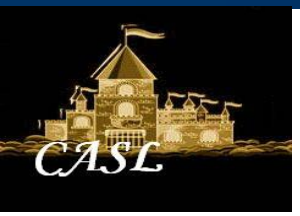


The Era of Heterogeneous Compute: Challenges and Opportunities

Sudhakar Yalamanchili

Computer Architecture and Systems Laboratory
Center for Experimental Research in Computer Systems
School of Electrical and Computer Engineering
Georgia Institute of Technology

System Diversity

Amazon EC2 GPU Instances

Mobile Platforms

Heterogeneity is Mainstream

Keeneland System

Tianhe-1A

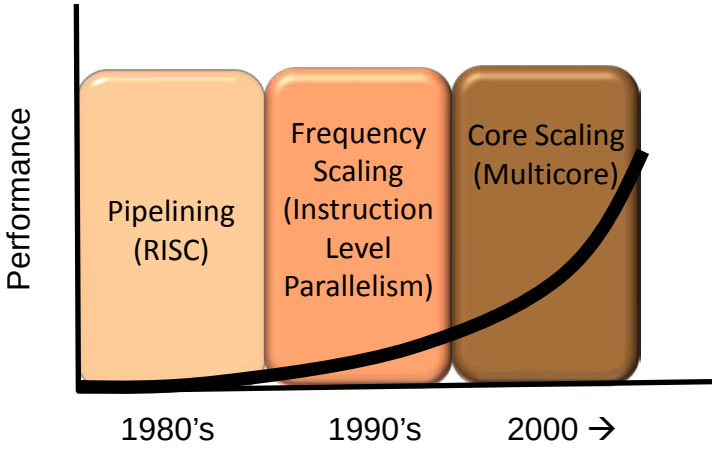
Outline

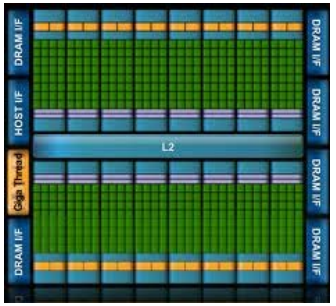
- **Drivers and Evolution to Heterogeneous Computing**
- The Ocelot Dynamic Execution Environment
- Dynamic Translation for Execution Models
- Dynamic Instrumentation of Kernels
- Related Projects

Evolution to Multicore

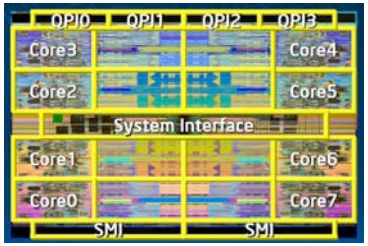
Power Wall

$$P = \alpha C V_{dd}^{\ 2} f + V_{dd} I_{st} + V_{dd} I_{leak}$$

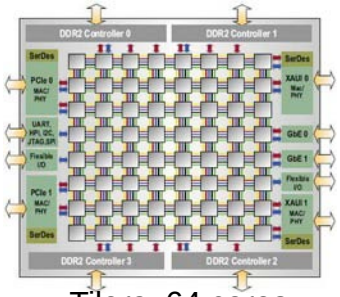




NVIDIA Fermi: 480 cores



Intel Nehalem-EX: 8 cores

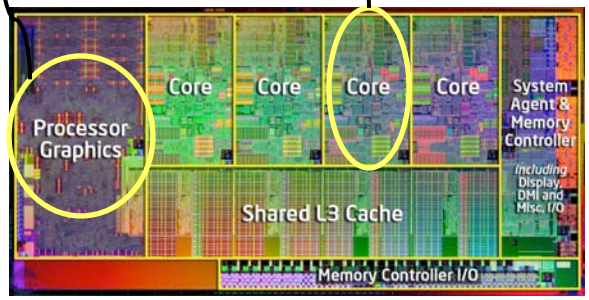


Tilera: 64 cores

Consolidation on Chip

Programmable Pipeline (GEN6)

Vector Extensions AES Instructions



Intel Sandy Bridge

Programmable Accelerator

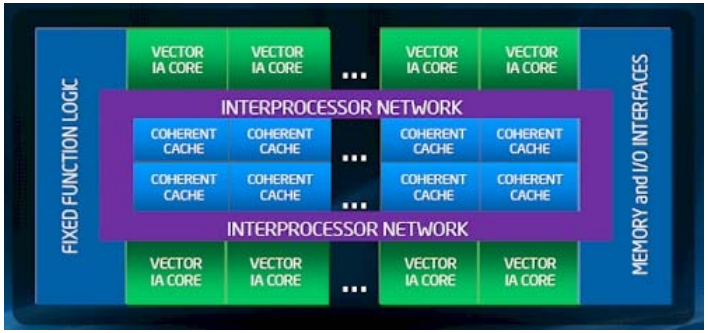
Multiple Models of Computation
Multi-ISA

16, PowerPC cores

Accelerators
- *Crypto Engine*
- *RegEx Engine*
- *XML Engine*
- *CP<[press Engine*

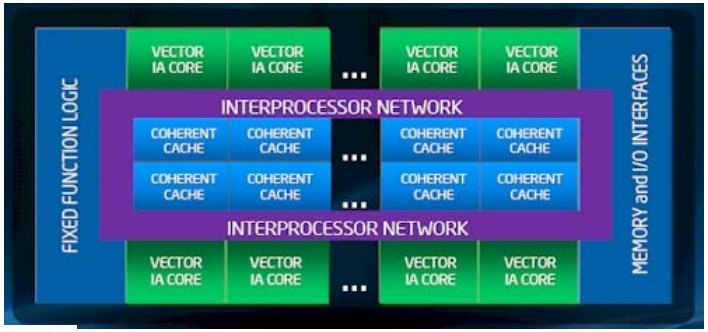
IBM *PowerEN*

Intel Knights Corner

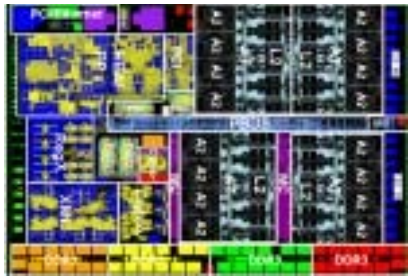


Major Customization Trends

Uniform ISA
Asymmetric



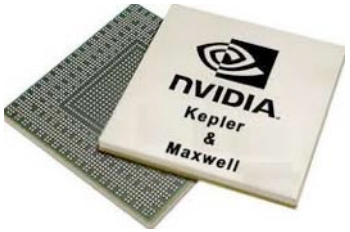
Knights Corner



PowerEN

- Minimal disruption to the software ecosystems
- Limited customization?

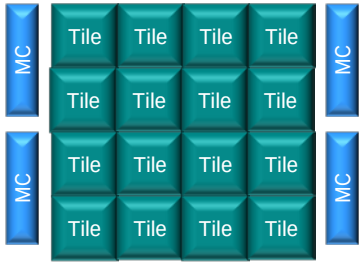
Multi-ISA
Heterogeneous



- Disruptive impact on the software stack?
- Higher degree of customization

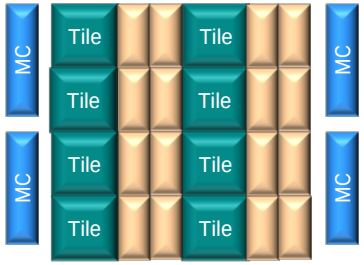
Asymmetry vs. Heterogeneity

Performance Asymmetry



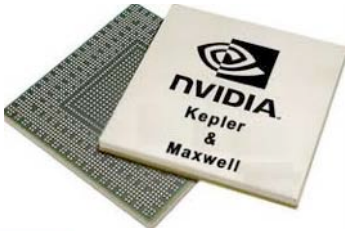
- Multiple voltage and frequency islands
- Different memory technologies
  - STT-RAM, PCM, Flash

Functional Asymmetry



- Complex cores and simple cores
- Shared instruction set architecture (ISA)
  - Subset ISA
  - Distinct microarchitecture
  - Fault and migrate model of operation[1]

Heterogeneous



- Multi-ISA
- Microarchitecture
  - Memory & Interconnect hierarchy

Uniform ISA — Multi-ISA

[1]Li., T., et.al., "Operating system support for shared ISA asymmetric multi-core architectures," *in WIOSCA,* 2008.

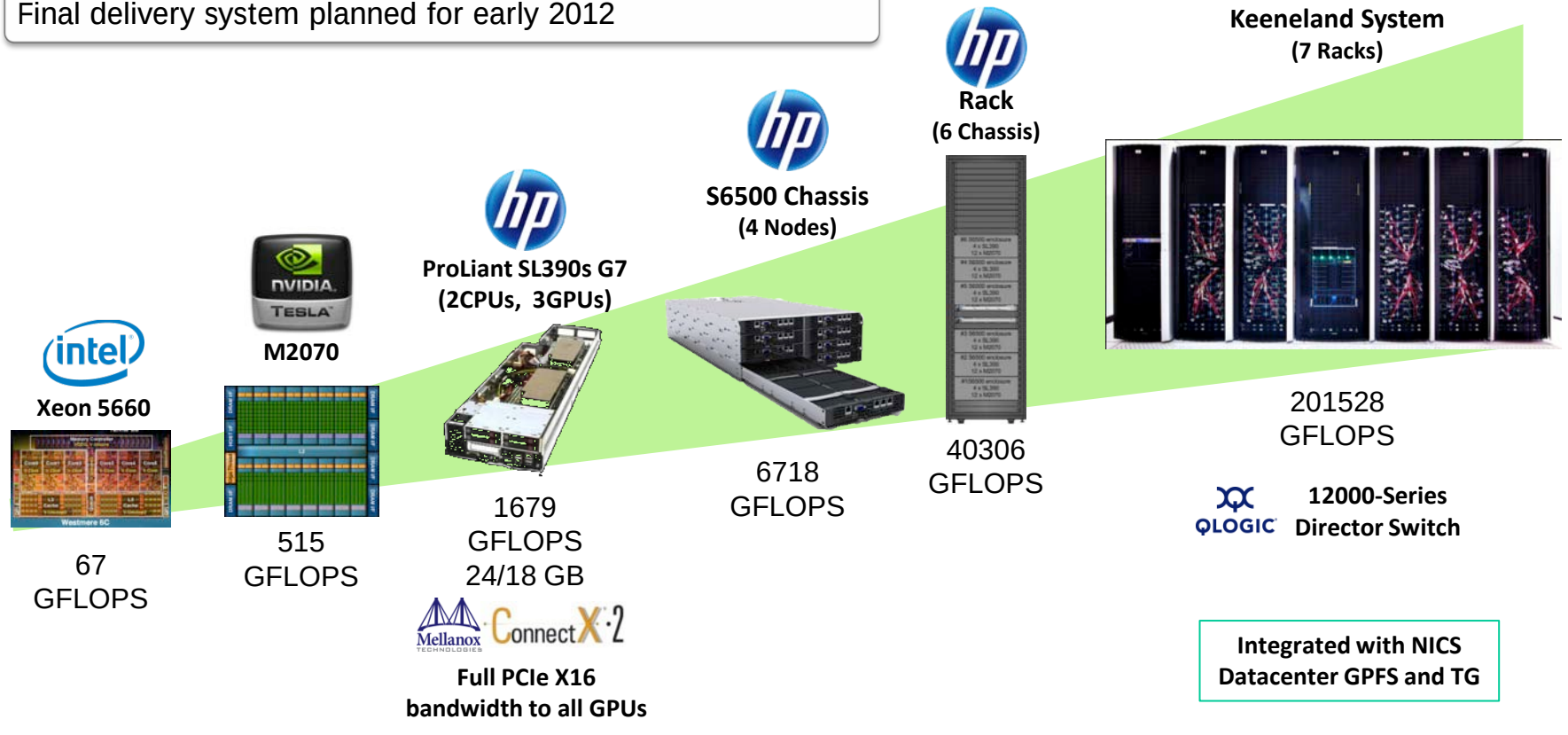
HPC Systems: Keeneland

Courtesy J. Vetter (GT/ORNL)

201 TFLOPS in 7 racks (90 sq ft incl service area)

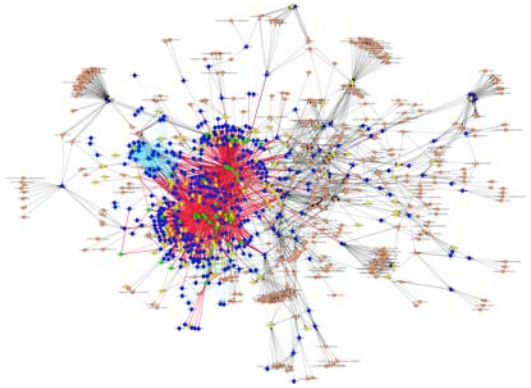
677 MFLOPS per watt on HPL (#9 on Green500, Nov 2010)

Final delivery system planned for early 2012



A Data Rich World

Large Graphs



Mixed Modalities and levels of parallelism

Irregular, Unstructured Computations and Data

facebook

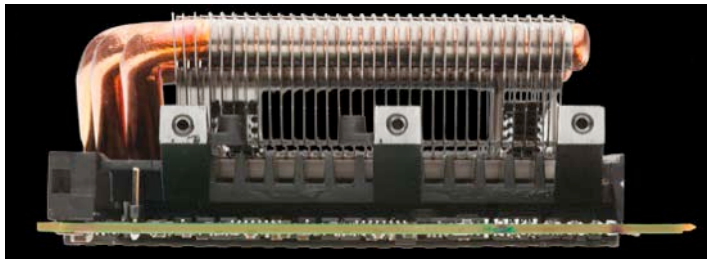
Pharma

conventioninsider.com

Images from math.nist.gov, blog.thefuturescompany.com,melihsozdinler.blogspot.com

Trend analysis

Enterprise: Amazon EC 2 GPU Instance



NVIDIA Tesla

Amazon EC2 GPU Instances

Elements	Characteristics
OS	CentOS 5.5
CPU	2 x Intel Xeon X5570 (quad-core "Nehalem" arch, 2.93GHz)
GPU	2 x NVIDIA Tesla "Fermi" M2050 GPU Nvidia GPU driver and CUDA toolkit 3.1
Memory	22 GB
Storage	1690 GB
I/O	10 GigE
Price	$2.10/hour

Impact on Software

At System Scale

At Chip Scale

- We need ISA level stability
  - Commercially, it is infeasible to constantly re-factor and re-optimize applications
  - Avoid software “silos”
- Performance portability
  - New architectures need new algorithms
- What about our existing software?

Will Heterogeneity Survive?

The New York Times

Technology

The Attack of the 'Killer Micros'

By JOHN MARKOFF
Published: May 06, 1991

The Convex Computer Corporation plans to introduce its first supercomputer on Tuesday. But Cray Research Inc., the king of supercomputing, says it is more worried by "killer micros" -- compact, extremely fast work stations that sell for less than $100,000.

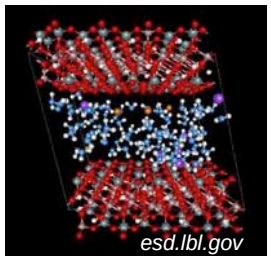
Indeed, Cray says the smaller machines will be even more of a threat to Convex than to it.

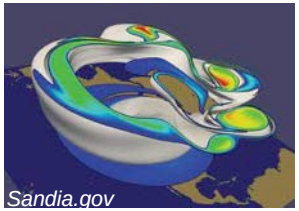
Cray Research has long dominated the market for the world's fastest and costliest computers. Now, John A. Rollwagen, Cray's chairman, seems to be looking past Convex, a Dallas-based maker of mini-supercomputers, which approach the speed of supercomputers and carry a significantly lower price.

Will We See Killer AMPs (Asymmetric Multicore Processors)?

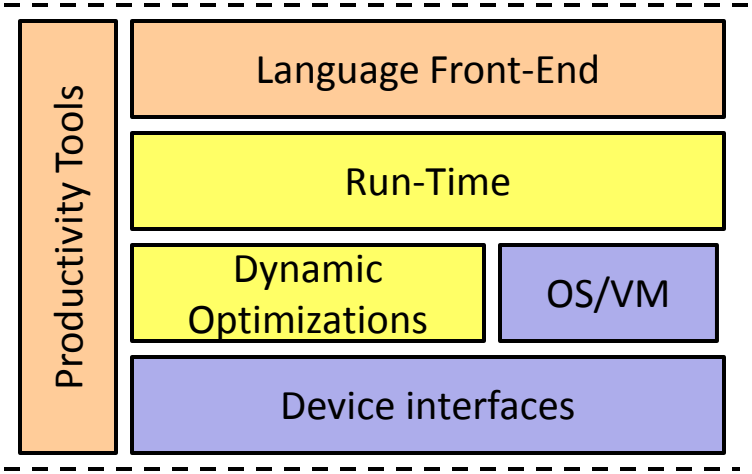
System Software Challenges of Heterogeneity

- Execution Portability
  - –Systems evolve over time
  - –New systems
- Performance Optimization
  - New algorithms
- Introspection
  - Productivity tools
- Application Migration
  - –Protect investments in existing code bases





Emerging Software Stacks



Outline

- Drivers and Evolution to Heterogeneous Computing
- **The Ocelot Dynamic Execution Environment**
- Dynamic Translation for Execution Models
- Dynamic Instrumentation of Kernels
- Related Projects

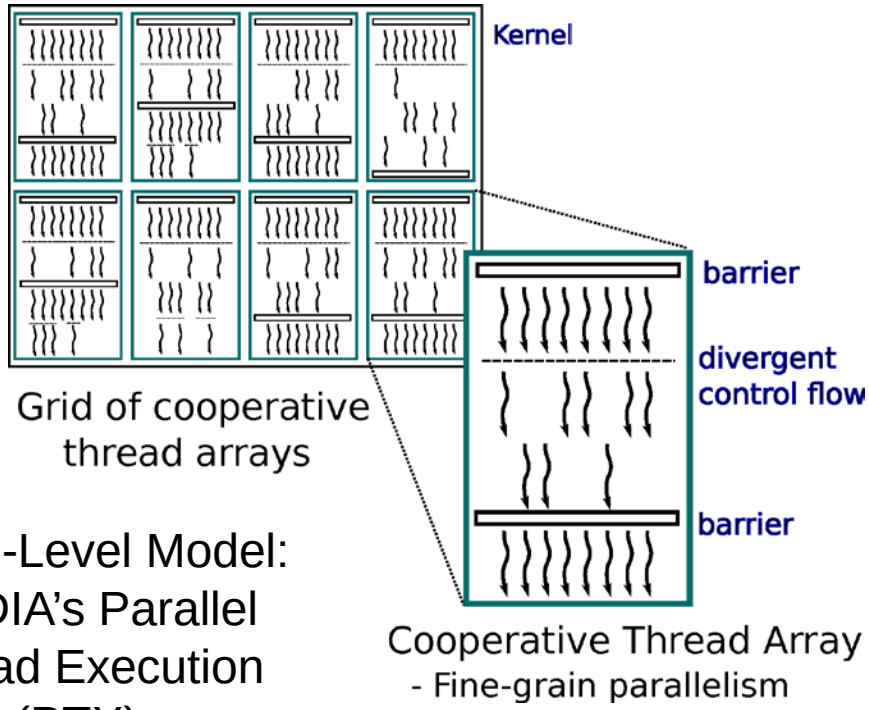
Ocelot: Project Goals

- Encourage proliferation of GPU computing
  - Lower the barriers to entry for researchers and developers
  - Establish links to industry standards, e.g., OpenCL
- Understand performance behavior of massively parallel, data intensive applications across multiple processor architecture types
- Develop the next generation of translation, optimization, and execution technologies for large scale, asymmetric and heterogeneous architectures.

http://code.google.com/p/gpuocelot/

Key Philosophy

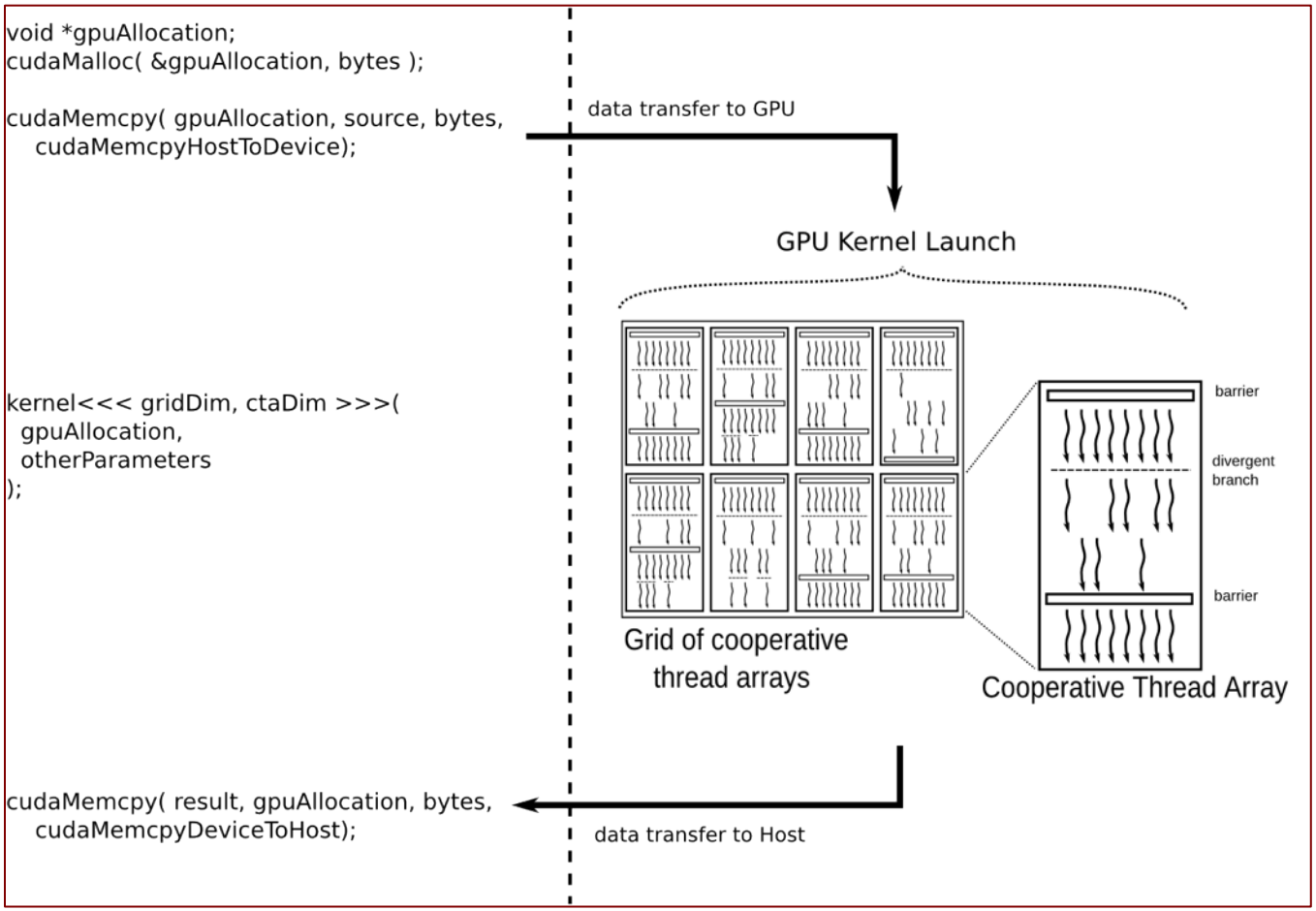
- Start with an explicitly parallel internal representations
  - Auto-serialization vs. auto-parallelization
  - Proliferation of domain specific languages and explicitly parallel language extensions like CUDA, OpenCL, and others



Kernel-Level Model: NVIDIA's Parallel Thread Execution (PTX)

Kernel level model: bulk synchronous processing (BSP)

NVIDIA's Compute Unified Device Architecture (CUDA)



Bulk synchronous execution model

- For access to CUDA tutorials

  http://developer.nvidia.com/cuda-education-training

Need for Execution Model Translation

CUDA

C/C++

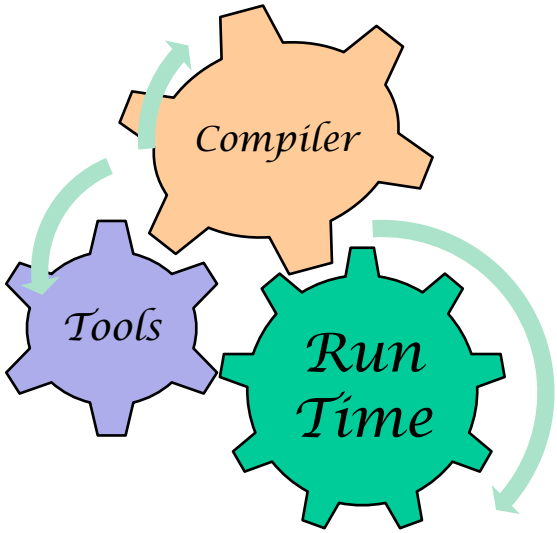
Haskell

C++AMP

Datalog

OpenCL

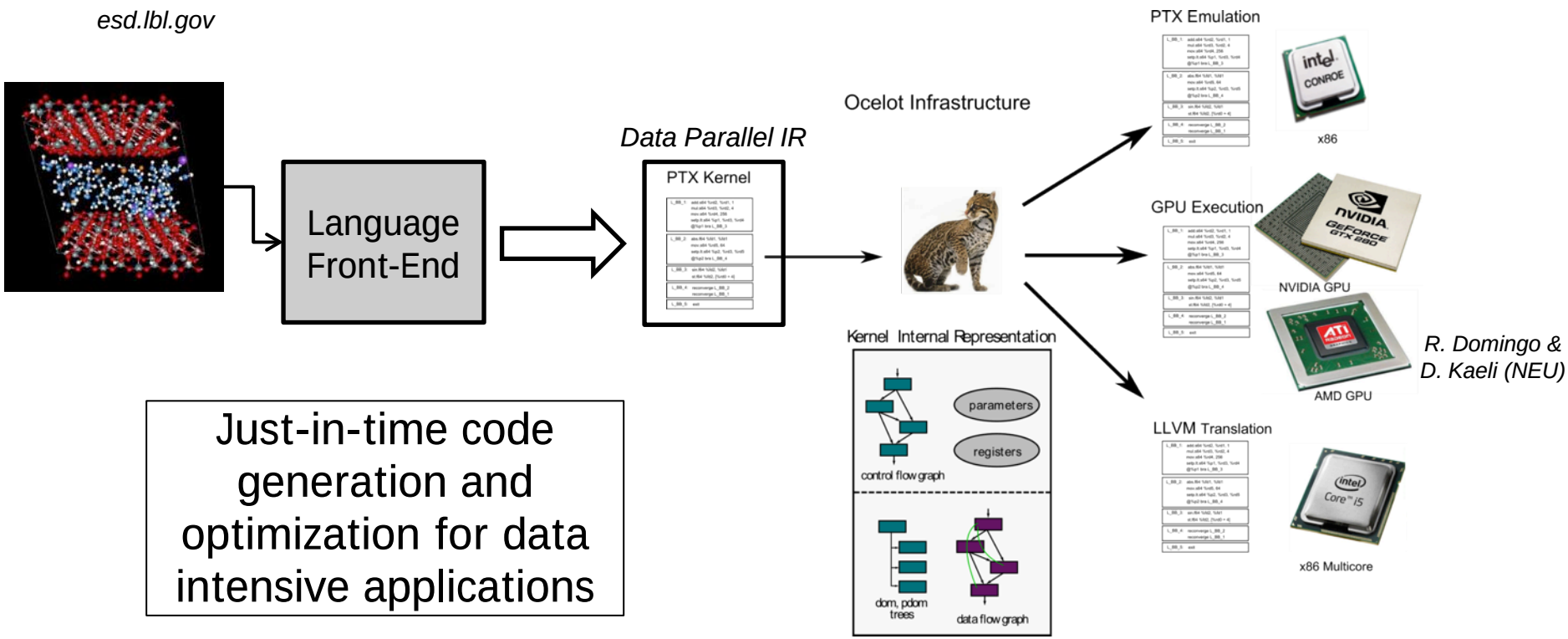
Languages: Designed for Productivity



Execution Models (EM):
Dynamic Translation of EMs to bridge this gap

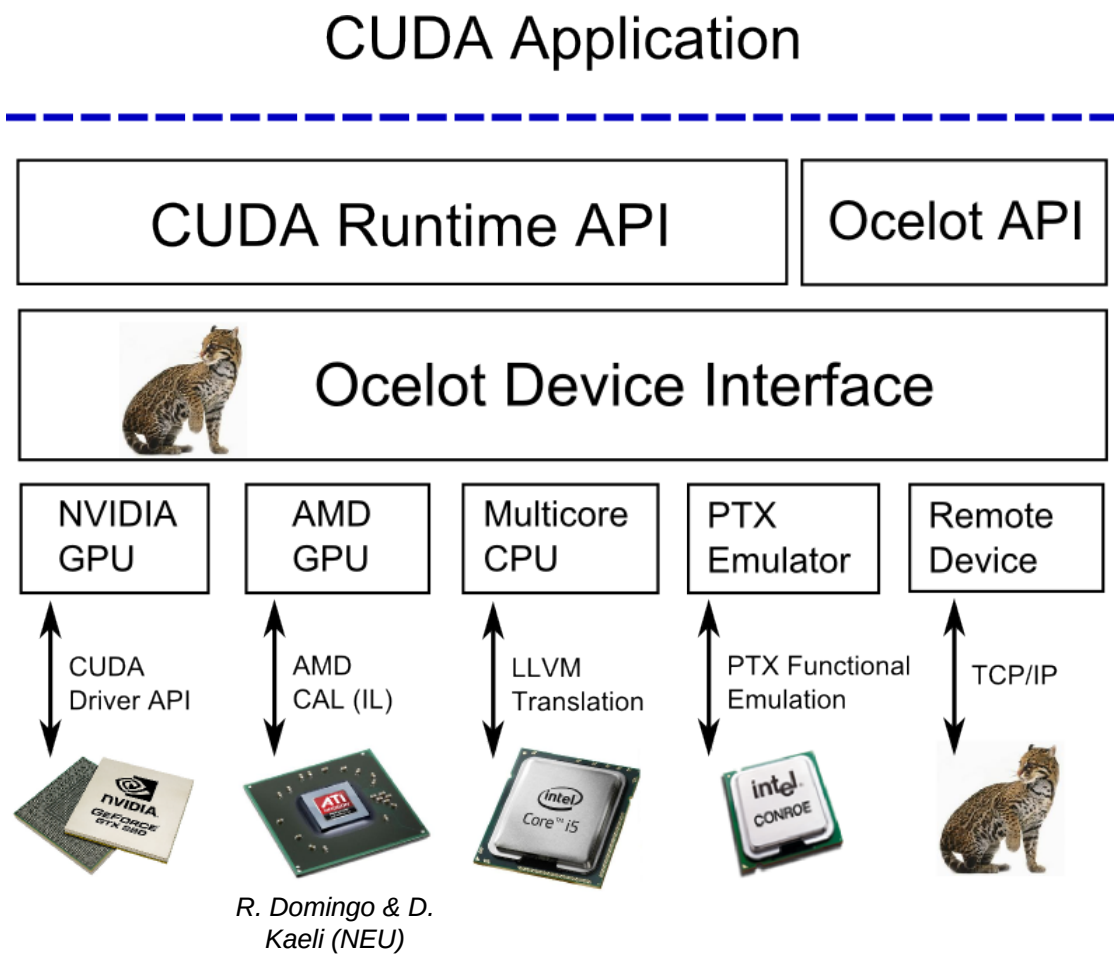
Hardware Architectures – Design under speed, cost, and energy constraints

Ocelot Vision: Multiplatform Dynamic Compilation



- Environment for i) compiler research, ii) architecture research, and iii) productivity tools

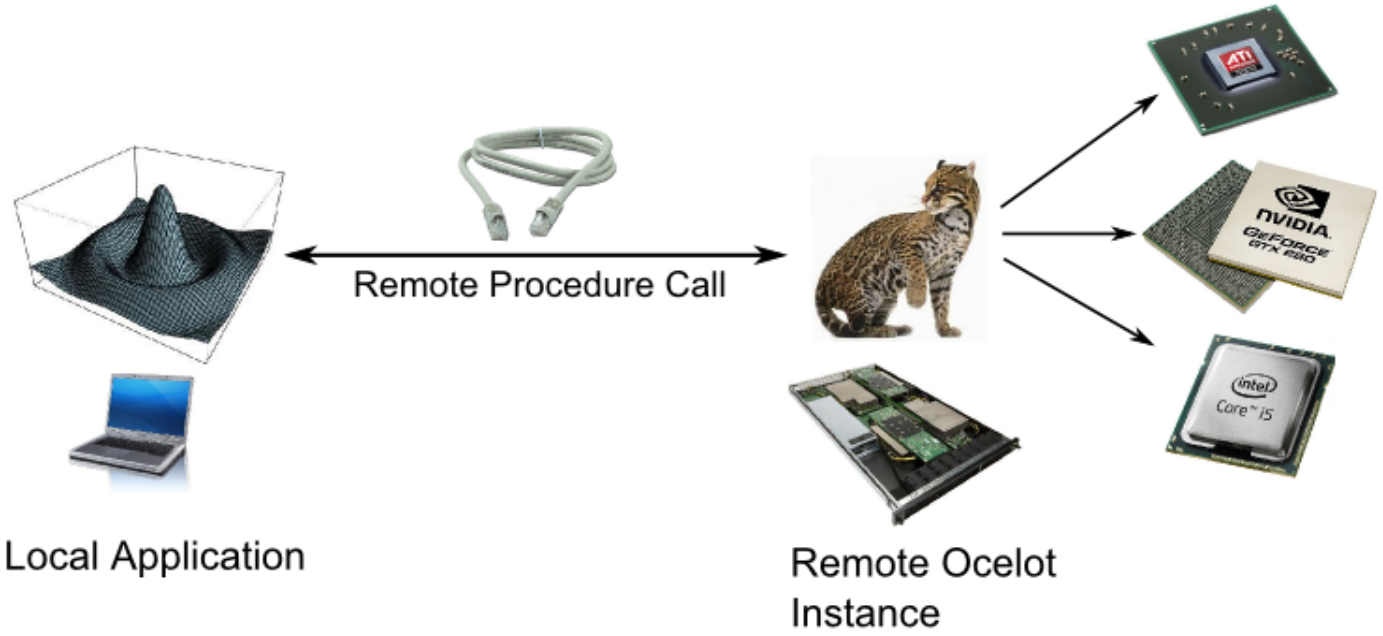
Ocelot CUDA Runtime Overview



- A complete reimplementation of the CUDA Runtime API
- Compatible with existing applications
  - Link against libocelot.so instead of libcudart
- Ocelot API Extensions
- Device switching

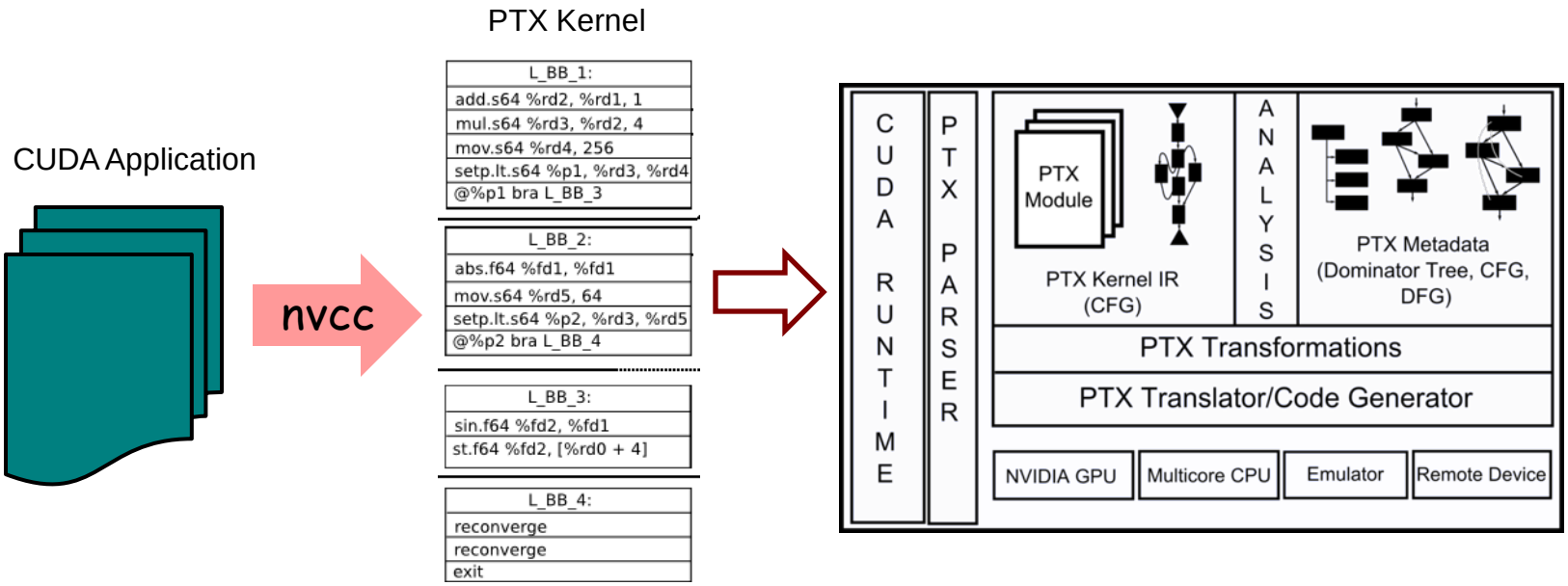
Kernels execute anywhere → Key to portability!

Remote Device Layer



- Remote procedure call layer for Ocelot device calls
- Execute local applications that run kernels remotely
- Multi-GPU applications can become multi-node

Ocelot Internal Structure[1]

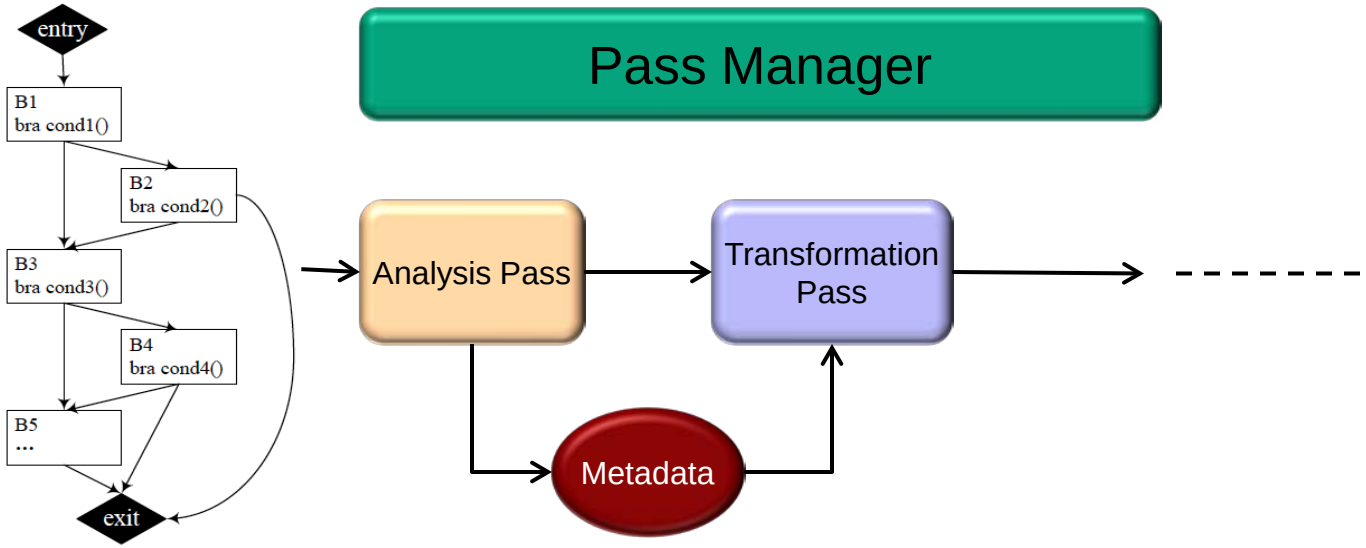


- Ocelot is built with *nvcc* and the LLVM backend
  - Structured around PTX IR→ LLVM IR Translator
- Compile stock CUDA applications without modification
- Other front-ends in progress: OpenCL and Datalog

[1]G. Diamos, A. Kerr, S. Yalamanchili, and N. Clark, “Ocelot: A Dynamic Optimizing Compiler for Bulk Synchronous Applications in Heterogeneous Systems,” *PACT*, September 2010. .

For Compiler Researchers

- Pass Manager Orchestrates analysis and transformation passes
  - Analysis Passes generate meta-data:
    - E.g., Data-flow graph, Dominator and Post-dominator trees, Thread frontiers
    - Meta-data consumed by transformations
  - Transformation Passes modify the IR
    - E.g., Dead code elimination, Instrumentation, etc.

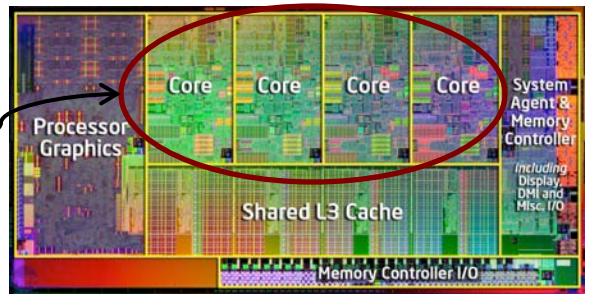


Outline

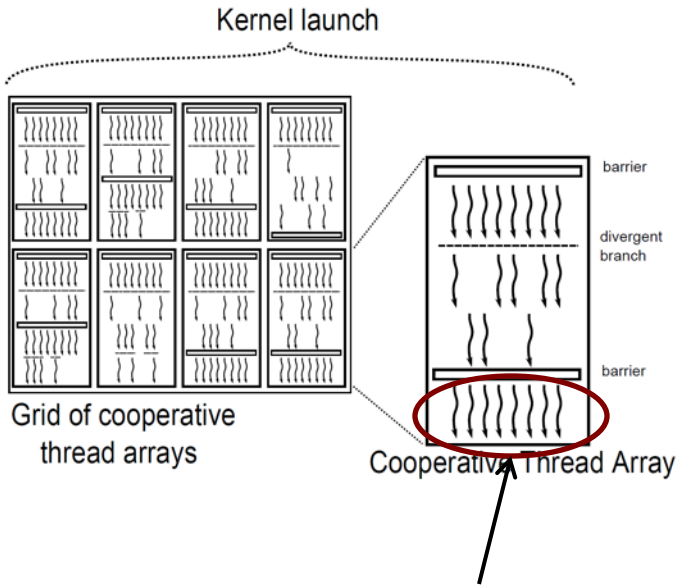
- Drivers and Evolution to Heterogeneous Computing
- The Ocelot Dynamic Execution Environment
- **Dynamic Translation for Execution Models**
- Dynamic Instrumentation of Kernels
- Related Projects

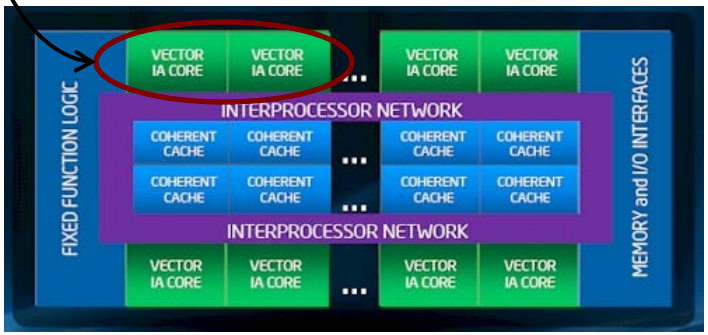
Execution Model Translation

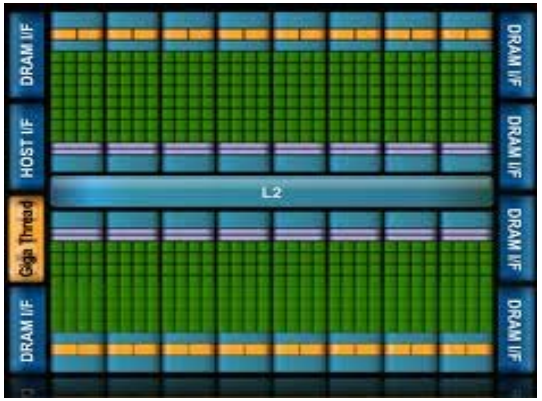
Utilize all resources



JIT for Parallel Code



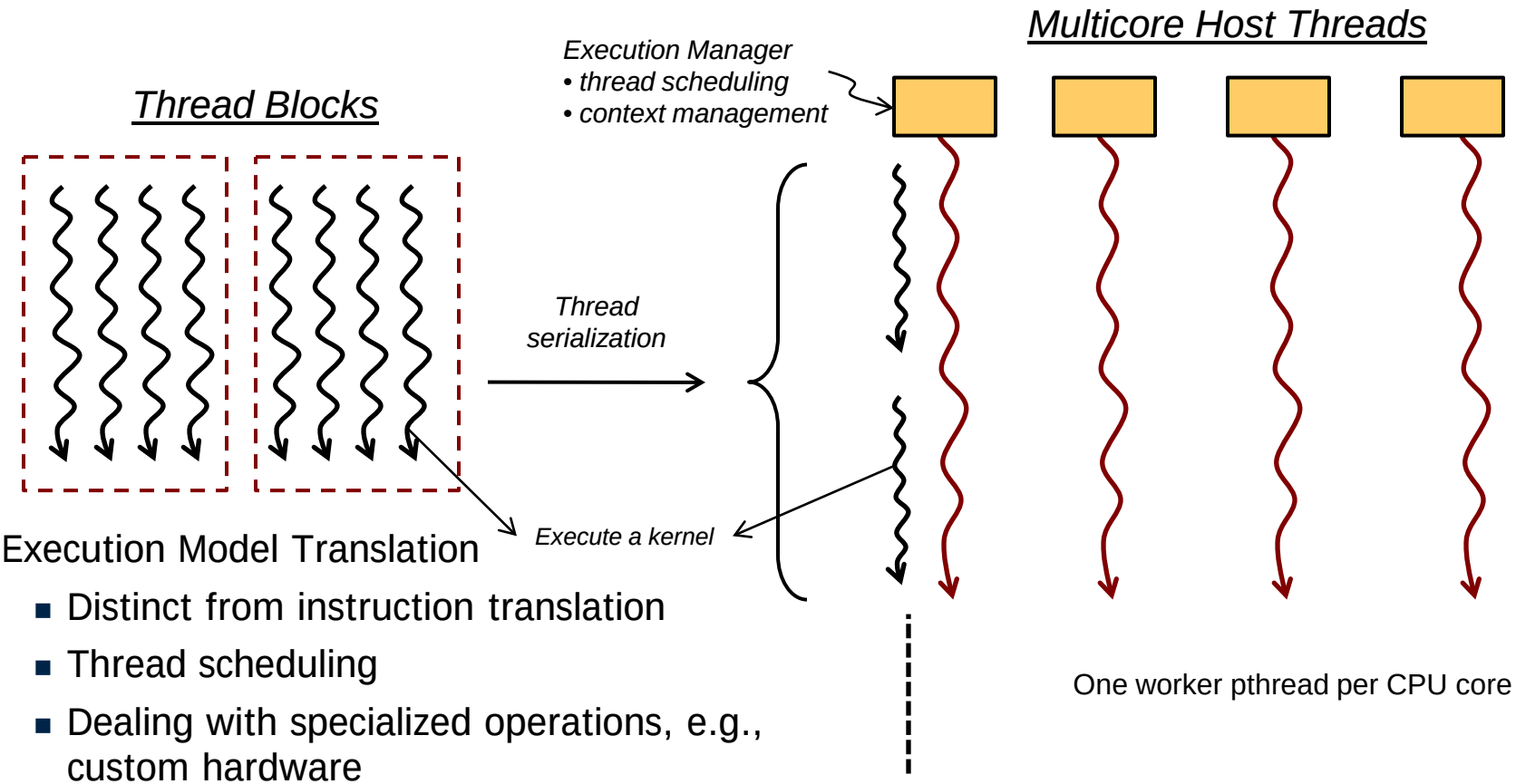




Serialization Transforms

kernel fusion/fission

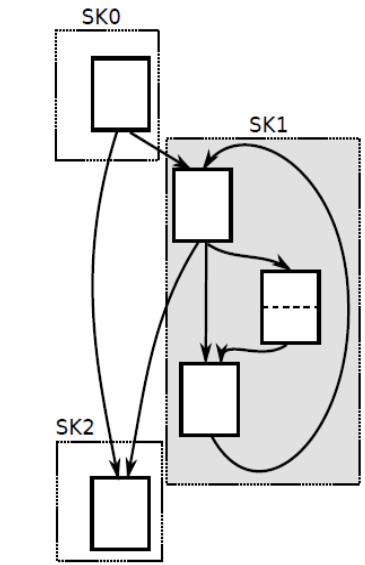
Translation to CPUs: Thread Fusion



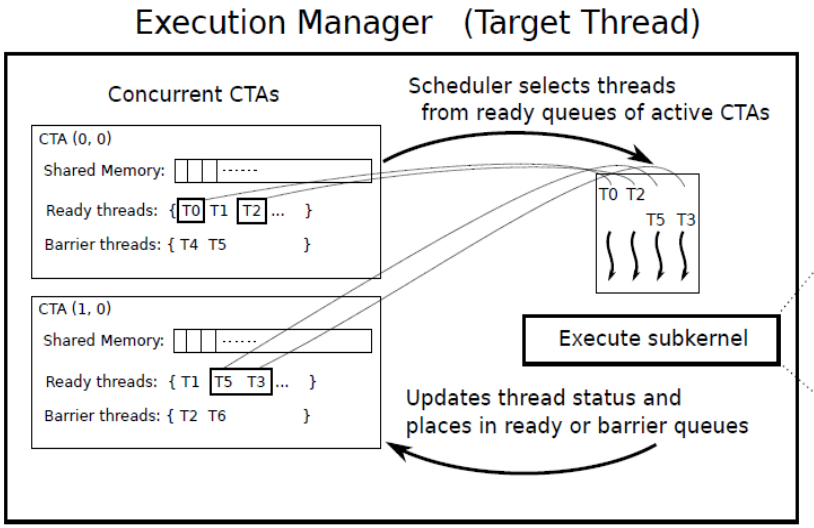
- Execution Model Translation
  - Distinct from instruction translation
  - Thread scheduling
  - Dealing with specialized operations, e.g., custom hardware
  - Handing control flow and synchronization
  - Mapping thread hierarchies, address spaces, fixed functions, etc.

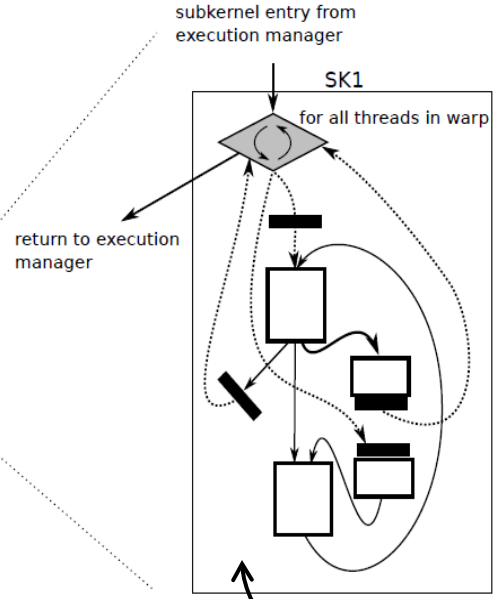
G. Diamos, A. Kerr, S. Yalamanchili and N. Clark, "Ocelot: A Dynamic Optimizing Compiler for Bulk-Synchronous Applications in Heterogeneous," *PACT)* 2010.

Dynamic Thread Fusion



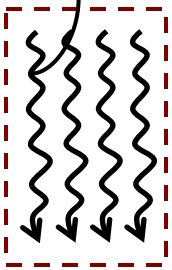
Flowgraph of Subkernels





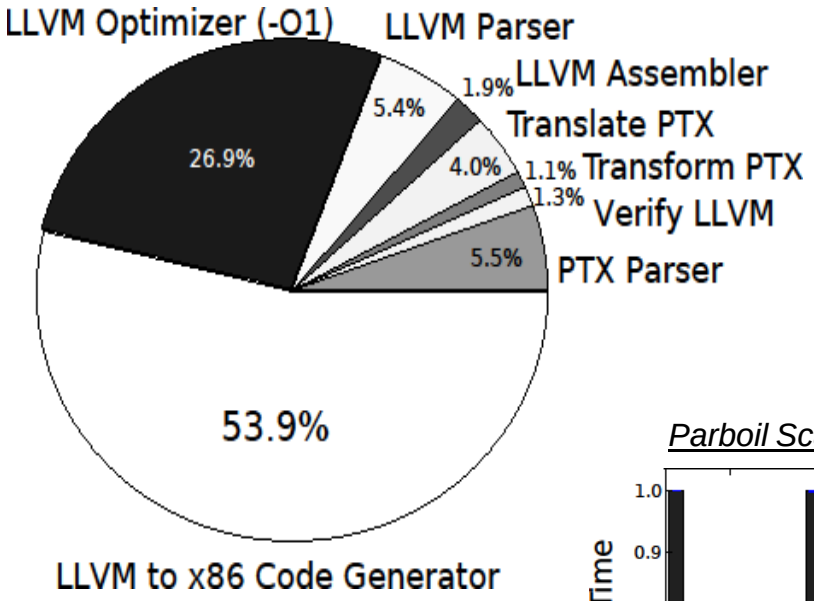
Subkernel with local scheduler

Each thread executes this code



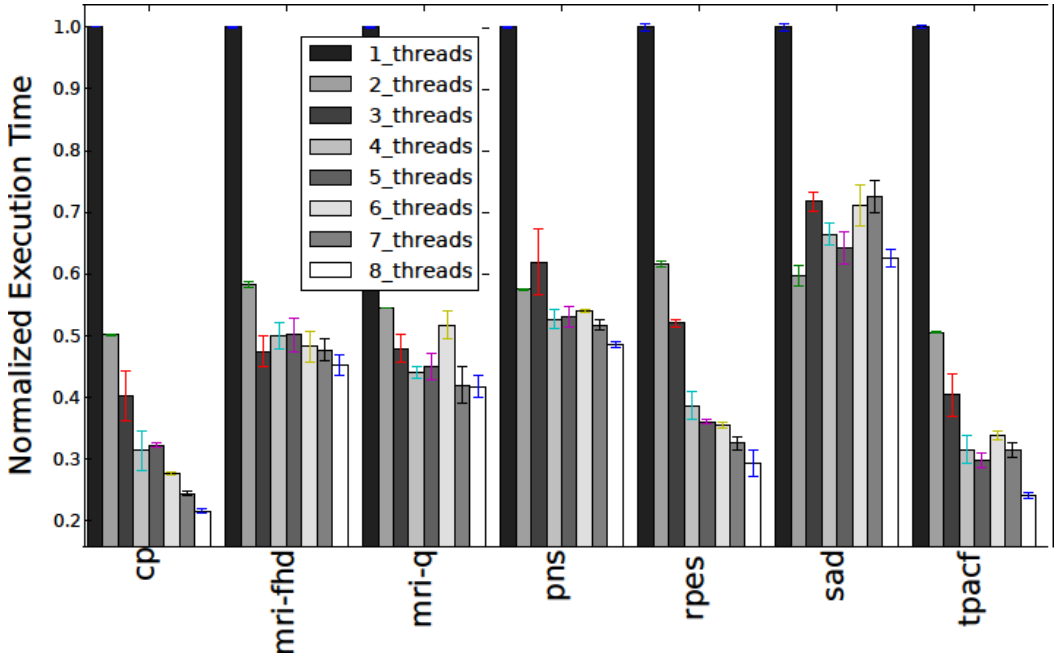
- Dynamic warp formation
  - What are the implications for cache behavior?
- Optimize for control flow divergence
- Improve opportunities for vectorization

Overheads Of Translation

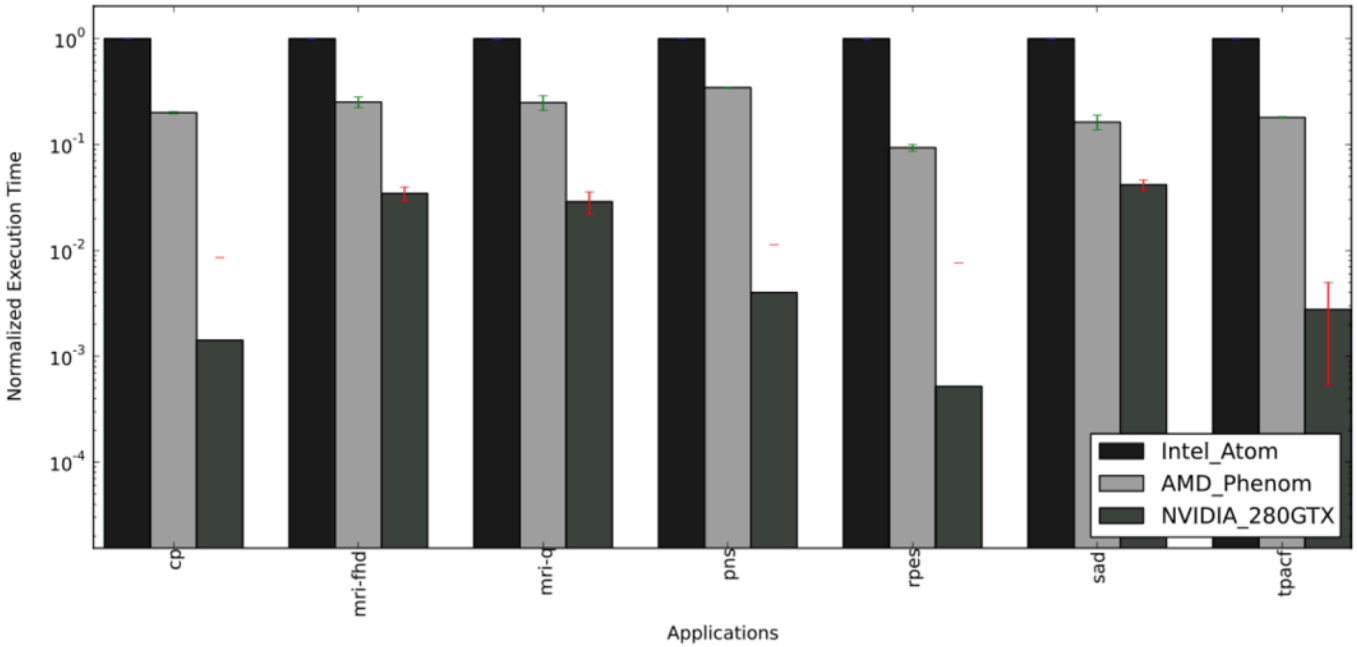


- Sub-kernel size = kernel size
- Amortized with the use of a code cache
- Challenge: Speeding up translation

Parboil Scaling



Target Scaling Using Ocelot



- The 12x Phenom vs. Atom advantage → 9.1x-11.4x speedup
- The 40x GPU vs. Phenom advantage → 8.9x-186x speedup
  - Upper end due to use of the fixed function hardware accelerators vs. software implementation on the Phenom

Key Performance Issues

- JIT compilation overheads
  - Dead code
  - Kernel size
  - Thread serialization granularity

Sub-kernels

- JIT throughput due to bottlenecks
  - Access to the code cache
  - Access to the JIT
  - Balancing throughput vs. JIT compilation overhead

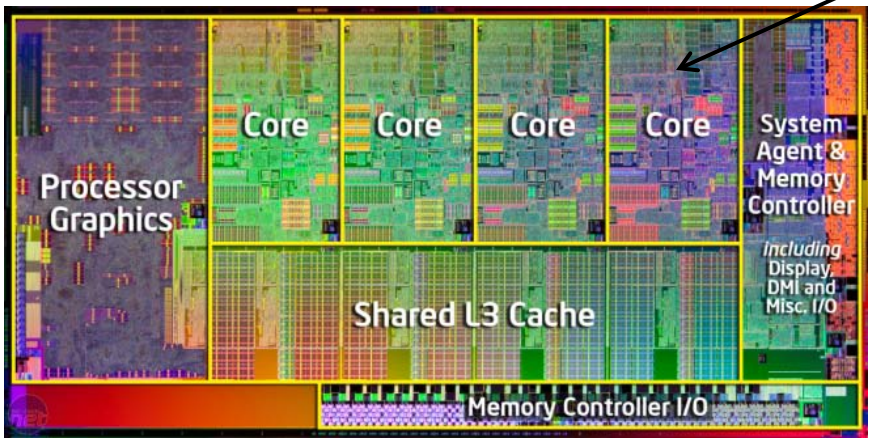
Specialization + Code caching

- Program behaviors
  - Synchronization
  - Control flow divergence
  - Promoting locality

Sub-kernels + Dynamic Warp Formation

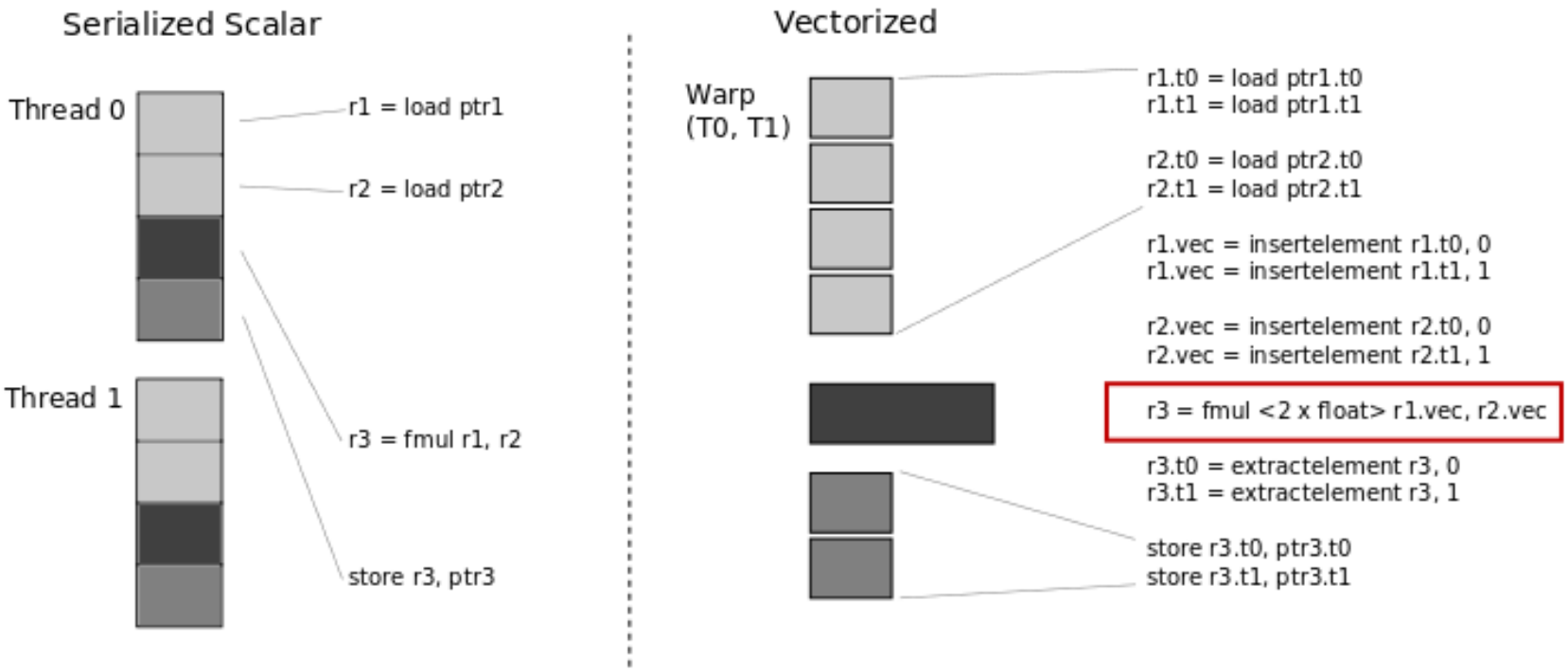
Can We Do Even Better?

SSE/AVX Vector extensions per core



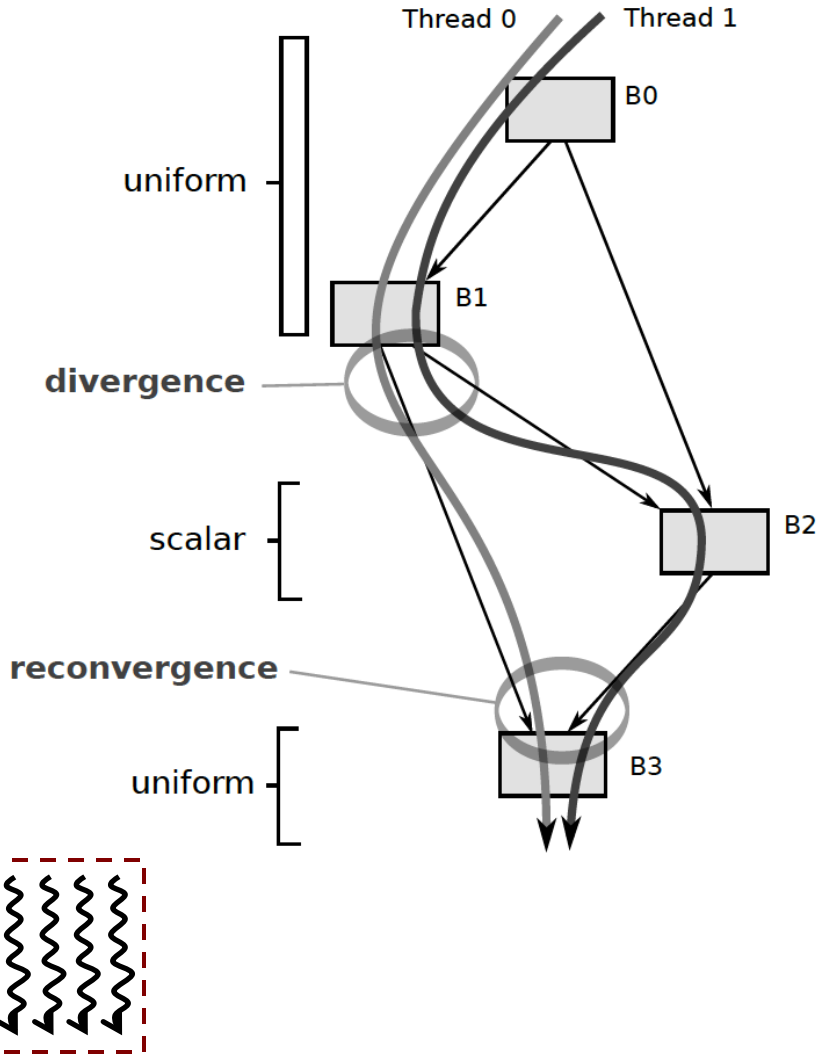
- Use the attached vector units within each core

Vectorization of Data Parallel Kernels



- What about control flow divergence?
- What about memory divergence?

Intelligent Warp Formation

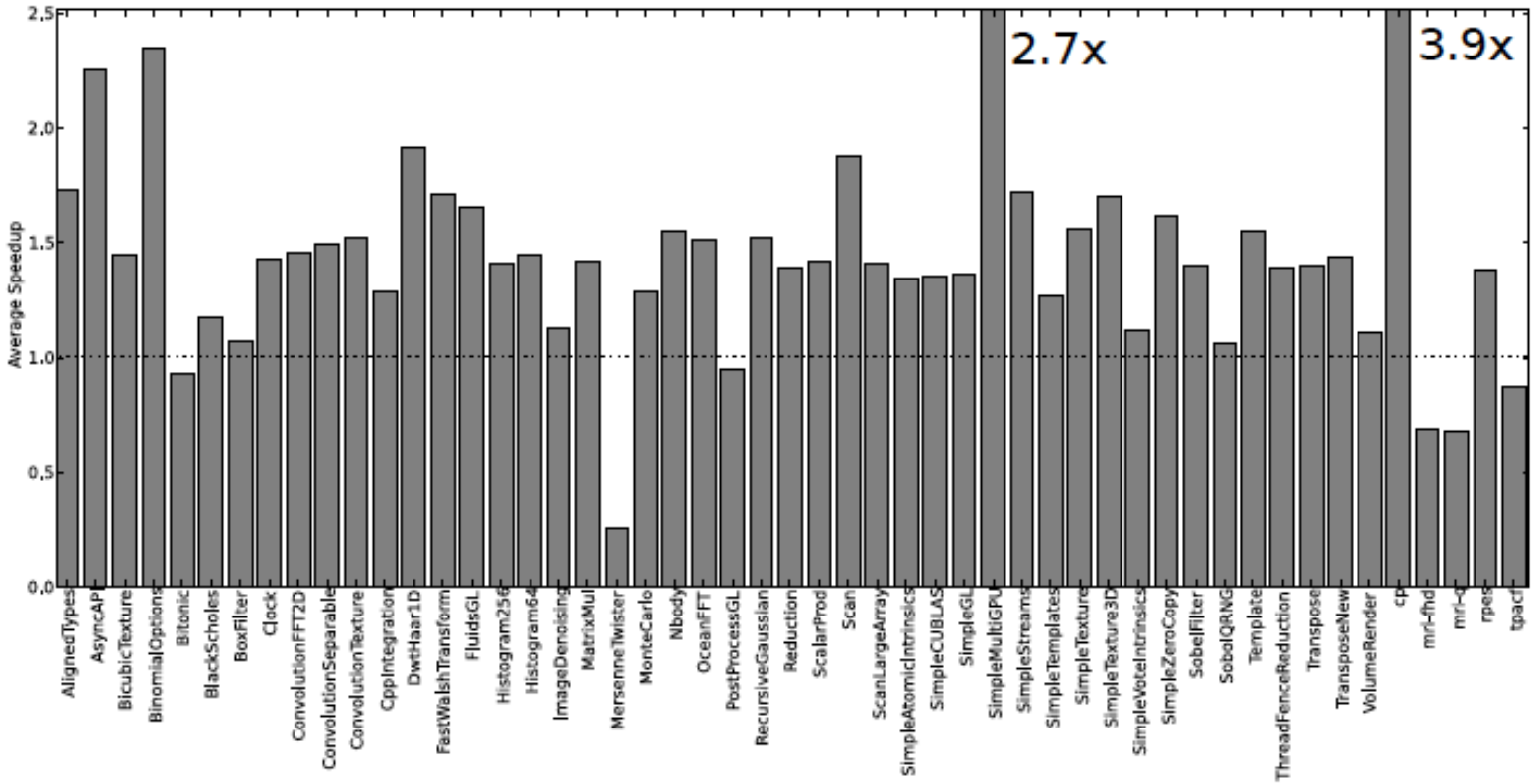


- Yield-on-diverge: divergent threads exit to the execution manager
- The execution manager selects threads (a warp) for vectorization
- A priori specialization and code caching to speed up translations

A. Kerr, G. Diamos, and. S. Yalamanchili, " Dynamic Compilation of Data Parallel Kernels for Vector Processors," *International Symposium on Code Generation and Optimization*, April 2012.

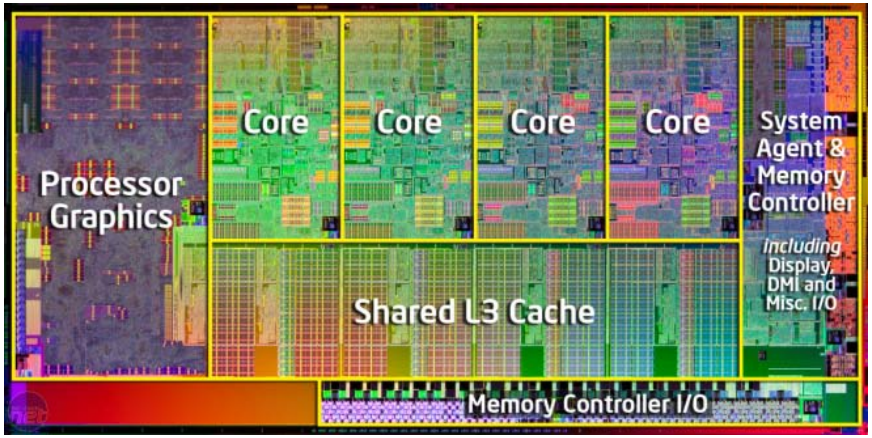
Vectorization: Performance

Average Speedup of 1.45X over base translation



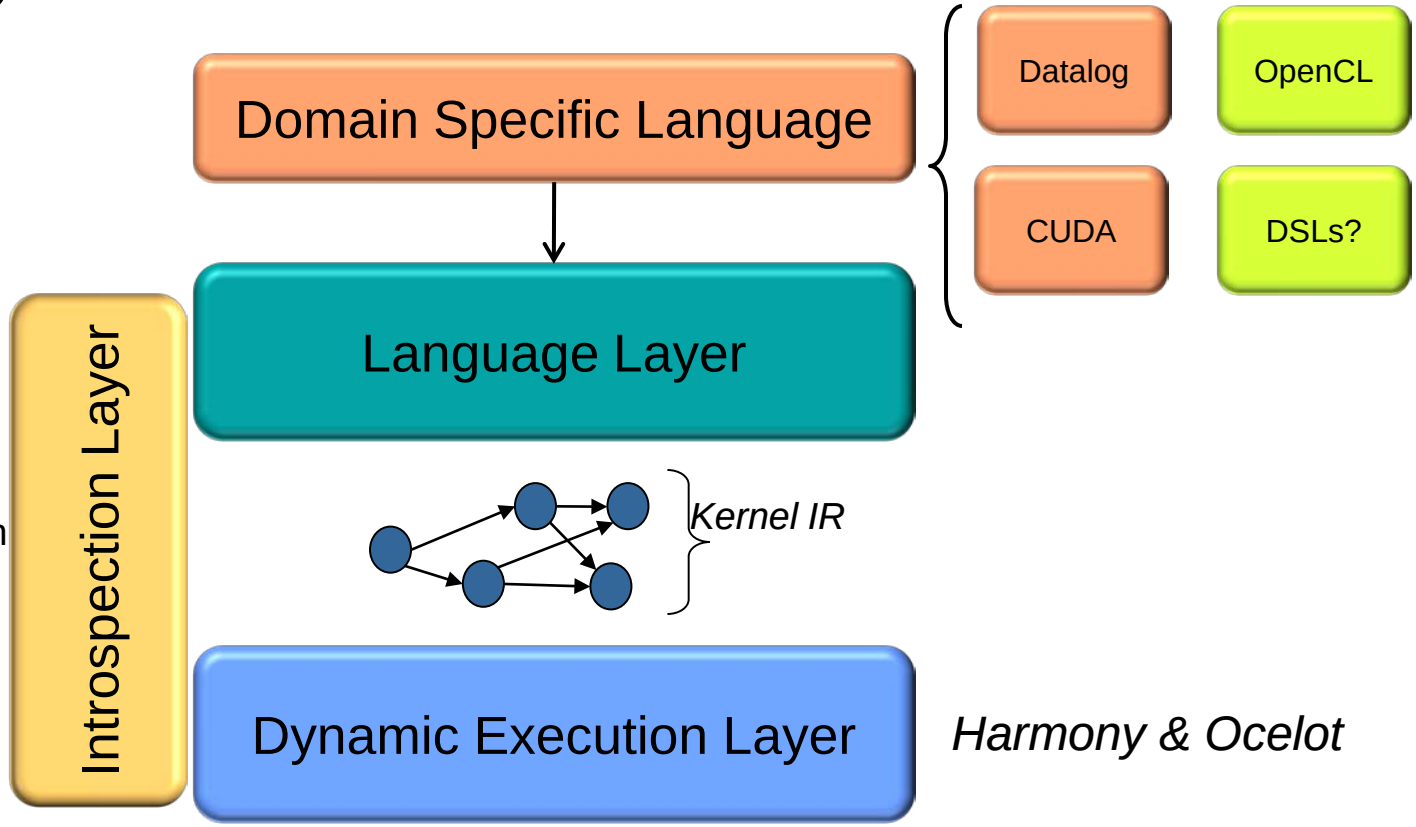
- Intel SandyBridge (i7-2600), SSE 4.2, Ubuntu 11.04 x86-64, 8 hardware threads
- Ocelot 2.0.1464 linked with LLVM 3.0.

System Impact



- Scope of optimization is now enhanced → kernels can execute anywhere
- Multi-ISA problem has been translated into a scheduling and resource management problem

Summary: Dynamic Execution Environments

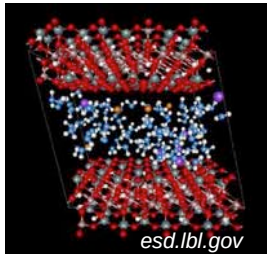


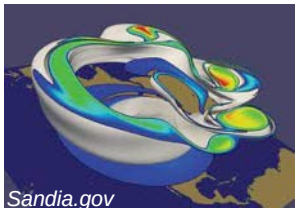
Productivity Tools
- Correctness & Debugging
- Performance Tuning
- Workload Characterization
- Instrumentation

- Core dynamic compiler and run-time system
- Standardized IR for compilation from domain specific languages
- Dynamic translation as a key technology

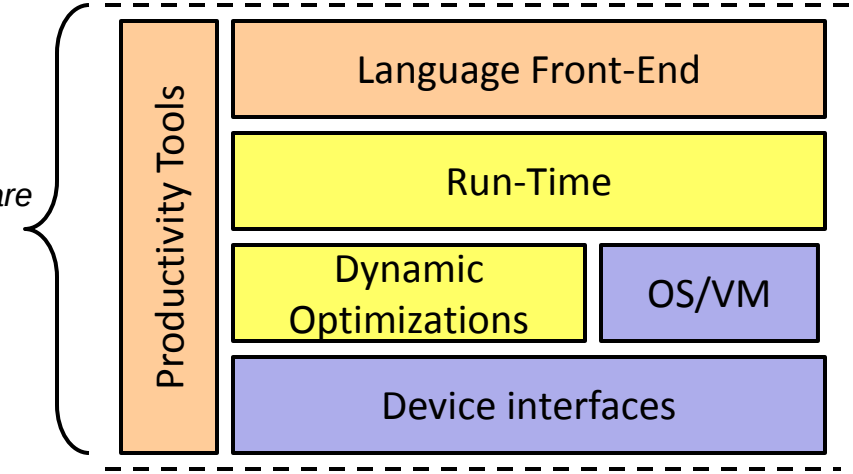
System Software Challenges of Heterogeneity

- Execution Portability
  - –Systems evolve over time
  - –New systems
- Performance Optimization
- Introspection
  - Productivity tools
- Application Migration
  - –Protect investments in existing code bases





Emerging Software Stacks



Outline

- Drivers and Evolution to Heterogeneous Computing
- The Ocelot Dynamic Execution Environment
- Dynamic Translation for Execution Models
- **Dynamic Instrumentation of Kernels**
- Related Projects

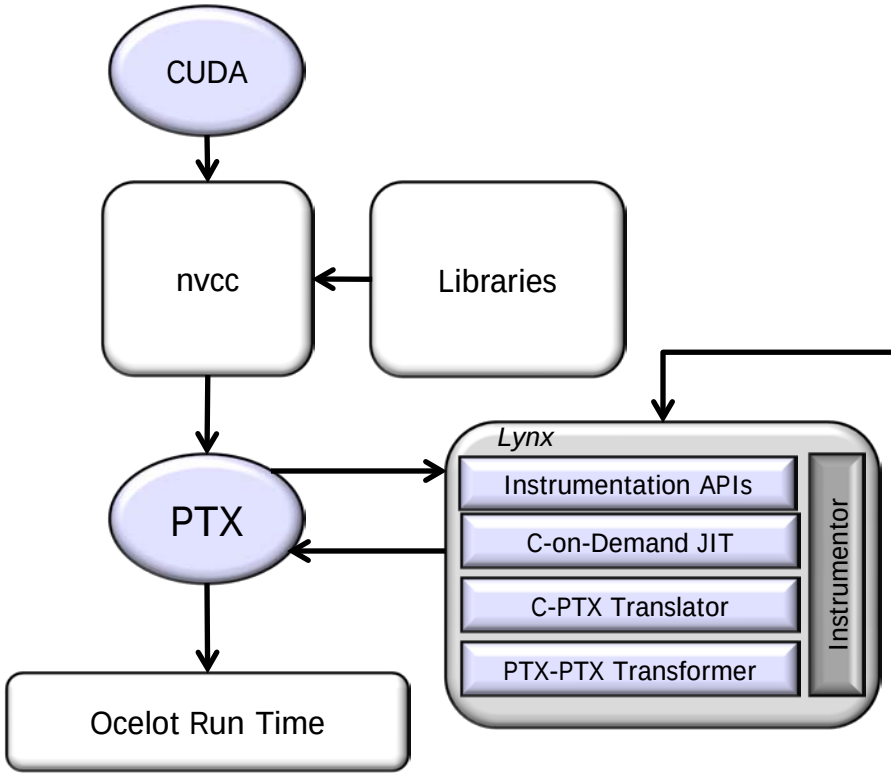
Dynamic Instrumentation as a Research Vehicle

- Run-time generation of user-defined, custom instrumentation code for CUDA kernels

Goals of dynamic binary instrumentation

- Performance Tuning
  - Observe details of program execution much faster than simulation
- Correctness & Debugging
  - Insert correctness checks and assertions
- Dynamic Optimization
  - Feedback-directed optimization and scheduling

Lynx: Software Architecture



Example Instrumentation Code

Memory Efficiency

```
unsigned long threadId = blockThreadId();
unsigned long warpId = (blockId() * blockDim()
  + threadId) >> 5;

ON_INSTRUCTION:   <- insert instrumentation on every instruction
MEM_READ:
MEM_WRITE:        <- only apply instrumentation to
GLOBAL:              global memory instructions
{
   sharedMem[threadId] = computeBaseAddress();

   if(leastActiveThreadInWarp())
   {
      globalMem[warpId * 2] +=
         uniqueElementCount(sharedMem, true);
      globalMem[warpId * 2 + 1] += 1;
   }
}
```

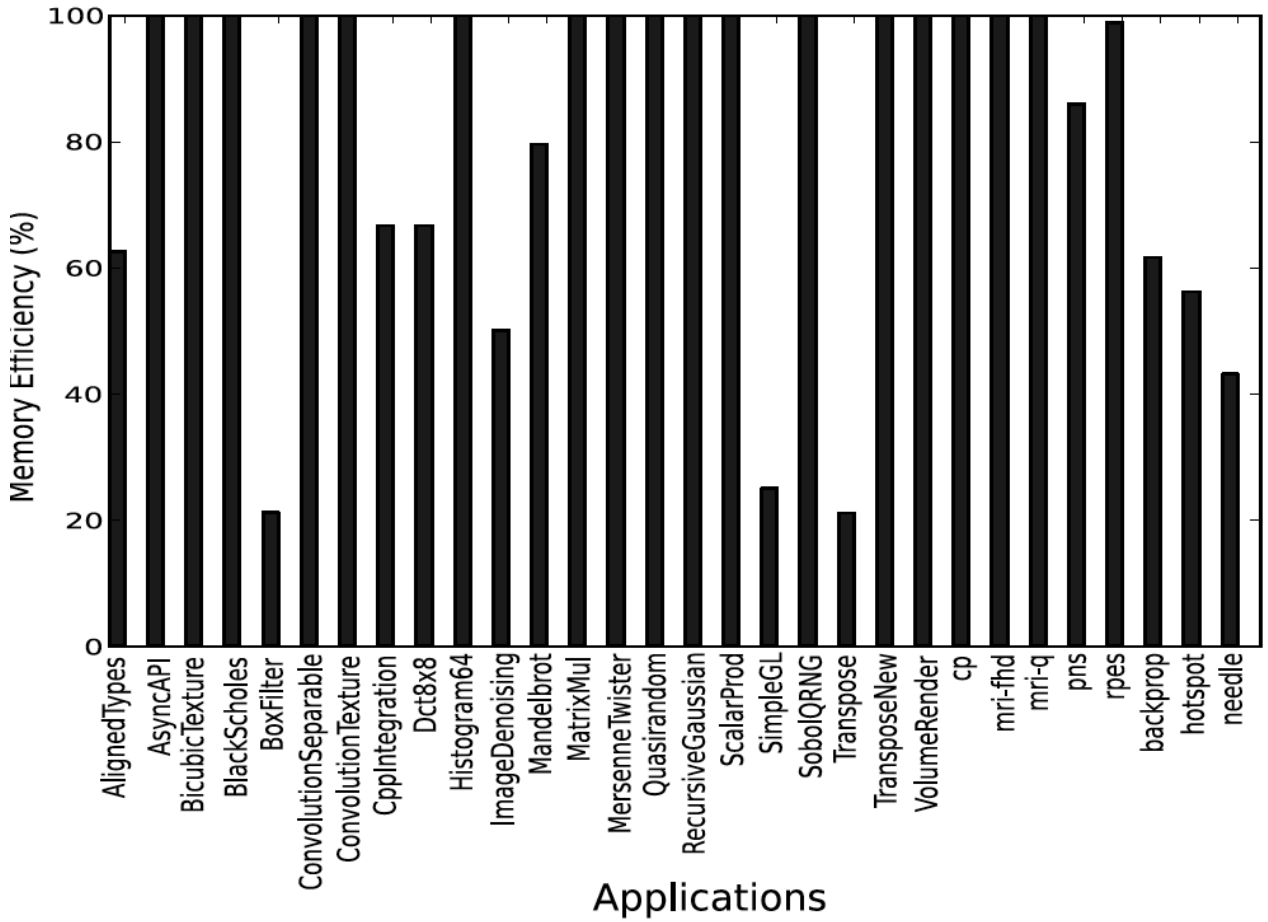
- Inspired by PIN
- Transparent instrumentation of CUDA applications
- Drive Auto-tuners and Resource Managers

N. Farooqui, A. Kerr, G. Eisenhauer, K. Schwan and S. Yalamanchili, “Lynx: Dynamic Instrumentation System for Data-Parallel Applications on GPGPU-based Architectures,” *ISPASS,* April 2012.

Lynx: Features

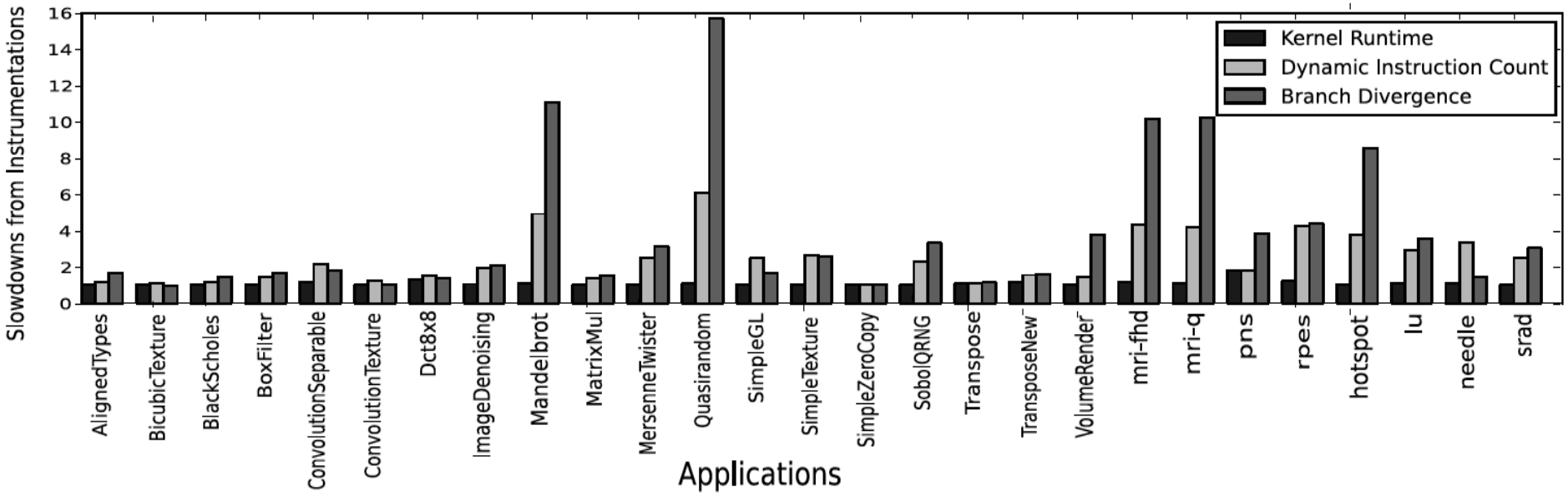
- Enables creation of instrumentation routines that are
  - Selective – instrument only what is needed
  - Transparent – without changes to source code
  - Customizable – user-defined
  - Efficient – using JIT compilation/translation
- Implemented as a transformation pass in Ocelot

Example: Computing Memory Efficiency



Memory Efficiency = (#Dynamic Warps/#Memory_Transactions

Lynx: Overheads



- Overheads are proportional to control flow activity in the kernels

Comparison of Lynx with Some Existing GPU Profiling Tools

FEATURES	*Compute Profiler/CUPTI*	*GPU Ocelot Emulator*	*Lynx*
Transparency	✓	✓	✓
Support for Selective Online Profiling	✗	✓	✓
Customization	✗	✓	✓
Ability to Attach/Detach Profiling at Run-Time	✗	✓	✓
Support for Comprehensive Profiling	✗	✓	✓
Support for Simultaneous Profiling of Multiple Metrics	✗	✓	✓
Native Device Execution	✓	✗	✓

Applications of Lynx

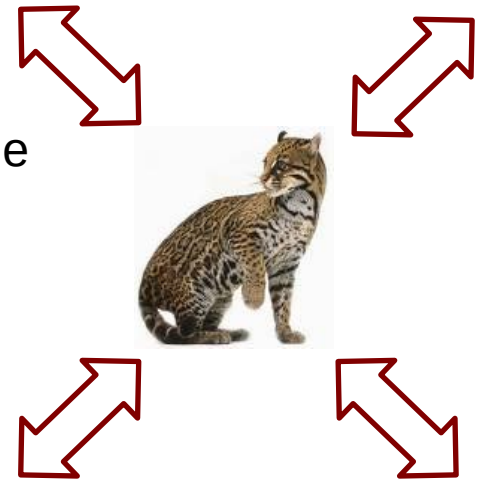
- Transparent modification of functionality
  - Reliable execution
- Correctness tools
  - Debugging support
  - Correctness checks
- Workload characterization
  - Trace analyzers

Applications of Ocelot

Dynamic Compilation

- Red Fox: Compiler for Accelerator Clouds
- DSL-Driven HPC Compiler
- OpenCL Compiler & Runtime

  (joint with H. Kim)



Harmony Run-Time

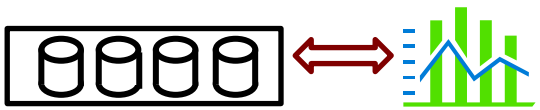
- Mapping & scheduling
- Optimizations: speculation, dependency tracking, etc.

Productivity Tools

- PTX 2.3 emulator
- Correctness and debugging tools
- Trace Generation & Profiling tools
- Dynamic Instrumentation (ala PIN for GPUs)

Eiger:

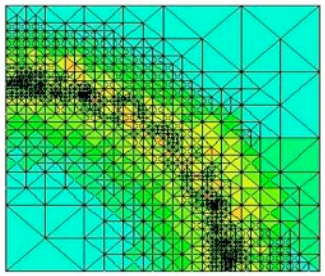
- Workload Characterization and Analysis
- Synthesis of models



Done

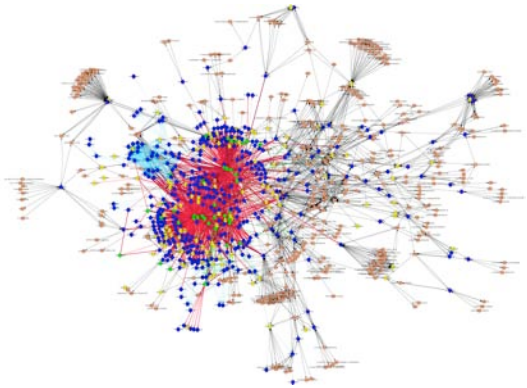
Application: Data Warehousing

Multi-resolution



Database and Data Warehousing

Large Graphs

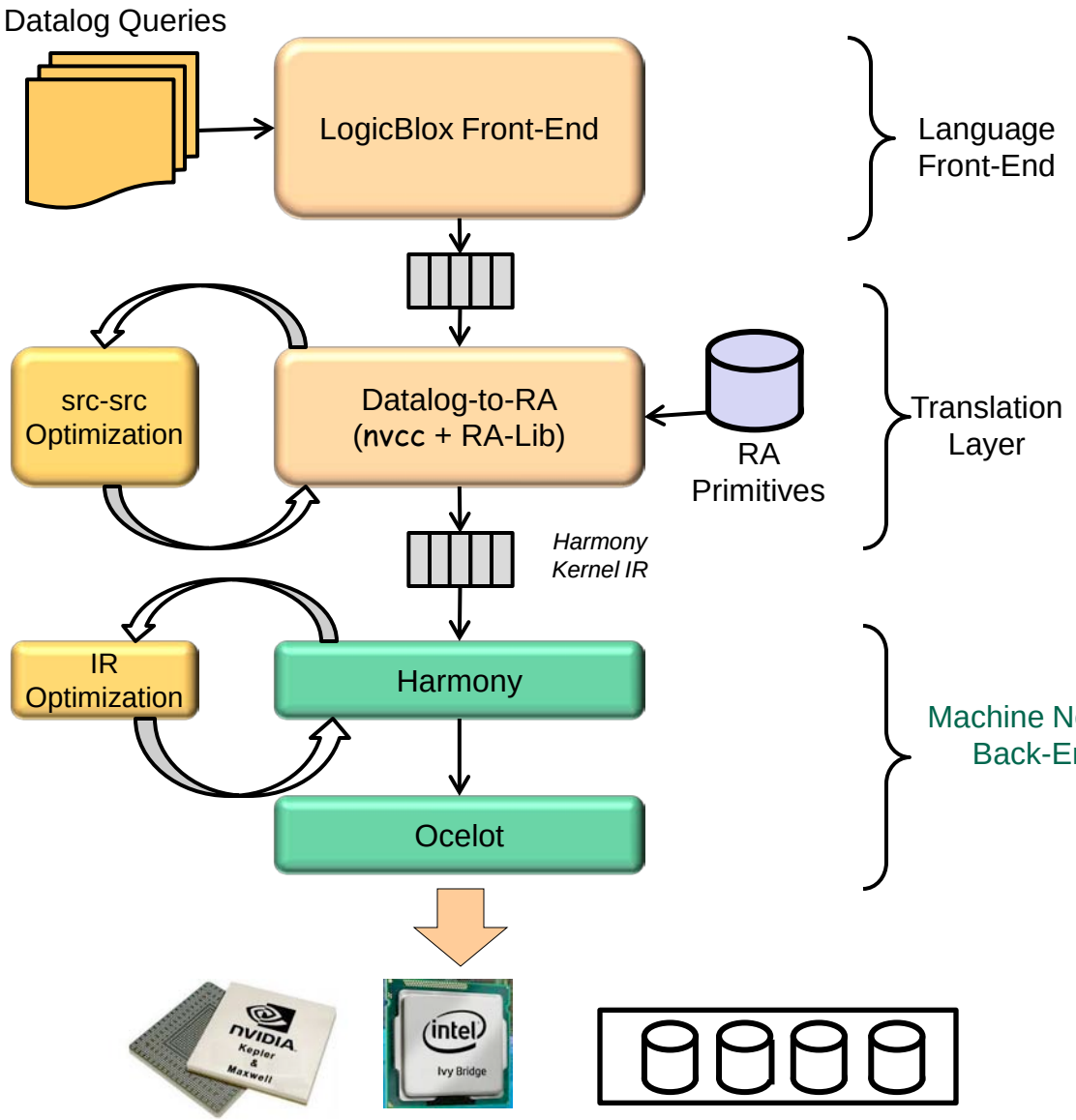


- Massive data sets
- On-line and off-line analysis
  - Retail analysis
  - Forecasting
  - Pricing
  - .....
- Combination of data queries and computational kernels
- Potential to change a companies business model!

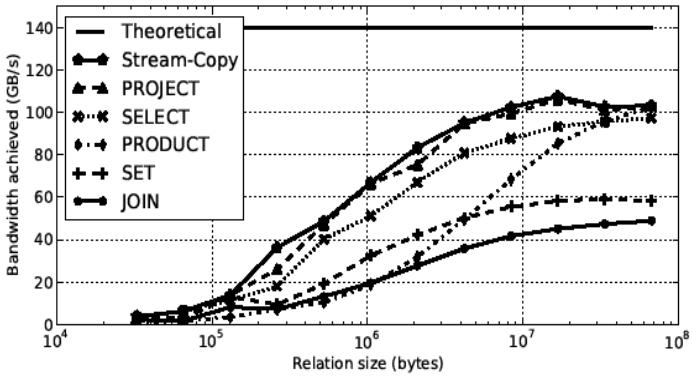
Images from math.nist.gov, blog.thefuturescompany.com,melihsozdinler.blogspot.com

Domain Specific Compilation: Red Fox

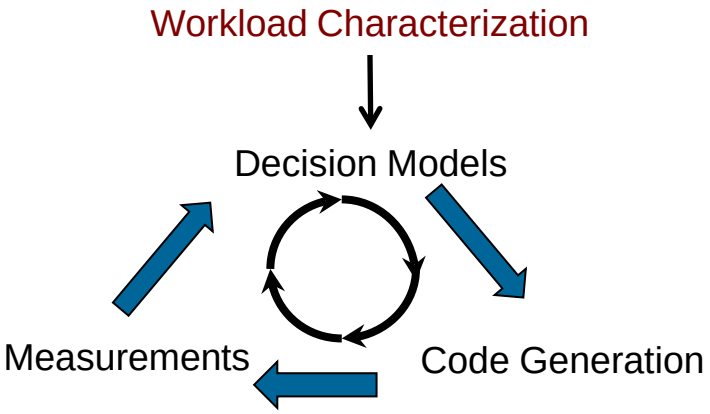
Joint with LogicBlox Inc.

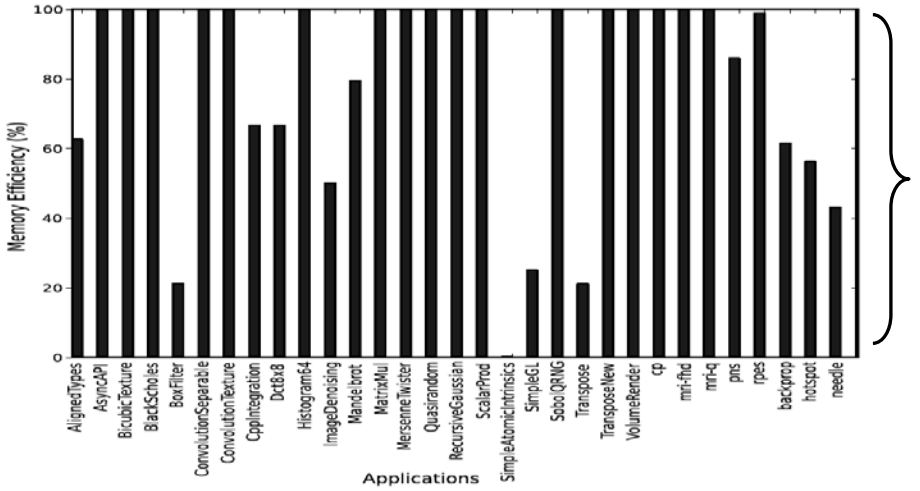


Targeting Accelerator Clouds for meeting the demands of data warehousing applications



Feedback-Driven Optimization: Autotuning

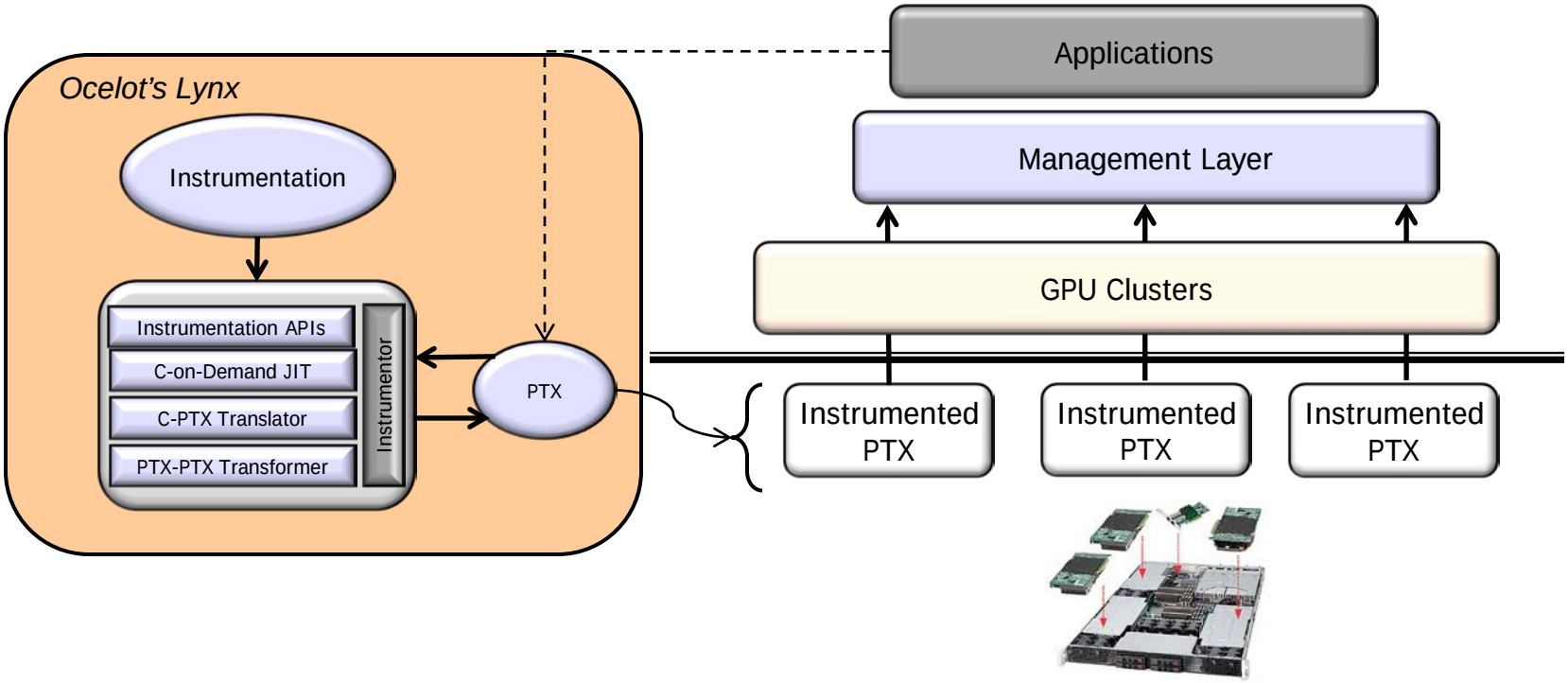




Not available with CUPTI

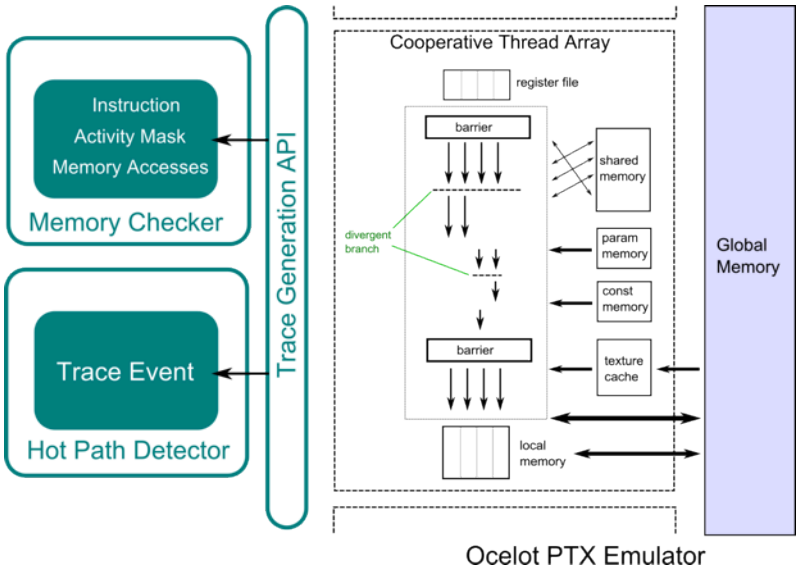
- Use Ocelot's dynamic instrumentation capability
- Real-Time feedback drives the Ocelot kernel JIT
- Decision models to drive existing/new auto-tuners
  - Change data layout to improve memory efficiency
  - Use different algorithms
  - Selective invocation → hot path profiling → algorithm selection

Feedback-Driven Resource Management

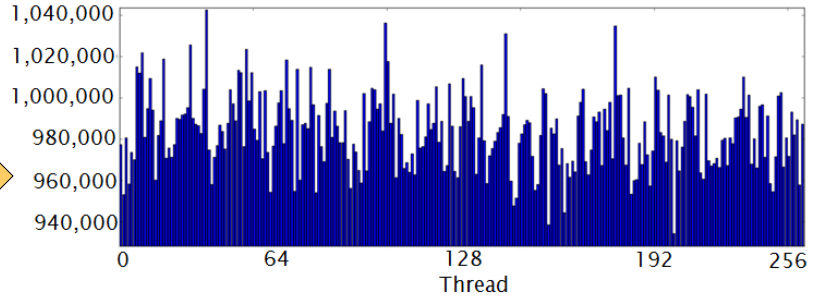


- Real time customized information available about GPU usage
- Can drive scheduling decisions
- Can drive management policies, e.g., power, throughput, etc.

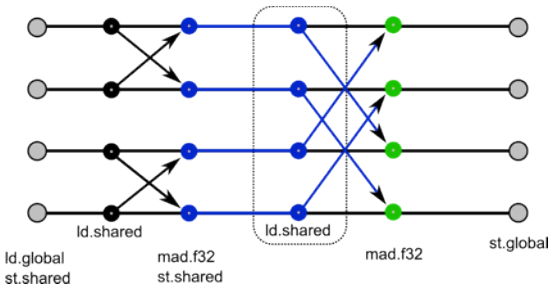
Workload Characterization and Analysis



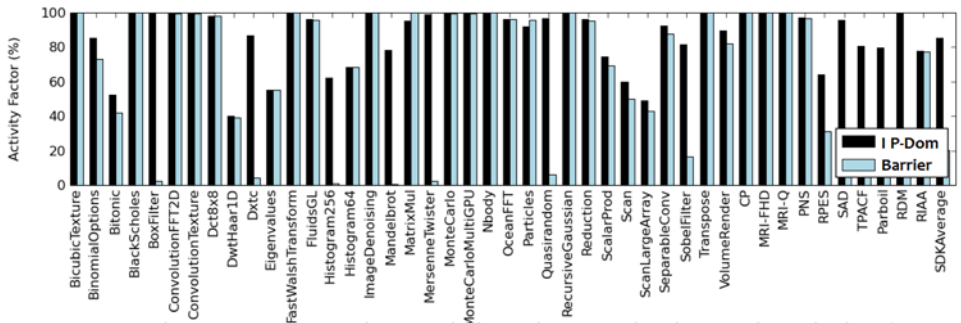
SM Load Imbalance (Mandelbrot)

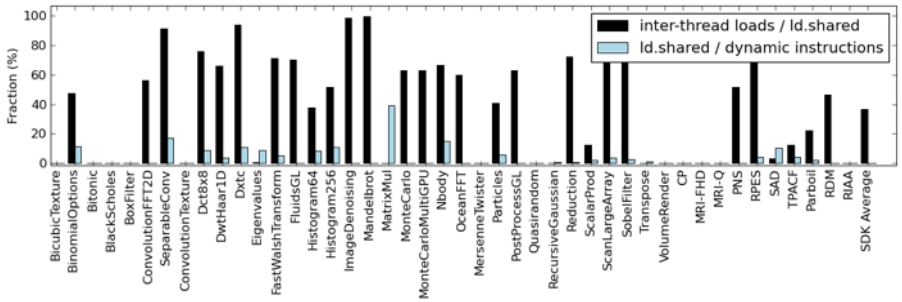


Intra-Thread Data Sharing



Activity Factor



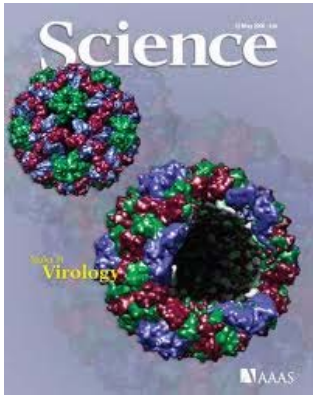


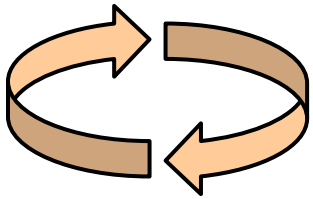
A. Kerr, G. Diamos, and S. Yalamanchili, A characterization and analysis of PTX kernels," *IEEE International Symposium on Workload Characterization,* Austin, TX, USA,

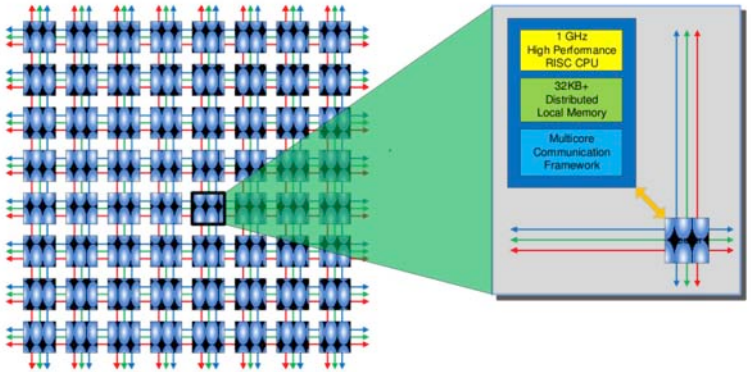
October 200

Constructing Performance Models: Eiger

- Develop a portable methodology to discover relationships between architectures and applications





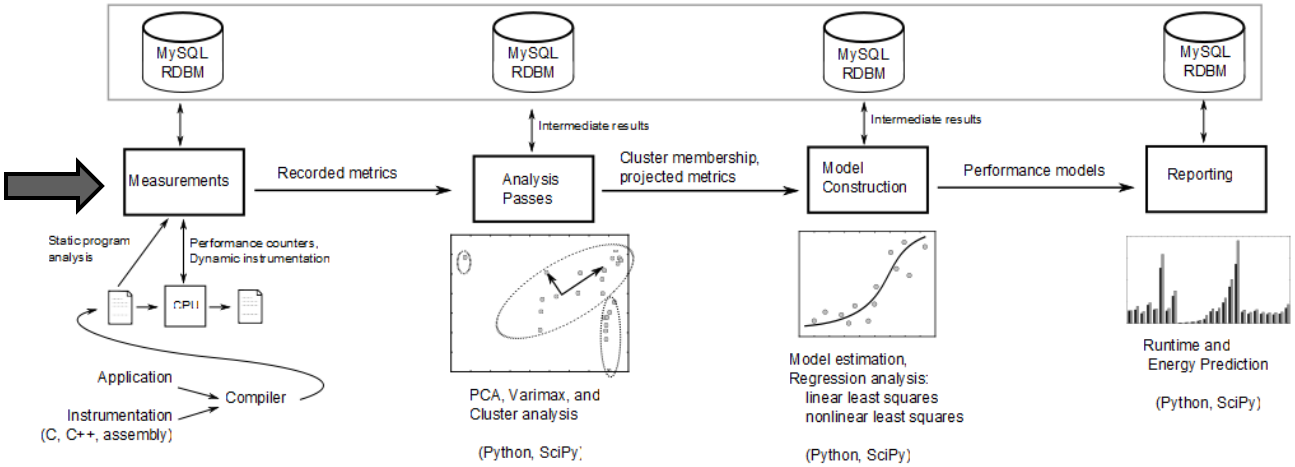


Adapteva's multicore from electronicdesign.com

- Extensions to Ocelot for the synthesis of performance models
  - Used in macroscale simulation models
  - Used in JIT compilers to make optimization decisions
  - Used in run-times to make scheduling decisions

Eiger Methodology

Ocelot JIT



SST/Macro

Sandia National Laboratories

- Use data analysis techniques to uncover application-architecture relationships
  - Discover and synthesize analytic models
- Extensible in source data, analysis passes, model construction techniques, and destination/use

Ocelot Team, Sponsors and Collaborators

- Ocelot Team
  - Gregory Diamos, Rodrigo Dominguez (NEU), Naila Farooqui, Andrew Kerr, Ashwin Lele, Si Li, Tri Pho, Jin Wang, Haicheng Wu, Sudhakar Yalamanchili & several open source contributors

IBM Research

NVIDIA

AMD

NSF

LOGICBLOX

Thank You

Questions?