How To Run Your CUDA Program Anywhere

– A Perspective from Virginia Tech's GPU-Accelerated HokieSpeed Cluster

Wu Feng

Dept. of Computer Science
Dept. of Electrical & Computer Engg.
Virginia Bioinformatics Institute
Virginia Tech

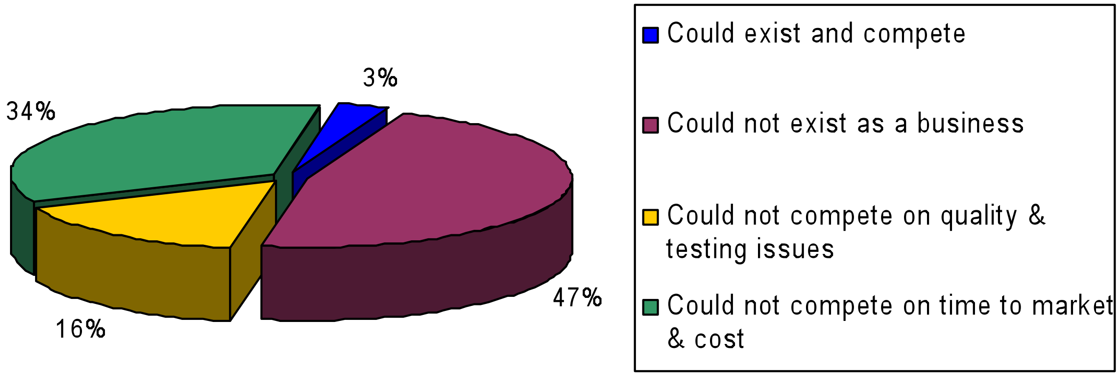
Dept. of Cancer Biology and the
Translational Science Institute
School of Medicine
Wake Forest University

VirginiaTech
Invent the Future

SyNeRG
synergy.cs.vt.edu

Why High-Performance Computing (HPC)?

Competitive Risk From *Not* Having Access to HPC



Data from Council of Competitiveness. Sponsored Survey Conducted by IDC

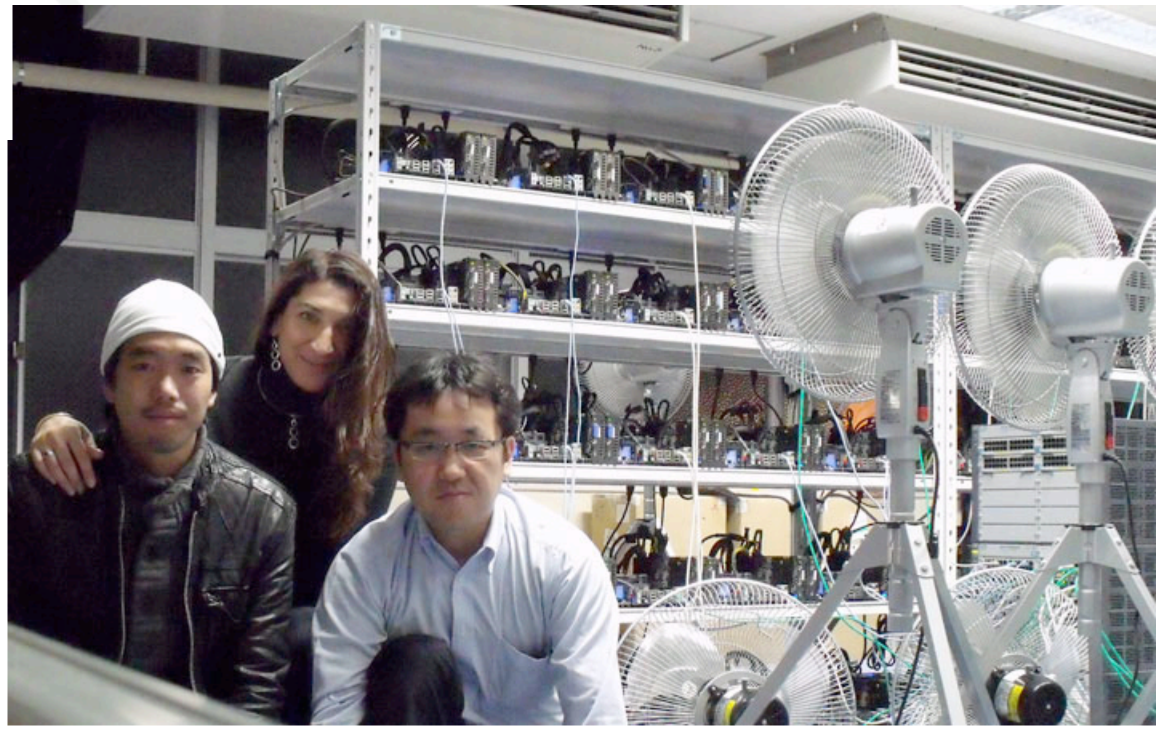
- Only 3% of companies could exist and compete without HPC.
  – 200+ participating companies, including many Fortune 500 companies (Proctor & Gamble and biological and chemical companies)

Why GPU-Accelerated HPC?

The Second Coming of the "Beowulf Cluster" for HPC

- The First Coming of the "Beowulf Cluster" (Early 1990s)
  - Aggregated commodity PC technology to create what was dubbed a Beowulf cluster, delivering supercomputing to the masses.
- The Second Coming of the "Beowulf Cluster" (Early 2010s)
  - Leveraging commodity PC technology again, but …
    … this time in the form of commodity GPUs …

"Beowulf 1.0 → Beowulf 2.0"

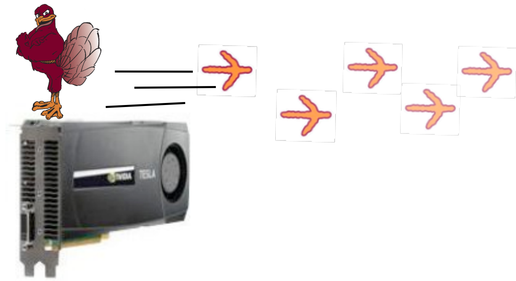


Green Supercomputers (Nov. 2010)

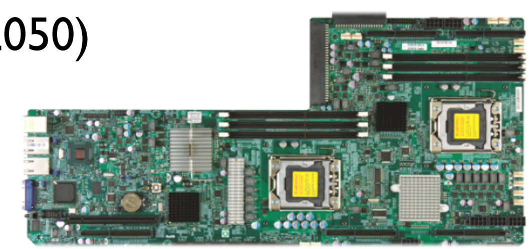
Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	1684.20	IBM Thomas J. Watson Research Center	NNSA/SC Blue Gene/Q Prototype	38.80
2	1448.03	National Astronomical Observatory of Japan	GRAPE-DR accelerator Cluster, Infiniband	24.59
3	958.35	GSIC Center, Tokyo Institute of Technology	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows	1243.80
4	933.06	NCSA	Hybrid Cluster Core i3 2.93Ghz Dual Core, NVIDIA C2050, Infiniband	36.00
5	828.67	RIKEN Advanced Institute for Computational Science	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect	57.96
6	773.38	Universitaet Wuppertal	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	57.54
6	773.38	Universitaet Regensburg	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	57.54
6	773.38	Forschungszentrum Juelich (FZJ)	QPACE SFB TR Cluster, PowerXCell 8i, 3.2 GHz, 3D-Torus	57.54
9	740.78	Universitaet Frankfurt	Supermicro Cluster, QC Opteron 2.1 GHz, ATI Radeon GPU, Infiniband	385.00
10	677.12	Georgia Institute of Technology	HP ProLiant SL390s G7 Xeon 6C X5660 2.8Ghz, nVidia Fermi, Infiniband QDR	94.40
11	636.36	National Institute for Environmental Studies	GOSAT Research Computation Facility, nvidia	117.15

HokieSpeed

A GPU-Accelerated Supercomputer for the Masses



- Purpose
  - To catalyze new approaches for conducting research via the synergistic amalgamation of heterogeneous CPU-GPU hardware and software
- Profile
  - Total Nodes: 209, where each compute node consists of
    - Motherboard: Supermicro 2026GT-TRF Dual Intel Xeon
    - CPUs: Two 2.4-GHz Intel Xeon E5645 6-core (12 CPU cores per node)
    - GPUs: Two NVIDIA Tesla Fermi GPUs (M2050/C2050)



VirginiaTech
Invent the Future

HokieSpeed: Performance at a Glance

- An Instantiation of "The Second Coming of Beowulf Clusters"
- A Commodity GPU-Accelerated Supercomputer
  - ***Speed***
    - Single-Precision Peak: 430.7 Tflops (Double-Precision Peak: 215.4 Tflops)
    - TOP500 Linpack: 120.4 Tflops
  - ***Greenness***
    - CPU TDP: 85 W. GPU TDP: 225 W (M2050) and 238 W (C2050).
    - **Green500 Rating: 928.96 MFLOPS/W**
      - Green LINPACK: 117.3 Tflops
      - Total Power: 126.27 kW (including the network and disks)

HokieSpeed Team

- P. Balaji, CS @ ANL and U. Chicago
- S. Beardsley, Bio @ Marquette U.
- K. Bisset, CS & Social Sciences @ VT
- A. Butt, CS & Green Computing @ VT
- Y. Cao, CS & Social Sciences, VT
- N. Cellinese, Bio @ U. Florida
- I. Chen, Data Mining & Social Sciences @ VT
- T. D. Crawford, Chemistry @ VT
- Eric de Sturler, Mathematics @ VT
- W. Feng (PI), CS, Green Computing, Bio, and Data Mining @ VT
- X. Feng, CS & Bio @ Marquette U.
- M. Gardner, Office of IT @ VT
- R. Ge, CS & Green Computing @ Marquette U.
- K. Hilu, Bio @ VT
- D. Hong, Mechanical Engg. @ VT
- C. Hsu, CS & Green Computing @ ORNL
- S. King, Geosciences @ VT
- H. Lin, CS & Bio @ VT
- C. Lu, CS, Data Mining, & Social Sciences @ VT
- M. Marathe, Bio & Social Sciences @ VT
- P. McCormick, CS @ LANL
- D. Nikolopoulos, CS & Green Computing @ ICS-Forth
- A. Onufriev, Biochemistry & CS @ VT
- N. Ramakrishnan, Data Mining @ VT
- A. Sandu, Chemistry & CS @ VT
- J. Setubal, Bio @ VT
- C. Struble, Bio @ Marquette U.
- D. Tafti, Mechanical Engg. @ VT
- E. Tilevich, CS @ VT
- C. Weiss, Geosciences @ VT
- L. Zhang, CS & Bio @ VT
- P. Zhang, Bio @ VT
- Y. Zhou, Geosciences @ VT

:Seneca SUPERMICRO NVIDIA

Forecast

- Motivation & Background
- Programming GPGPUs
- How To Run Your CUDA Program Anywhere
  - CU2CL: A CUDA-to-OpenCL Source-to-Source Translator
- Evaluation
  - Coverage, Translation Time, and Performance
- Future Work
- Summary

Programming GPGPUs

CUDA

- NVIDIA's framework that popularized GPGPU programming

OpenCL

- An open standard (Khronos Group) that provides a vendor-neutral environment to program GPUs, CPUs, FPGAs, etc.

Challenges for Domain Scientists & Engineers

- *Novices:* Writing from Scratch → Learning Curve
  - OpenCL too low level an API compared to CUDA
    … easier to start with CUDA
- *Experts:* Leveraging CUDA Investment → Multithreaded CPU, OpenCL, OpenMP
- Porting from CUDA to OpenCL
  - Tedious and error-prone

Initialization Code: CUDA

None!

Initialization Code: OpenCL

OpenCL

```
clGetPlatformIDs(1, &clPlatform, NULL);
clGetDeviceIDs(clPlatform, CL_DEVICE_TYPE_GPU, 1, &clDevice, NULL);
clContext = clCreateContext(NULL, 1, &clDevice, NULL, NULL, &errcode);
clCommands = clCreateCommandQueue(clContext, clDevice, 0, &errcode);

kernelFile = fopen("srad_kernel.cl", "r");
fseek(kernelFile, 0, SEEK_END);
kernelLength = (size_t) ftell(kernelFile);
kernelSource = (char *) malloc(sizeof(char)*kernelLength);
rewind(kernelFile);
fread((void *) kernelSource, kernelLength, 1, kernelFile);
fclose(kernelFile);

clProgram = clCreateProgramWithSource(clContext, 1, (const char **)
    &kernelSource, &kernelLength, &errcode);
free(kernelSource);
clBuildProgram(clProgram, 1, &clDevice, NULL, NULL, NULL);

clKernel_srad1 = clCreateKernel(clProgram, "srad_cuda_1", &errcode);
clKernel_srad2 = clCreateKernel(clProgram, "srad_cuda_2", &errcode);
```

VirginiaTech Invent the Future

SyNeRGy
synergy.cs.vt.edu

Executing Device Code: CUDA

```
dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);

for (int i=0; i < matrix_dim-BLOCK_SIZE; i += BLOCK_SIZE)
{
    lud_diagonal<<<1, BLOCK_SIZE>>>(m, matrix_dim, i);
    lud_perimeter<<<(matrix_dim-i)/BLOCK_SIZE-1, BLOCK_SIZE*2>>>(m, matrix_dim, i);
    dim3 dimGrid((matrix_dim-i)/BLOCK_SIZE-1, (matrix_dim-i)/BLOCK_SIZE-1);
    lud_internal<<<dimGrid, dimBlock>>>(m, matrix_dim, i);
}
lud_diagonal<<<1,BLOCK_SIZE>>>(m, matrix_dim, i);
```

Executing Device Code: OpenCL

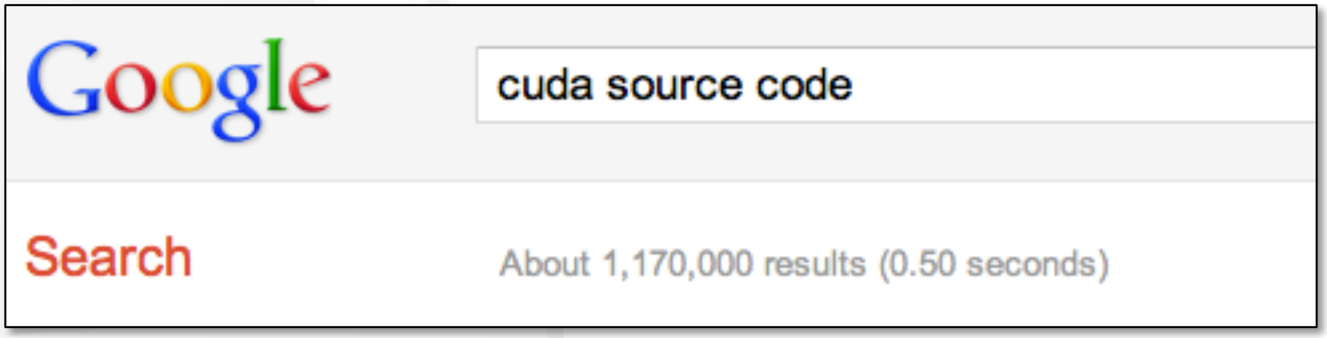
```
size_t localWorkSize[2];
size_t globalWorkSize[2];

for (int i=0; i < matrix_dim-BLOCK_SIZE; i += BLOCK_SIZE)
{
    clSetKernelArg(clKernel_diagonal, 0, sizeof(cl_mem), (void *) &d_m);
    clSetKernelArg(clKernel_diagonal, 1, sizeof(int), (void *) &matrix_dim);
    clSetKernelArg(clKernel_diagonal, 2, sizeof(int), (void *) &i);
    localWorkSize[0] = BLOCK_SIZE;
    globalWorkSize[0] = BLOCK_SIZE;
    clEnqueueNDRangeKernel(clCommands, clKernel_diagonal, 1, NULL, globalWorkSize, localWorkSize,
0, NULL, NULL);
... (14 more lines)
}
clSetKernelArg(clKernel_diagonal, 0, sizeof(cl_mem), (void *) &d_m);
clSetKernelArg(clKernel_diagonal, 1, sizeof(int), (void *) &matrix_dim);
clSetKernelArg(clKernel_diagonal, 2, sizeof(int), (void *) &i);
localWorkSize[0] = BLOCK_SIZE;
globalWorkSize[0] = BLOCK_SIZE;
clEnqueueNDRangeKernel(clCommands, clKernel_diagonal, 1, NULL, globalWorkSize, localWorkSize, 0,
NULL, NULL);
```

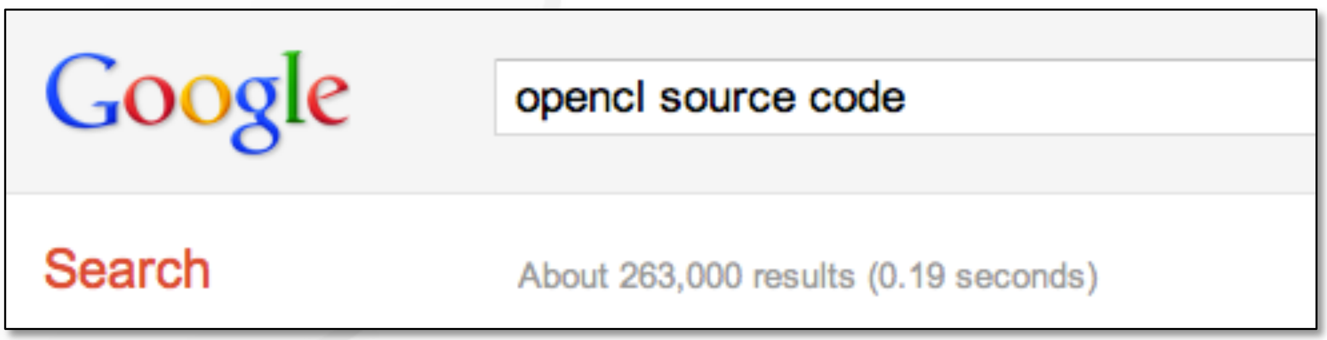
Challenges for Domain Scientists & Engineers

- *Novices:* Writing from Scratch → Learning Curve
  - OpenCL too low level an API compared to CUDA
    … arguably easier to start with CUDA
- ***Experts:* Leveraging CUDA Investment**

Prevalence



First public release February 2007



First public release December 2008

Search Date: 2011-09-14

CUDA-Accelerated Applications

See: http://www.nvidia.com/object/cuda_app_tesla.html

GOVERNMENT & DEFENSE
Ikena: Imagery Analysis and Video Forensics
Signal Processing Library: GPU VSIPL
IDL and MATLAB® Acceleration: GPULib
GIS: Manifold

MOLECULAR DYNAMICS, COMPUTATIONAL CHEMISTRY
OpenMM library for molecular dynamics on GPUs
GROMACS using OpenMM
NAMD molecular dynamics
VMD visualization of molecular dynamics
HOOMD molecular dynamics
Acellera: ACEMD bio-molecular dynamics package
BigDFT: DFT (Density functional theory) electronic structure
MDGPU
GPUGrid.net

LIFE SCIENCES, BIO-INFORMATICS
GPU HMMER
DNA Sequence alignment: MUMmerGPU
LISSOM: model of human neocortex using CUDA
Silicon Informatics: AutoDock

ELECTRODYNAMICS AND ELECTROMAGNETIC
Acceleware: FDTD Solver
Acceleware: EM Solutions
Remcom XStream FDTD
SPEAG Semcad X
CST Microwave Studio
Quantum electrodynamics library
GPMAD : Particle beam dynamics simulator

MEDICAL IMAGING, CT, MRI
RealityServer
GPULib:IDL acceleration
Acceleware: Imaging Solutions
Digisens: SnapCT tomographic reconstruction software
Techniscan: Whole Breast Ultrasound Imaging System

OIL & GAS
Acceleware: Kirchoff and Reverse Time Migration
SeismicCity: 3D seismic imaging for prestack depth migration
OpenGeoSolutions: Spectral decomposition and inversion
Mercury Computer systems: 3D data visualization
ffA: 3D Seismic processing software
Headwave: Prestack data processing

FINANCIAL COMPUTING AND OPTIONS PRICING
SciComp: derivatives pricing
Hanweck: options pricing
Exegy: Risk Analysis
Aqumin: 3D Visualization of market data
Level 3 Finance
OnEye (Australia): Accelerated Trading Solutions
Arbitragis Trading

MATLAB, LABVIEW, MATHEMATICA, R
CUDA Acceleration for MATLAB
Accelereyes: Jacket™ engine for MATLAB
GPULib: mathematical functions for IDL and MATLAB
Integrating Simulink with CUDA using S-functions
Enabling GPU Computing in the R Statistical Environment
Mathematica plug-in for CUDA
National Instruments LabView for NVIDIA GPUs

ELECTRONIC DESIGN AUTOMATION
Agilent EESof: ADS SPICE simulator
Synopsys: Sentaraus TCAD
Gauda: Optical proximity correction (OPC)

WEATHER AND OCEAN MODELING
CUDA-accelerated WRF code

VIDEO, IMAGING, AND VISION APPLICATIONS
Axxon Intellect Enterprise Video Surveillance Software
Pflow CUDA Plugin for Autodesk 3ds Max
RUINS Shatter CUDA Plug-in for Maya
Bullet 3D Multi-Physics Library with CUDA Support
CUDA Voxel Rendering Engine
Furryball: Direct3D GPU Rendering Plugin for Maya

Examples of Available CUDA Source Code

Source: http://gpgpu.org/

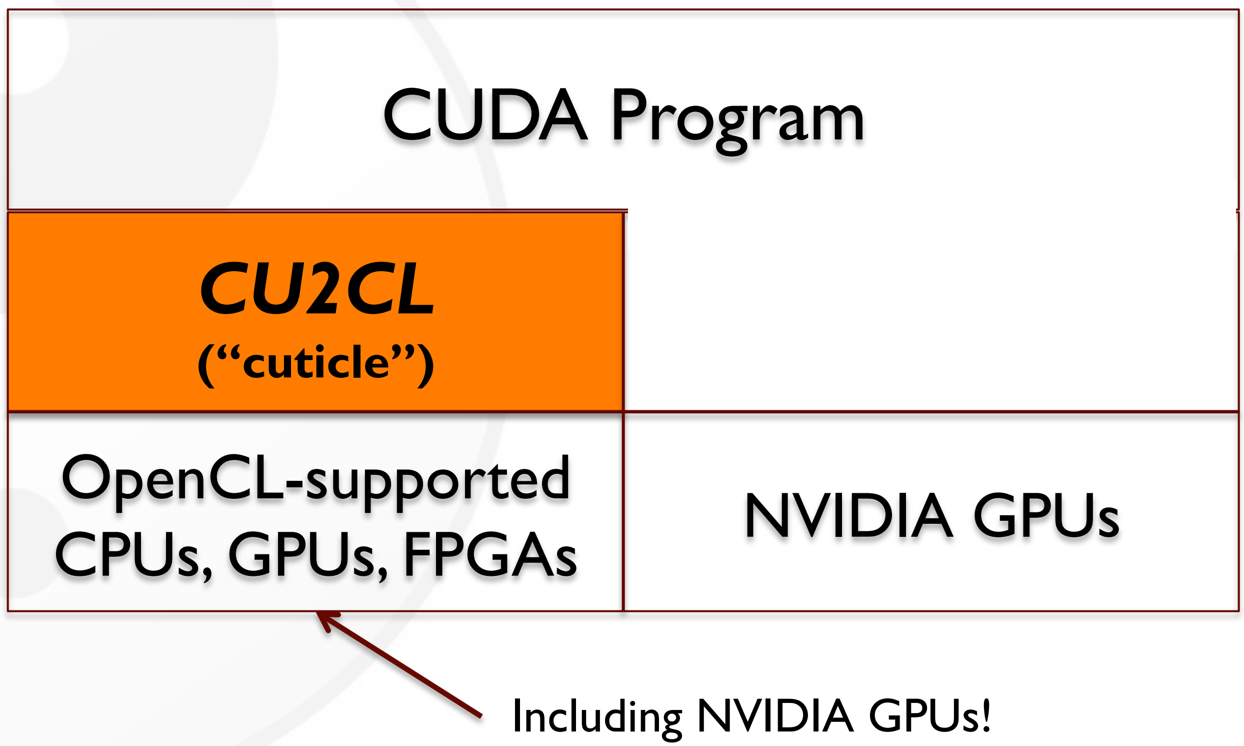
- odeint: ODE solver
- OpenCurrent: PDE solver
- R+GPU: accelerate R
- Alenka: "SQL for CUDA"
- GPIUTMD: multi-particle dynamics
- rCUDA: remote invocation
- HOOMD-blue: particle dynamics
- Exact String Matching for GPU
- GMAC: asymmetric distributed memory
- TRNG: random number generation
- OpenNL: numeric library
- VMD: visual molecular dynamics
- CUDA memtest
- GPU-accelerated Ising model
- Image segmentation via Livewire
- OpenFOAM: accelerated CFD
- PFAC: string matching
- NBSimple: n-body code
- WaveTomography: wave propagation reconstruction
- CUDAEASY: cosmological lattice
- HPMC: volumetric iso-surface extraction
- OpenMM: molecular dynamics
- MUMmerGPU: DNA alignment
- SpMV4GPU: sparse-matrix multiplication toolkit

and more…

Challenges for Domain Scientists & Engineers

- Novices: Writing from Scratch → Learning Curve
  - OpenCL too low level an API compared to CUDA
    … arguably easier to start with CUDA
- Experts: Leveraging CUDA Investment
- Our Solution
  - How To Run Your CUDA Program Anywhere via
    CU2CL: A CUDA-to-OpenCL Source-to-Source Translator *("cuticle")*

How To Run Your CUDA Program Anywhere



Forecast

- Motivation & Background
- Programming GPGPUs
- How To Run Your CUDA Program Anywhere
  - CU2CL: A CUDA-to-OpenCL Source-to-Source Translator
- Evaluation
  - Coverage, Translation Time, and Performance
- Future Work
- Summary

Goals of CU2CL ("cuticle")

- Automatically support many CUDA applications
- Maintainable OpenCL code for future development
- Seek adoption from CUDA and OpenCL communities

Image: http://img3.imagebanana.com/img/30sdnj4/goodman02_PinkyBrain01.jpg

Ecosystem for “CUDA Anywhere”

Source Languages

CUDA | OpenCL | Other

Translator Infrastructure

Language-Dependent Front Ends

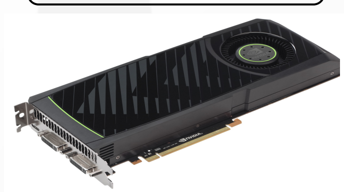
Platform-Dependent De-optimizations

Platform-Independent Optimizations

Platform-Specific Optimizations

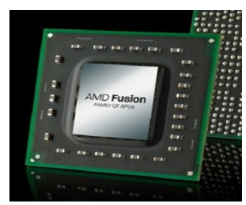
Device-Specific Driver

PTX → NVIDIA GPU

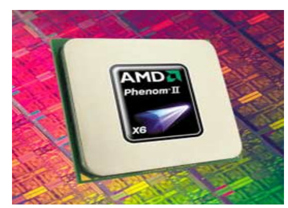


CAL → AMD GPU

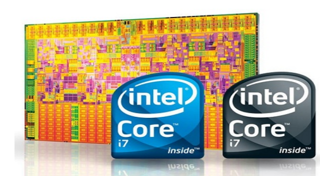
ASM & CAL → AMD APU



ASM → AMD CPU



ASM → Intel CPU



CUDA & OpenCL API

CUDA Module	OpenCL Module
Thread	Contexts & Command Queues
Device	Platforms & Devices
Stream	Command Queues
Event	Events
Memory	Memory Objects

CUDA & OpenCL Data

CUDA	OpenCL
Vector types (e.g. float4)	Host: cl_float4 Kernel: float4
dim3	size_t[3]
cudaStream_t	cl_command_queue
cudaEvent_t	cl_event
Device pointers (e.g. float* created through cudaMalloc)	cl_mem created through clCreateBuffer
cudaChannelFormat	cl_image_format
textureReference	cl_mem created through clCreateImage
cudaDeviceProp	No direct equivalent

CUDA & OpenCL Synchronization Functions

Writing Kernels: Synchronization

C for CUDA terminology	OpenCL terminology
__syncthreads()	**barrier()**
__threadfence()	No direct equivalent.
__threadfence_block()	**mem_fence(CLK_GLOBAL_MEM_FENCE | CLK_LOCAL_MEM_FENCE)**
No direct equivalent.	**read_mem_fence()**
No direct equivalent.	**write_mem_fence()**

CUDA Language Overview

- Divides host (CPU) and device (GPU) code
- Both are extensions of C/C++
- Host-side APIs set up GPUs and launch kernels

Runtime API

- High-level API
- Implicitly performs most set up/tear down
- Assumes single device

Driver API

- Low-level API
- Requires explicit set up/tear down
- Allows multiple devices

OpenCL Language Overview

OpenCL

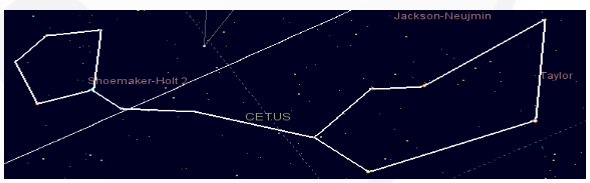
- Divides host (CPU) and device (GPU) code
- Generalized, device-agnostic API
- Support for task- and data-parallel kernels
- Kernels written in separate files
- No interplay between host and device code
  - No extern'd constant or shared memory
- No direct access to device pointers (CUDA 4.0)
  - Prevents device structs with device pointers

Translator Base to Build Upon

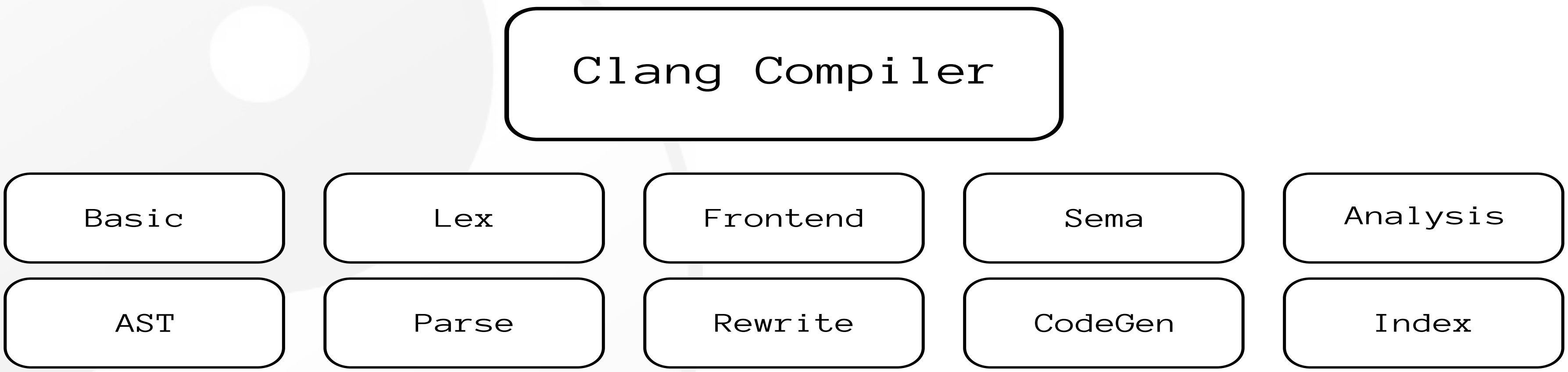
- Production-quality compiler
- Ease of extensibility

Clang

Cetus



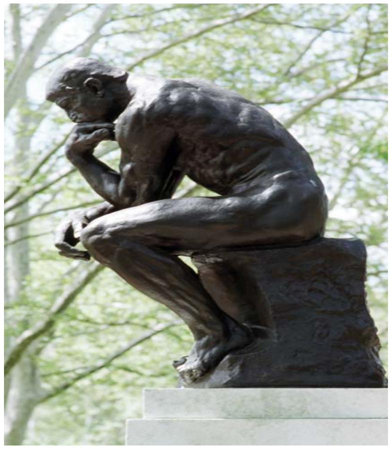
The Clang Compiler Framework



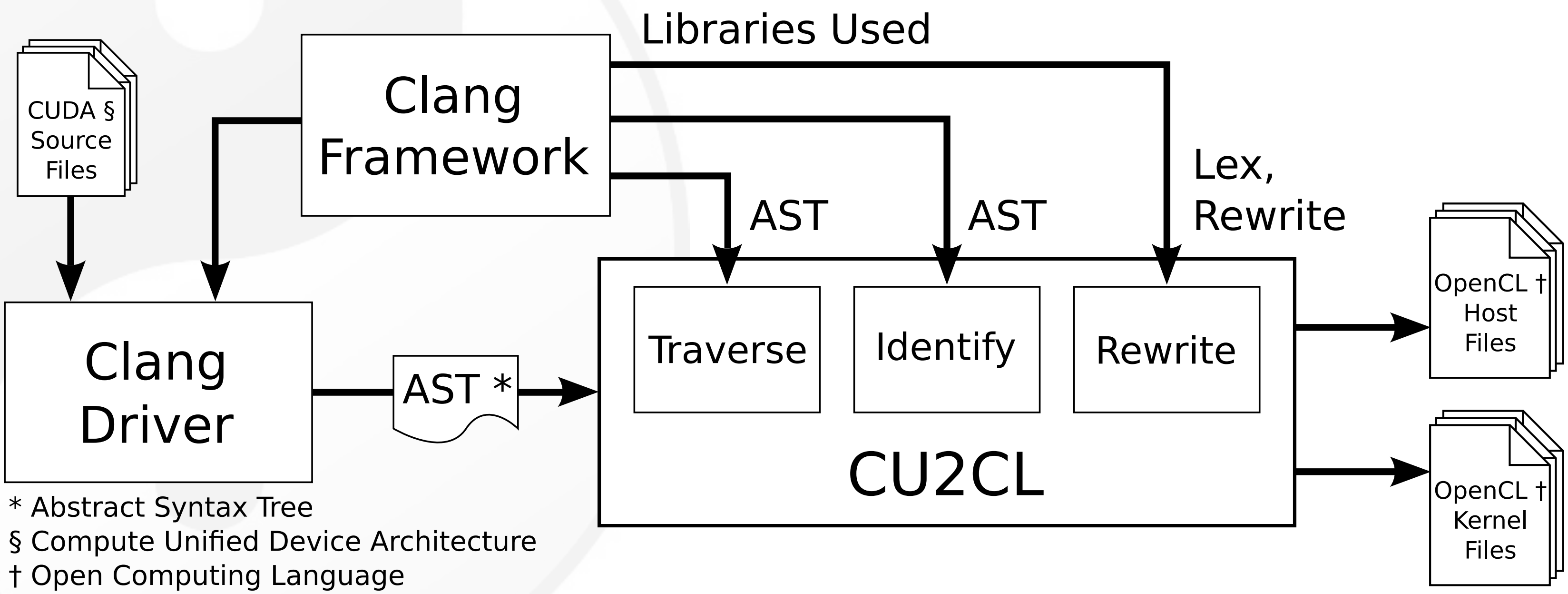
- Useful libraries for C/C++ source-level tools
- Powerful AST representation
- Clang compiler built on top

AST-Driven, String-Based Rewriting

- Characteristics
  - Does not modify the AST
  - Instead, edit text in source ranges
- Benefits
  - Useful for transformations with limited scope
  - Preserves formatting and comments



Architecture of CU2CL

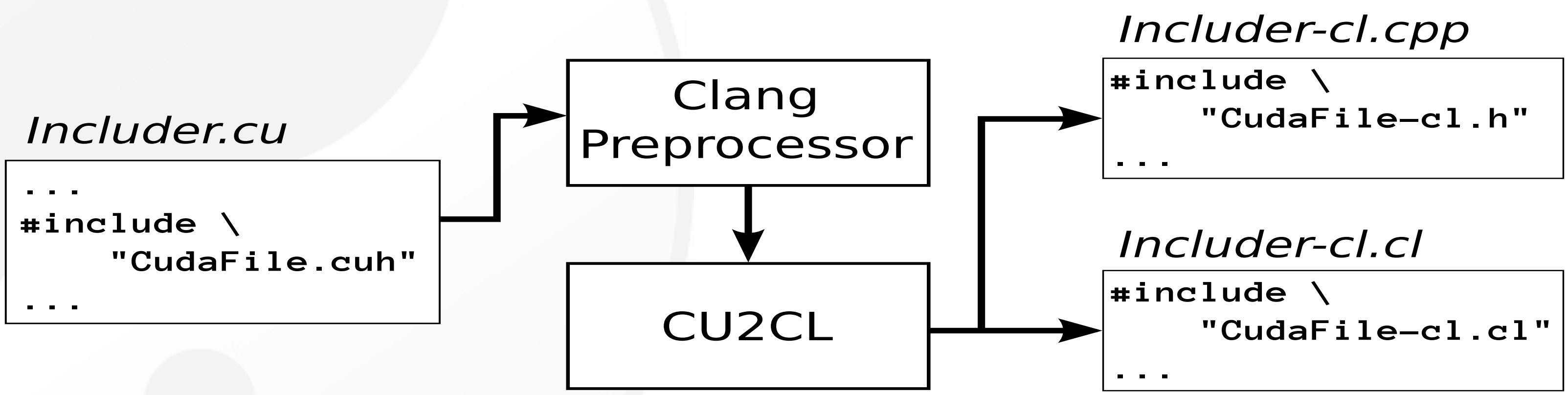


* Abstract Syntax Tree
§ Compute Unified Device Architecture
† Open Computing Language

Translation Procedure of CU2CL

- Traverse the AST
  - Clang's AST library, walking nodes and children
- Identify structures of interest
  - Common patterns arise
- Rewrite original source range as necessary
  - Variable declarations: rewrite type
  - Expressions: recursively rewrite full expression
  - Host code: remove from kernel files
  - Device code: remove from host files
  - #includes: rewrite to point to new files

Rewriting #includes



Rewriting #includes

Includer.cu

```
...
#include \
    "CudaFile.cuh"
...
```

Forecast

- Motivation & Background
- Programming GPGPUs
- How To Run Your CUDA Program Anywhere
  – CU2CL: A CUDA-to-OpenCL Source-to-Source Translator
- Evaluation
  – Coverage, Translation Time, and Performance
- Future Work
- Summary

Experimental Set-Up

- CPU
  - 2 x 2.0-GHz Intel Xeon E5405 quad-core
  - 4 GB of Ram
- GPU
  - NVIDIA GTX 280
  - 1 GB of graphics memory
- Applications
  - CUDA SDK
    - asyncAPI, bandwidthTest, BlackScholes, matrixMul, scalarProd, vectorAdd
  - Rodinia
    - Back Propagation, Breadth-First Search, Hotspot, Needleman-Wunsch, SRAD

CUDA Coverage: CUDA SDK and Rodinia

Source	Application	CUDA Lines	Changed	Percentage
CUDA SDK	asyncAPI	136	4	97.06
	bandwidthTest	891	9	98.99
	BlackScholes	347	4	98.85
	matrixMul	351	2	99.43
	scalarProd	171	4	97.66
	vectorAdd	147	0	100.00
Rodinia	Back Propagation	313	5	98.40
	Breadth-First Search	306	8	97.39
	Hotspot	328	7	97.87
	Needleman-Wunsch	418	0	100.00
	SRAD	541	0	100.00

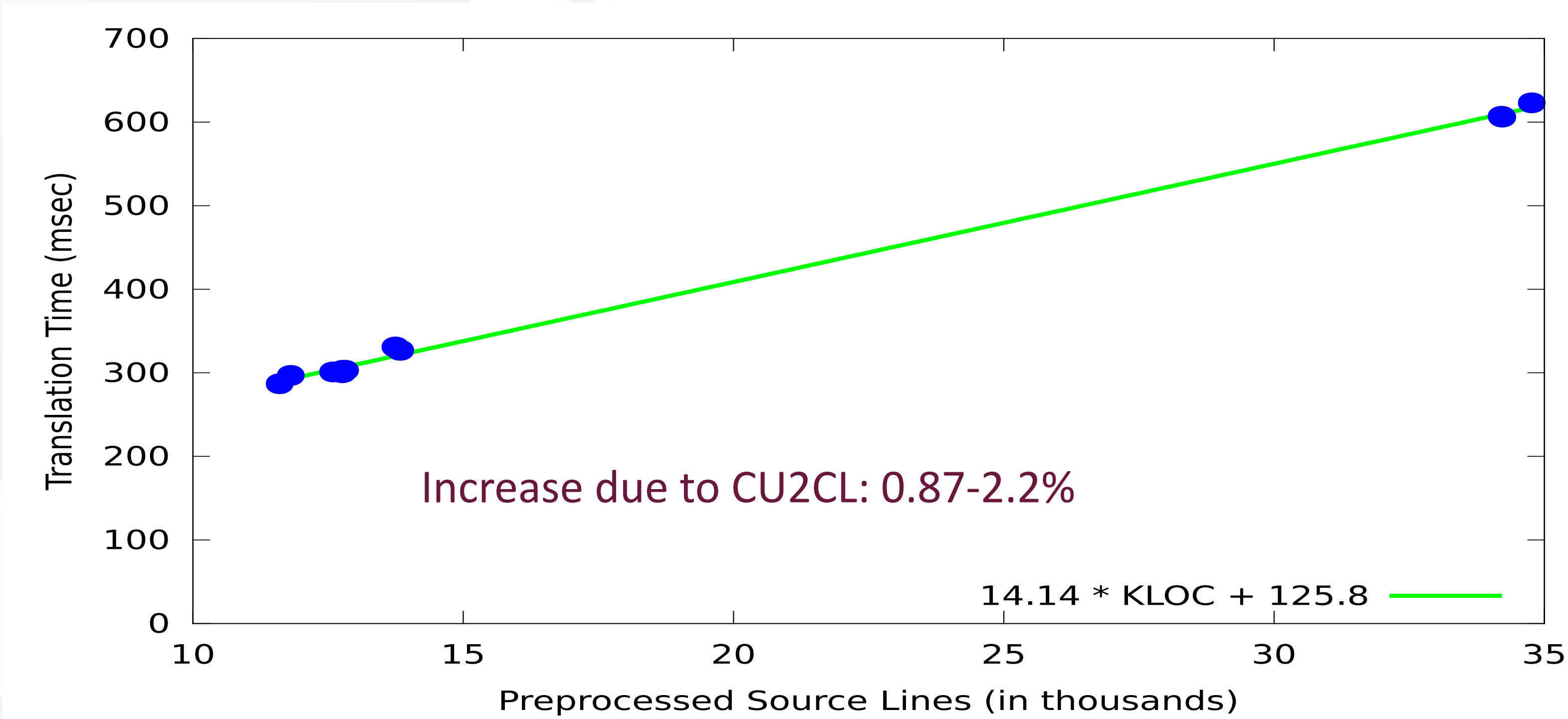
CUDA Coverage: Molecular Modeling App

Source	Application	CUDA Lines	Changed	Percentage
Virginia Tech	GEM	2,511	5	99.8

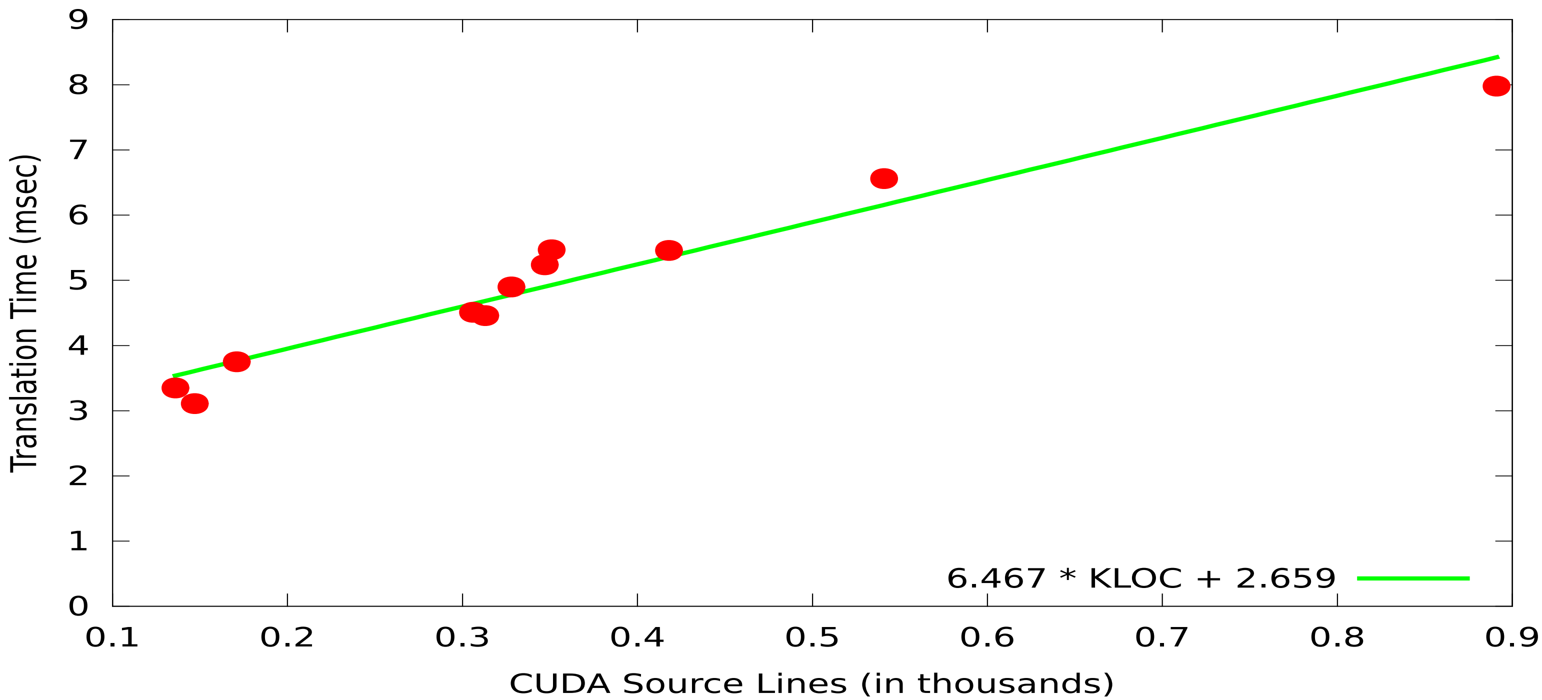
2,511 CUDA lines out of 6,727 total SLOC in GEM application

- *Fundamental Application in Computational Biology*
  - Simulate interactions between atoms & molecules for a period of time by approximations of known physics
- *Example Usage*
  - Understand mechanism behind the function of molecules
    - Catalytic activity, ligand binding, complex formation, charge transport

Model for Total Translation Time



Model for CU2CL-Only Translation Time



VirginiaTech
Invent the Future

SyNeRGy
synergy.cs.vt.edu

Status of OpenCL & the 13 Dwarfs

2009 – 2011
vs.
CU2CL

Dwarf	Done
Dense linear algebra	LU Decomposition
Sparse linear algebra	Matrix Multiplication
Spectral methods	FFT
N-Body methods	GEM
Structured grids	SRAD
Unstructured grids	CFD solver
MapReduce	
Combinational logic	CRC
Graph traversal	BFS, Bitonic sort
Dynamic programming	Needleman-Wunsch
Backtrack and Branch-and-Bound	
Graphical models	Hidden Markov Model
Finite state machines	Temporal Data Mining

Translated Application Performance (sec)

Application	CUDA	Automatic OpenCL	Manual OpenCL
vectorAdd	0.0499	0.0516	0.0521
Hotspot	0.0177	0.0565	0.0561
Needleman-Wunsch	6.65	8.77	8.77
SRAD	1.25	1.55	1.54

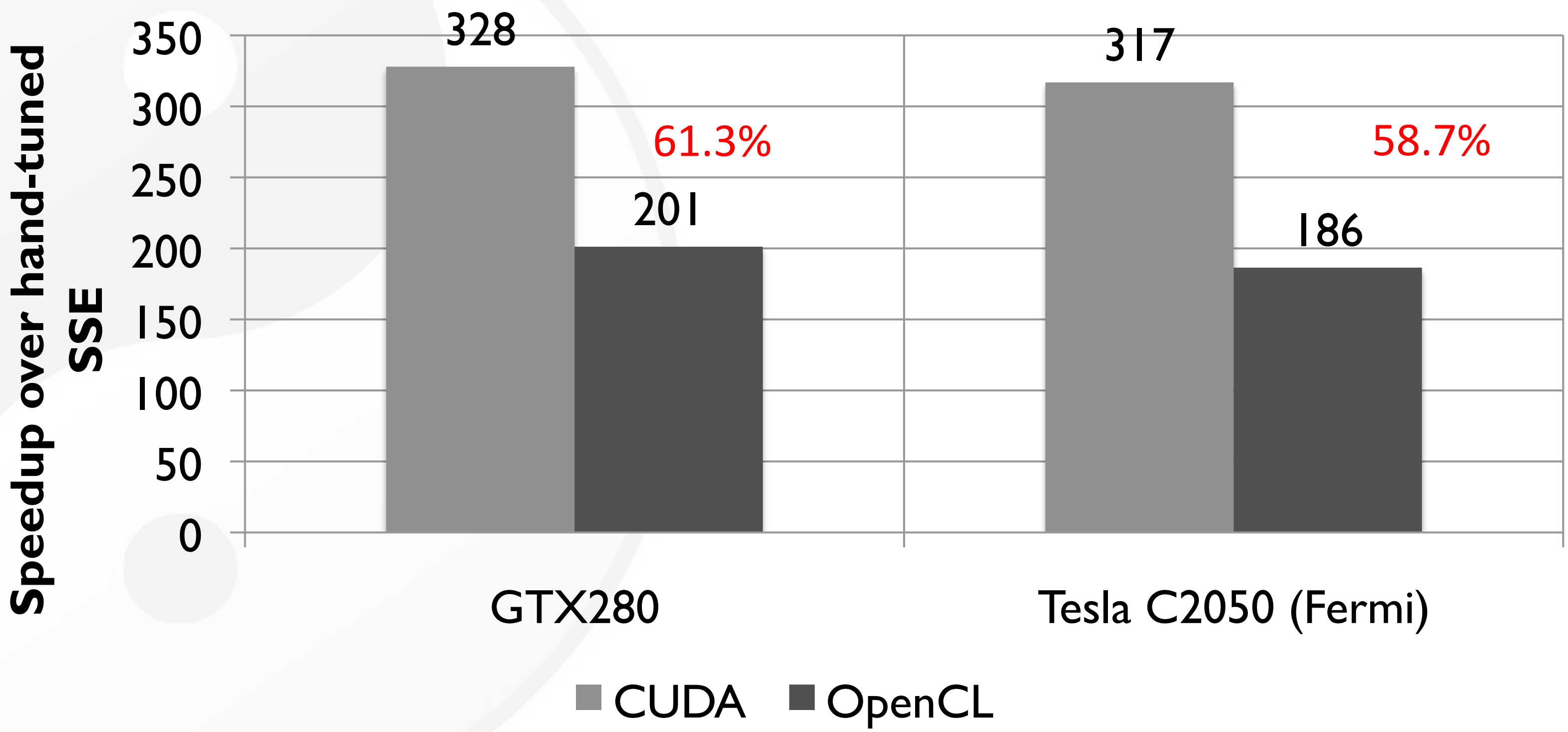
- Automatically translated OpenCL codes yield similar execution times to manually translated OpenCL codes
- OpenCL performance lags CUDA (at least for OpenCL 1.0)

CU2CL with OpenCL and the 13 Dwarfs

Dwarf	Implemented	AMD GPU Unoptimized	NVIDIA GPU Unoptimized	AMD CPU Unoptimized
Dense linear algebra	LU Decomposition			
Sparse linear algebra	Matrix Multiplication			
Spectral methods	FFT			
N-Body methods	GEM	GEM	GEM	GEM
Structured grids	SRAD			
Unstructured grids	CFD Solver			
MapReduce	StreamMR	StreamMR		
Combinational logic	CRC			
Graph traversal	BFS, Bitonic Sort			
Dynamic programming	Needleman-Wunsch		Smith-Waterman	
Backtrack and Branch-and-Bound				
Graphical models	Hidden Markov Model			
Finite state machines	Temporal Data Mining		TDM	

CU2CL

OpenCL Optimization Potential



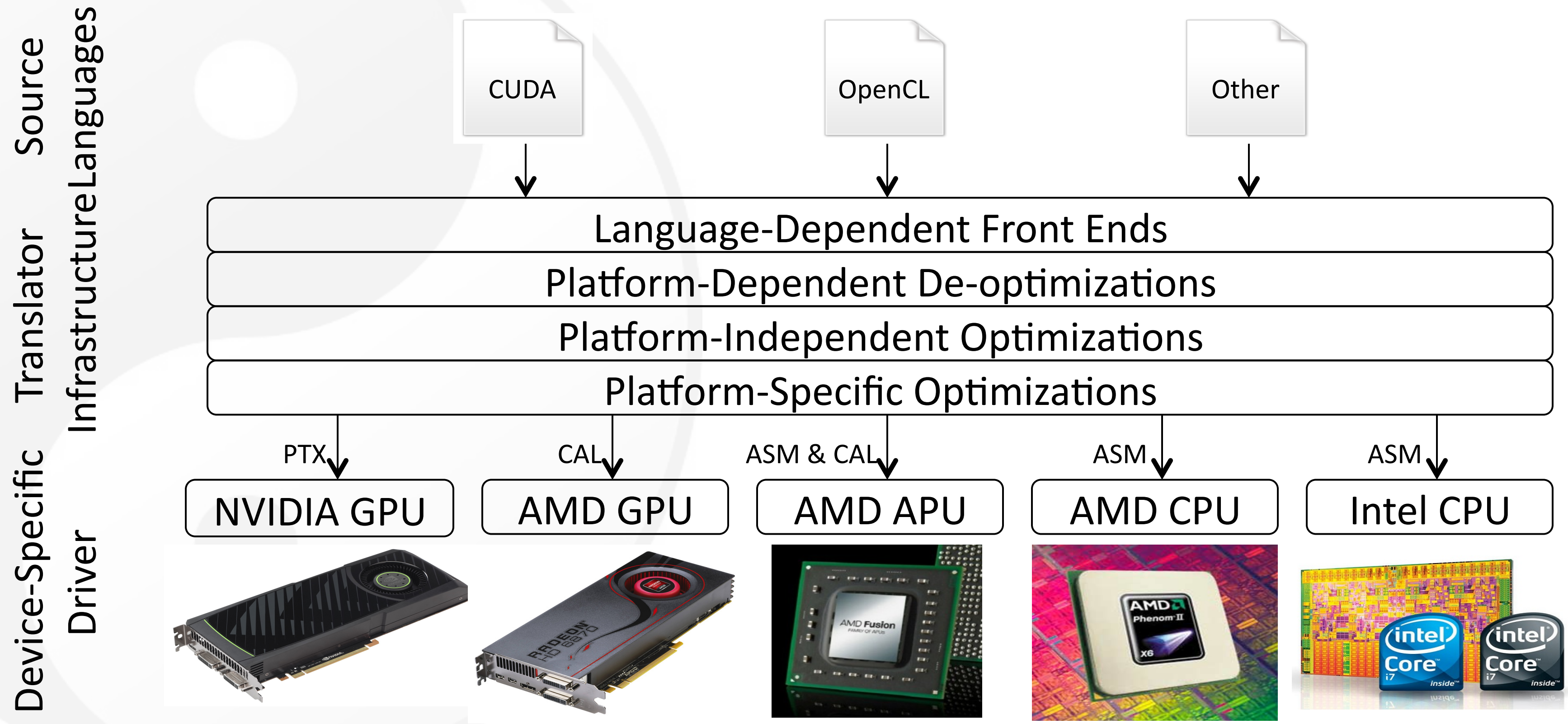
Status of OpenCL & the 13 Dwarfs

2009 – 2011

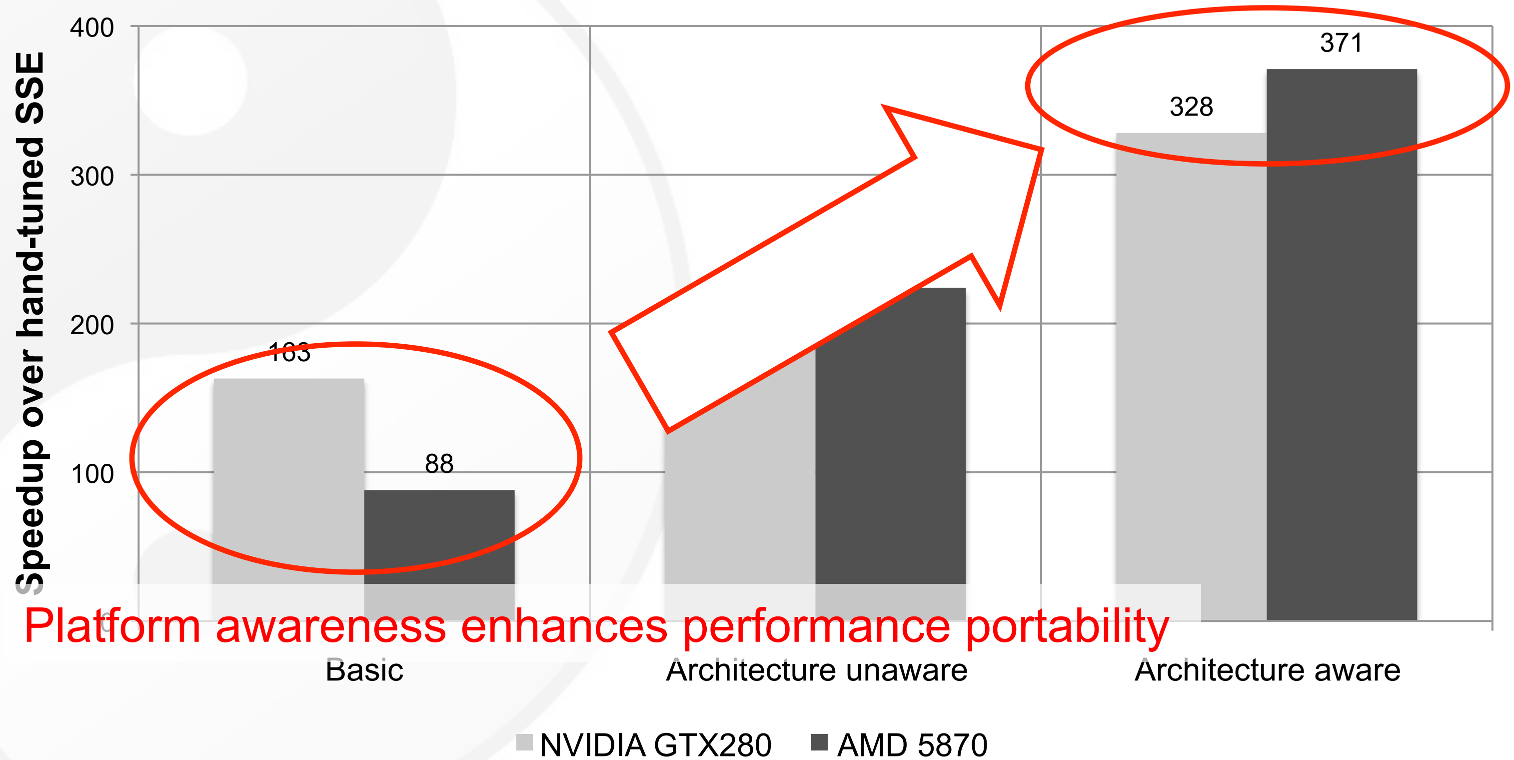
Dwarf	Done
Dense linear algebra	LU Decomposition
Sparse linear algebra	Matrix Multiplication
Spectral methods	FFT
N-Body methods	GEM
Structured grids	SRAD
Unstructured grids	CFD solver
MapReduce	
Combinational logic	CRC
Graph traversal	BFS, Bitonic sort
Dynamic programming	Needleman-Wunsch
Backtrack and Branch-and-Bound	
Graphical models	Hidden Markov Model
Finite state machines	Temporal Data Mining

201x → 328x

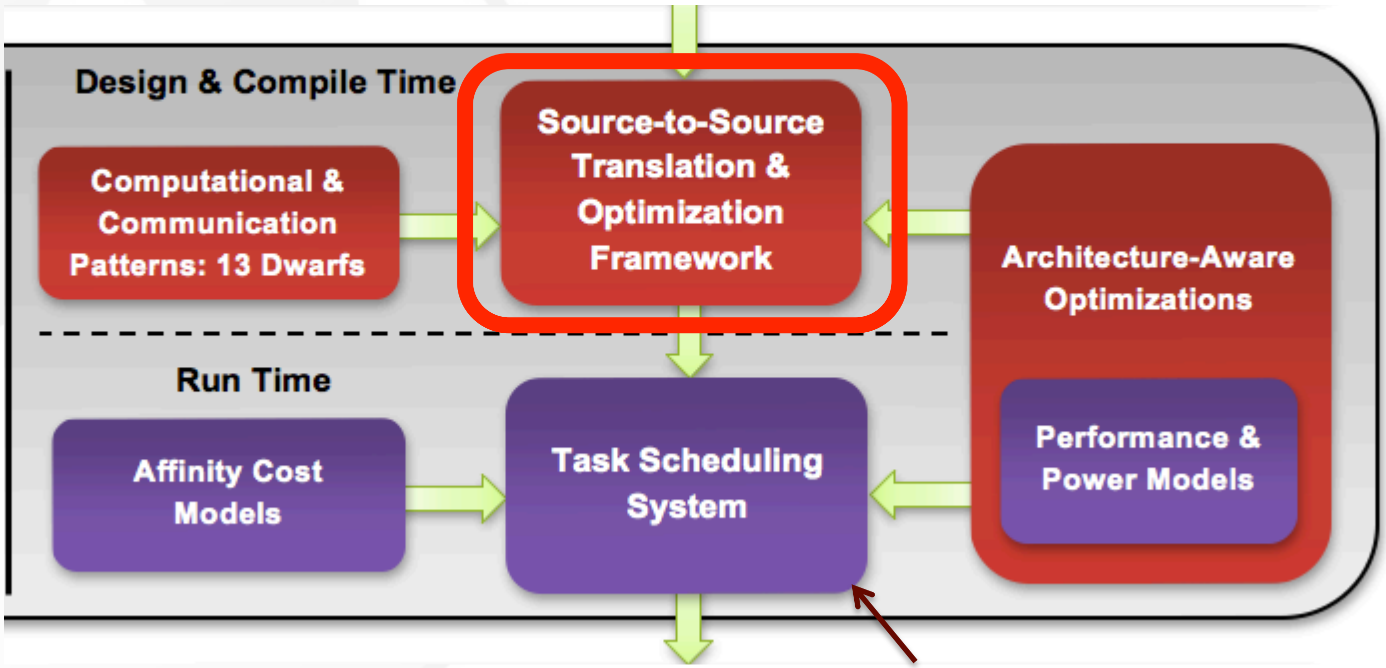
Translator Ecosystem



Potential Due to Optimization



The Bigger Picture



OpenACC “pre-1.0”

VirginiaTech Invent the Future

SyNeRG synergy.cs.vt.edu

CU2CL: Acknowledgments

- Collaborators
  - Gabriel Martinez, M.S. (now at Intel)
  - Mark Gardner, Ph.D.
- Infrastructure
  - Clang compiler and LLVM framework

Contributions

1. General approach for translating CUDA to OpenCL
2. Robust CUDA to OpenCL translator
   – Less than 2000 source lines of code
   – Extends open-source Clang compiler/framework
   – AST-driven, string-based source rewriting ➔ maintainable OpenCL code
3. Already a useful tool
   – Eliminates most hand translation of CUDA 3.x constructs
   – Translated OpenCL performance = hand-translated

- Will be presented at ICPADS in December
- Already receiving many inquiries about release

CU2CL: Lessons Learned

1. Automatic-translated code and hand-translated code from CUDA to OpenCL yields the same performance
2. OpenCL performance is not as good as CUDA as implementations are not as mature
3. A translation ecosystem for performance portability is possible

Questions?