
- I-Structure/SSB
- Nomadic Threads
- Lazy evaluation on data flow system

**Fault tolerance /Resiliency**
- Fault tolerance in dataflow systems
- Fault tolerance in network of distributed system
- Radiation-induced faults in modern processors

**Power/Perf. Optimization**
- SMP/CMT Trade
- Power efficient cache
- Nomadic Threads
Future Research

- Road to exascale computing is challenging
  - Multifaceted design issues
  - Physical limitations
- Target application: NWChem
  - Linear algebra
  - Quantum mechanics problem solvers
  - CPMD
- We will solve and challenge the exascale computing issues with our framework “Exxen”
Thanks for listening!

Any Questions?
CALL FOR PAPERS

First Workshop on
Data-Flow Execution Models for Extreme Scale Computing
(DFM 2011)
in conjunction with PACT 2011

Galveston, Island, Texas, USA October 10, 2011,
Submission: August 15, 2011
http://www.cs.acy.ac.cy/dfm2011

The purpose of DFM 2011 is to bring together researchers that are interested in novel computational models based on the Data-Flow principles of execution.

In the past designers were able to build faster and faster computers by relying on improvements on fabrication technologies and architectural/organization optimizations. The inability of the sequential model to tolerate long latencies has slowed down the performance gains. This limitation leads to the switch to multiple cores per chip and thus the move into the concurrency era. New concurrent models/paradigms are needed in order to fully utilize the potential of Multi-core chips. The Data-flow model is a formal model that can handle concurrency and tolerate memory and synchronization latencies. Data-Flow systems can also be simpler and more power efficient than conventional systems.

Recent work has shown that the Data-Flow principles can be used to develop systems that can outperform systems based on conventional techniques. Thus, it is time to revisit Data-driven computation and bring it to the Multi-core and extreme scale computing.

DFM 2011 solicits novel papers that include but are not limited to:

- Novel Data-Flow inspired Execution models and architectures
- Functional and single assignment based Languages.
- Strict and non-strict execution models.
- Compilers and tools for Data-Flow/Data-Driven systems.
- Hybrid Data-driven/Control-Driven systems.
- Survey papers on Data-Flow/Data-Driven systems.

Extended versions of the best papers will be published in a special issue of the IJPP.

ORGANIZING COMMITTEE

Skevos Erripidou,
University of Cyprus
Guang Gao,
University of Delaware
Jean-Luc Gaudiot,
University of California at Irvine
Vivek Sarkar,
Rice University
Ian Watson,
University of Manchester
Kei Hiraki,
University of Tokyo
David Abramson,
Monash University
Costas Kyriacou,
Frederic University
Pedro Traacoso,
University of Cyprus (Publicity Chair)

IMPORTANT DATES

Submission deadline: Aug 15
Notification of Authors: Sep 5

SUBMISSION INFORMATION

DFM 2011 will accept both Full and Short papers. Full papers should be prepared using the ACM SIG Proceedings format, and should be no longer than 8 pages. Short Papers should be submitted in the form of extended abstracts (up to 4 pages).

KEYNOTE SPEAKERS

TBA
Previous work

- **H/W and S/W co-design**
  - **PIM**
    - An Efficient Data-Distribution Mechanism in a PIM (Processor-In-Memory) Architecture Applied to Motion Estimation
      Jung-Yup Kang, Sandeep Gupta, and Jean-Luc Gaudiot
      *IEEE Transactions on Computers, Vol. 57, No. 3, March 2008*
    - An Efficient PIM (Processor-In-Memory) Architecture for BLAST
      Jung-Yup Kang, Sandeep Gupta, and Jean-Luc Gaudiot
      *Proceedings of the 38th Asilomar Conference on Signals, Systems and Computers, pp. 503-507, Pacific Grove, California, November 7-10, 2004*
  - **H/W acceleration**
    - Workload Characterization of Cryptography Algorithms for Hardware Acceleration
      Jed Kao-Tung Chang, Chen Liu, Shaoshan Liu, and Jean-Luc Gaudiot
      *Proceedings of the 2nd ACM International Conference on Performance Engineering (ICPE 2011), Karlsruhe, Germany, March 14-16, 2011*
    - Achieving Middleware Execution Efficiency: Hardware-Assisted Garbage Collection Operations
      Jie Tang, Shaoshan Liu, Zhimin Gu, Xiao-Feng Li and Jean-Luc Gaudiot
      *Journal of Supercomputing, DOI: 10.1007/s11227-010-0493-0, November, 2010*
## Previous work

- **H/W optimization**
  - **Power**
    - **Prefetching in Embedded Mobile Systems Can Be Energy-Efficient**
      Jie Tang, Shaoshan Liu, Zhimin Gu, Chen Liu, and Jean-Luc Gaudiot
    - **On Energy Efficiency of Reconfigurable Systems with Run-Time Partial Reconfiguration**
      Shaoshan Liu, Richard Neil Pittman, Alessandro Forin, and Jean-Luc Gaudiot
      *Proceedings of the 21st IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2010), Rennes, France, July 7-9, 2010*
    - **Adaptive Techniques for Leakage Power Management in L2 Cache Peripheral Circuits**
      H. Homayoun, A. Veidenbaum, and J-L. Gaudiot
      *Proceedings of the XXVI IEEE International Conference on Computer Design (ICCD 2008), Lake Tahoe, California, October 12-15, 2008*
  - **Performance/resource**
    - **A Low-Complexity Microprocessor Design with Speculative Pre-Execution**
      Won Woo Ro and Jean-Luc Gaudiot
      *Journal of System Architecture, Volume 54, Issue 12, December 2008, Pages 1101-1112*
    - **Area and System Clock Effects on SMT/CMP Processors**
      James Burns and Jean-Luc Gaudiot
      *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT’01), pp. 211-218, Barcelona, Spain, September 8-12, 2001*
Previous work

- **Value Prediction, Speculation**
  - Compiler assisted prefetching/speculation
    - Speculative pre-execution assisted by compiler (SPEAR)
      Won W. Ro and Jean-Luc Gaudiot
      *Journal of Parallel and Distributed Computing, Volume 66, Issue 8, Pages 1076-1089, August 2006*
  - Value Prediction/Speculation framework for GPGPU
    - Value Prediction and Speculative Execution on GPU
      Shaoshan Liu, Christine Eisenbeis, and Jean-Luc Gaudiot
      *International Journal of Parallel Programming, DOI: 10.1007/s10766-010-0155-0, December, 2010*
    - A Theoretical Framework for Value Prediction in Parallel Systems
      Shaoshan Liu, Christine Eisenbeis, and Jean-Luc Gaudiot
      *Proceedings of the Thirty Ninth International Conference on Parallel Processing, San Diego, California, September 13-16, 2010*
  - Impact of Value Prediction on system architecture
    - Potential Impact of Value Prediction on Communication in Many-Core Architectures
      Shaoshan Liu and Jean-Luc Gaudiot
      *IEEE Transactions on Computers, Vol. 58, No. 6, pp. 759-769, June 2009*
    - Value Prediction in Modern Many-Core Systems
      Shaoshan Liu (Ph.D. Student) and Jean-Luc Gaudiot (Advisor)
Previous work

- Dataflow systems
  - Implementing Parallel Branch-And-Bound with Extended Sisal 2.0
    Yung-Syau Chen and Jean-Luc Gaudiot
    *Parallel Processing of the Letters, Vol. 8, No. 1, pp. 41-50, March 1998*
  - The Sisal Model of Functional Programming and its Implementation
    Jean-Luc Gaudiot, Wim Böhm, Walid A. Najjar, Tom DeBoni, John Feo, and Patrick Miller
    *Proceedings of the 2nd Aizu International Symposium on Parallel Algorithms/Architectures Synthesis (pAs '97), pp. 112-123, Aizu-Wakamatsu, Japan, March 17-21, 1997*
  - I-Structure Software Cache - A Split-Phase Transaction Runtime Cache System
    Wen-Yen Lin and Jean-Luc Gaudiot
    *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT '96), pp. 122-126, Boston, Massachusetts, October 20-23, 1996*
  - Nomadic Threads: A Migrating Multithreaded Approach to Remote Memory Accesses in Multiprocessors
    Stephen Jenks and Jean-Luc Gaudiot
    *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT '96), pp. 2-11, Boston, Massachusetts, October 20-23, 1996*
LQCD Simulations

- Lattice Quantum ChromoDynamics (LQCD) applications are usually ported onto a large-scale cluster, such as the IBM Blue Gene supercomputer. It involves intensive communication between the compute nodes.
- Each node in the system computes a spinor, and each spinor communicates with the four neighboring spinors during execution.
- During communication, the SU3 vectors are exchanged between two neighbors, and each SU3 vector consists of **six 32-bit floating point** values.
Communication Pattern of LQCD Simulations

- When running the LQCD simulations on the IBM BlueGene/L machine, the communications do not overlap with computation or memory access.
- When porting LQCD simulation with a $2^4$ lattice size on one compute node:
  - If communication is turned off, it reaches 31.5% of the peak performance.
  - If communication is turned on, it only reaches 12.6% of the peak performance.
  - => communication is the major performance bottleneck in LQCD simulations.
Value Predictability of LQCD Communication

- We randomly selected four spinors and capture the values they communicate with their neighbors (each communication consists of six 32-bit floating point values).
- We apply information theory measures to characterize the value predictability.
- It is “easy” to predict individual values; but “hard” to predict all six values.

<table>
<thead>
<tr>
<th>spinors</th>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>55.30%</td>
<td>55.20%</td>
<td>55.10%</td>
<td>55.00%</td>
<td>55.30%</td>
<td>55.30%</td>
<td>27.50%</td>
</tr>
<tr>
<td>112</td>
<td>55.20%</td>
<td>55.20%</td>
<td>55.20%</td>
<td>55.20%</td>
<td>55.10%</td>
<td>55.10%</td>
<td>27.50%</td>
</tr>
<tr>
<td>199</td>
<td>57.00%</td>
<td>57.00%</td>
<td>57.00%</td>
<td>57.20%</td>
<td>57.20%</td>
<td>57.20%</td>
<td>29.00%</td>
</tr>
<tr>
<td>222</td>
<td>57.00%</td>
<td>57.00%</td>
<td>57.20%</td>
<td>57.20%</td>
<td>57.10%</td>
<td>57.10%</td>
<td>29.00%</td>
</tr>
</tbody>
</table>
We applied a simple LVP on these values to predict the communication values. Conf is a decision circuit with LVP: decide whether to use the predicted value. The decision circuit is able to filter out a majority of incorrect prediction. With this simple design: 22% of correct prediction => performance gain. 2.6% of rollbacks => performance degradation.

<table>
<thead>
<tr>
<th>spinors</th>
<th>lvp accuracy</th>
<th>pred,conf</th>
<th>pred, !conf</th>
<th>!pred, conf</th>
<th>!pred, !conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>24.56%</td>
<td>21.40%</td>
<td>3.16%</td>
<td>3.16%</td>
<td>72.28%</td>
</tr>
<tr>
<td>112</td>
<td>24.70%</td>
<td>21.52%</td>
<td>3.18%</td>
<td>3.18%</td>
<td>72.12%</td>
</tr>
<tr>
<td>199</td>
<td>24.95%</td>
<td>22.82%</td>
<td>2.13%</td>
<td>2.13%</td>
<td>72.92%</td>
</tr>
<tr>
<td>222</td>
<td>25.00%</td>
<td>22.83%</td>
<td>2.14%</td>
<td>2.14%</td>
<td>72.89%</td>
</tr>
<tr>
<td>AVG</td>
<td>24.80%</td>
<td>22.14%</td>
<td>2.65%</td>
<td>2.65%</td>
<td>72.55%</td>
</tr>
</tbody>
</table>
LVP on LQCD Communication

\[ com\_lat = mispred\% \times com\_lat + pred\_oh + rollback\% \times rb\_oh \]

- Rollback overhead \((rb\_oh)\): \(~50\) cycles
- Prediction Overhead \((pred\_oh)\): \(~40\) cycles
- Percentage of mispredictions \((mispred\%)\): \(72.55\%\)
- Percentage of rollbacks \((rollback\%)\): \(2.6\%\)
- Communication latency \((com\_lat)\): \(186.5\) cycles
  - Value prediction would bring performance gain if the communication latency is higher than \(186.5\) cycles
  - With hardware implementation \((pred\_oh = 7\) cycles\), \(com\_lat\) becomes \(37.5\) cycles
World at war for top computing performance

China and Japan have designed the top supercomputers today (from top500.org)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer/Year Vendor</th>
<th>Cores</th>
<th>R_max</th>
<th>R_peak</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K Computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu</td>
<td>543352</td>
<td>6162.00</td>
<td>8773.83</td>
<td>9098.56</td>
</tr>
<tr>
<td>2</td>
<td>National Supercomputing Center in Tianjin China</td>
<td>Tianhe-1A - NUDT TH MPP, X6670 2.03GHz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT</td>
<td>186368</td>
<td>2568.00</td>
<td>4701.00</td>
<td>4040.00</td>
</tr>
<tr>
<td>3</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Jaguar - Cray XT5-HE Opteron 6-core 2.8 GHz / 2009 Cray Inc.</td>
<td>224120</td>
<td>1759.00</td>
<td>2331.00</td>
<td>6950.00</td>
</tr>
<tr>
<td>4</td>
<td>National Supercomputing Centre in Shenzhen (NSCC) China</td>
<td>Nebulae - Dawning TC3600 Blade, Intel X6650, Nvidia Tesla C2050 GPU / 2010 Dawning</td>
<td>120340</td>
<td>1271.00</td>
<td>2984.30</td>
<td>2500.00</td>
</tr>
<tr>
<td>5</td>
<td>OSIC Center, Tokyo Institute of Technology Japan</td>
<td>TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 8C X6670, Nvidia GPU, Linux/Windows / 2010 NEC/HP</td>
<td>73278</td>
<td>1192.00</td>
<td>2287.53</td>
<td>1398.01</td>
</tr>
</tbody>
</table>
Speculation on GPU

- Decompress_SPEC: this is a kernel loop in the decompress routine inside bzip2 of SPEC CPU 2006. It traverses an array and un-compresses the data. During data un-compression, it traverses an array that contains the addresses of the source data. Then, it fetches the data through pointer indirection.
Value Prediction on GPU

- PDE_PRED: a wavefront PDE solver.