

An Ecosystem for the New HPC: Heterogeneous Parallel Computing

Wu FENG

Professor and Elizabeth & James E. Turner Fellow

Dept. of Computer Science

Dept. of Electrical & Computer Engineering

Health Sciences

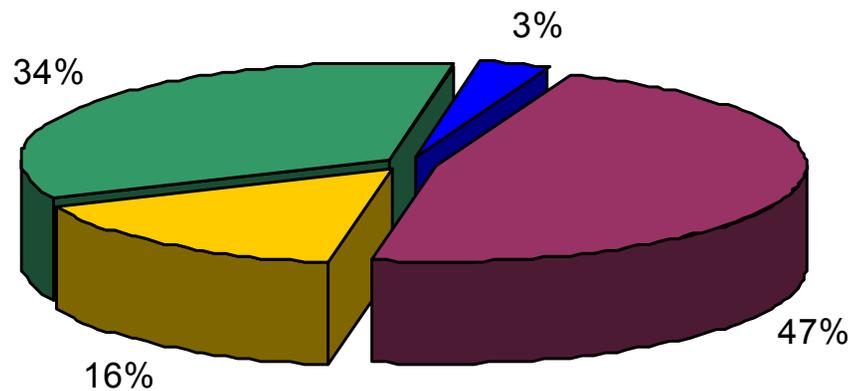
Virginia Bioinformatics Institute

Japanese **'Computnik'** Earth Simulator Shatters U.S. Supercomputer Hegemony

Tokyo 20 April 2002 The Japanese Earth Simulator is on-line and producing results that alarm the USA, that considered itself as being leading in supercomputing technology. With over 35 Tflop/s, it five times outperforms the Ascii White supercomputer that is leading the current TOP500 list. No doubt that position is for the Earth Simulator, not only for the next list, but probably even for

Importance of High-Performance Computing (HPC)

Competitive Risk From Not Having Access to HPC

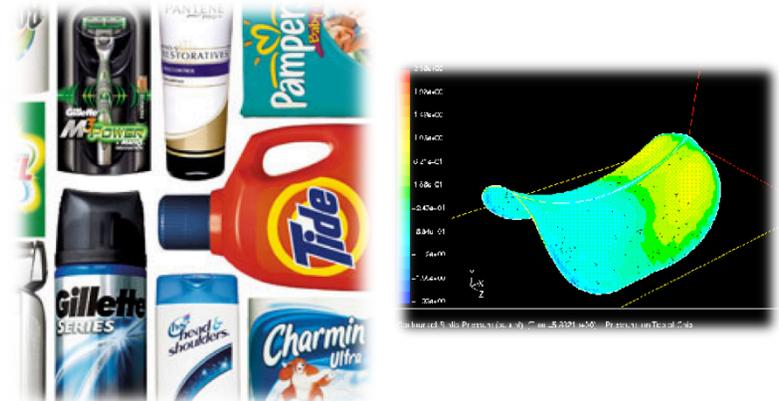


Data from Council of Competitiveness.
Sponsored Survey Conducted by IDC

- Could exist and compete
- Could not exist as a business
- Could not compete on quality & testing issues
- Could not compete on time to market & cost

Only 3% of companies could exist and compete *without* HPC.

- ★ 200+ participating companies, including many Fortune 500 (Proctor & Gamble and biological and chemical companies)



China Wrests Supercomputer Title From U.S.

By ASHLEE VANCE

Published: October 28, 2010

A Chinese scientific research center has built the fastest supercomputer ever made, replacing the United States as maker of the swiftest machine,

 RECOMMEND

 TWITTER

 LINKEDIN

Computnik 2.0?

Tianhe-1A: Computnik Revisited?



- The Second Coming of Computnik? Computnik 2.0?
 - No ... “only” 43% faster than the previous #1 supercomputer, *but*
 - \$20M cheaper than the previous #1 supercomputer
 - 42% less power consumption
- The Second Coming of the “Beowulf Cluster” for HPC
 - The further commoditization of HPC

The First Coming of the “Beowulf Cluster”

- Utilize *commodity* PCs (with commodity CPUs) to build a supercomputer

The Second Coming of the “Beowulf Cluster”

- Utilize *commodity* PCs (with *commodity* CPUs) to build a supercomputer
- +
- Utilize *commodity* graphics processing units (GPUs) to build a supercomputer



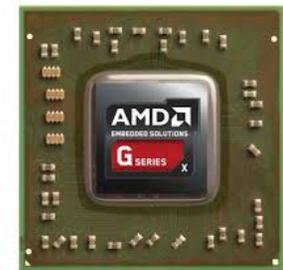
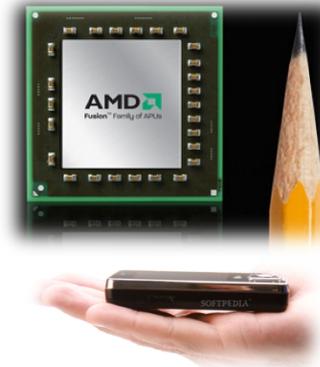
Issue: Extracting *performance* with *programming ease* and *portability* → *productivity*

“Holy Grail” Vision

- Ecosystem for the New HPC: Heterogeneous Parallel Computing



#96 on
TOP500
(11/11)



Highest-ranked commodity supercomputer
in U.S. on the **Green500** (11/11)

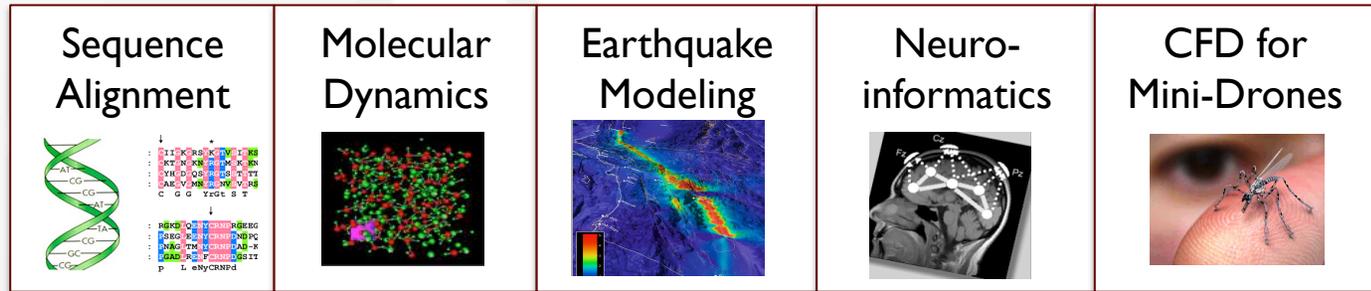
- Enabling software that tunes parameters of hardware devices
... with respect to **performance, programmability, and portability**
... via a benchmark suite of dwarfs (i.e., motifs) and mini-apps

Intra-Node

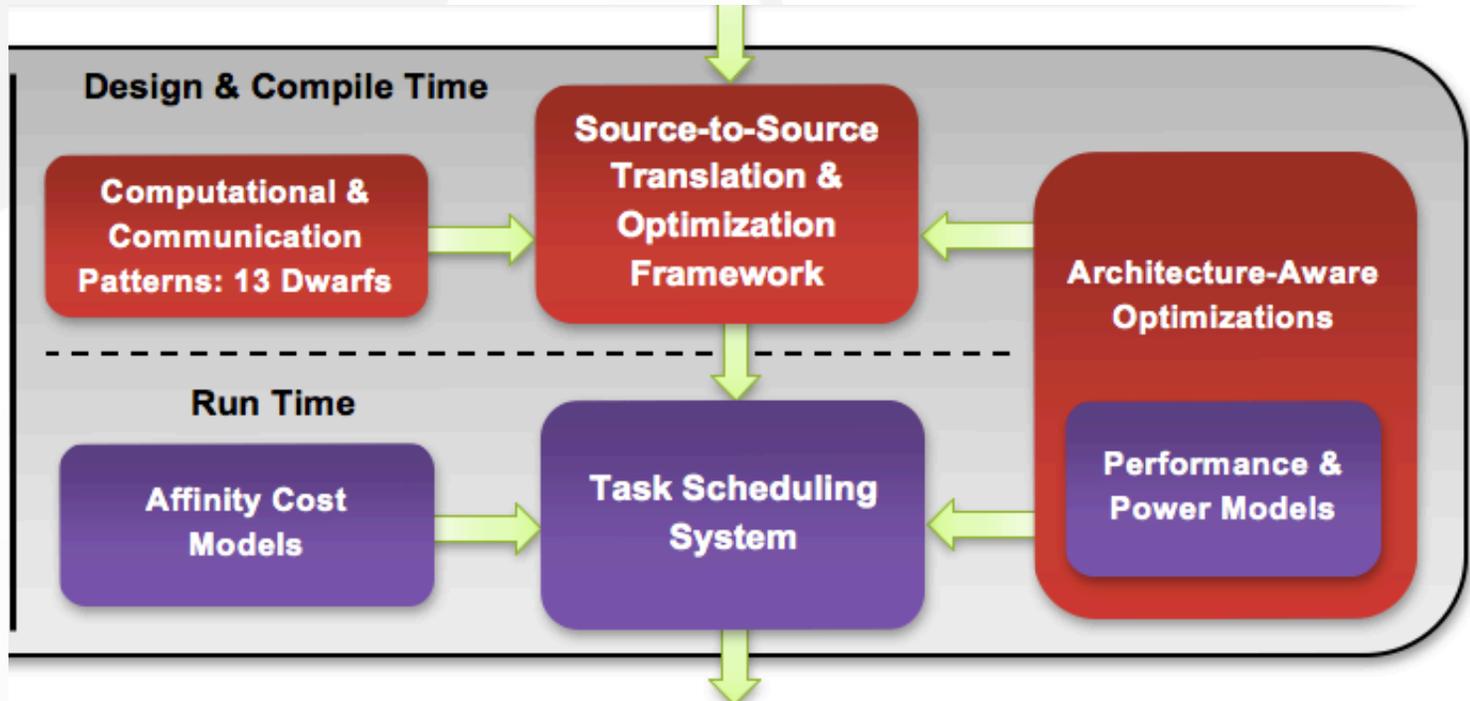
An Ecosystem for Heterogeneous Parallel Computing

^

Applications



Software Ecosystem

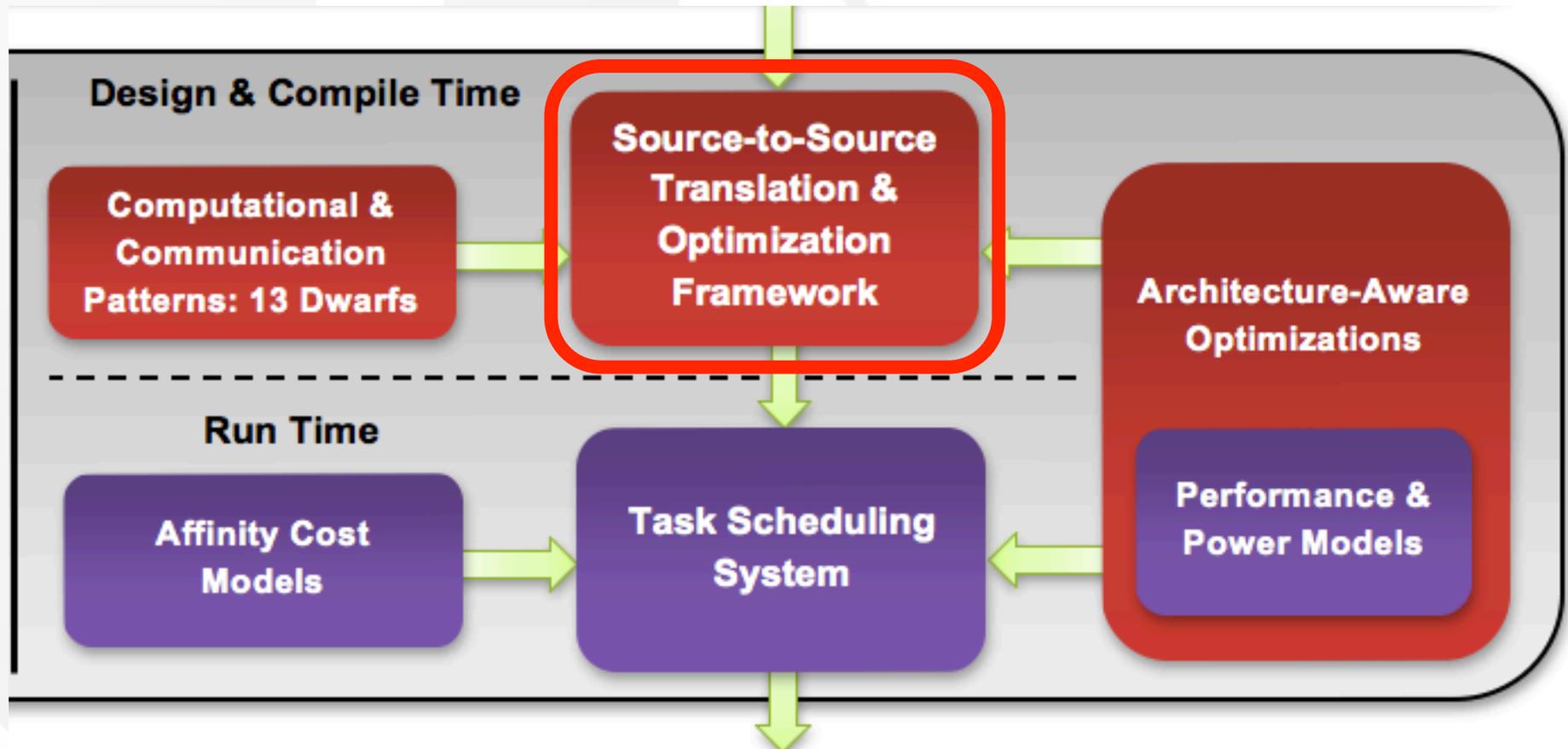


Heterogeneous Parallel Computing (HPC) Platform

Performance, Programmability, Portability

Roadmap

Goal: Minimize the re-writing of code, e.g., CFD for mini-drones.
CUDA → OpenCL and OpenMP → OpenACC



Programming GPUs

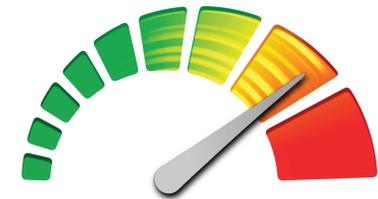
CUDA

- NVIDIA's proprietary framework



OpenCL

- Open standard for heterogeneous parallel computing (Khronos Group)
- Vendor-neutral environment for CPUs, GPUs, APUs, and even FPGAs



OpenCL

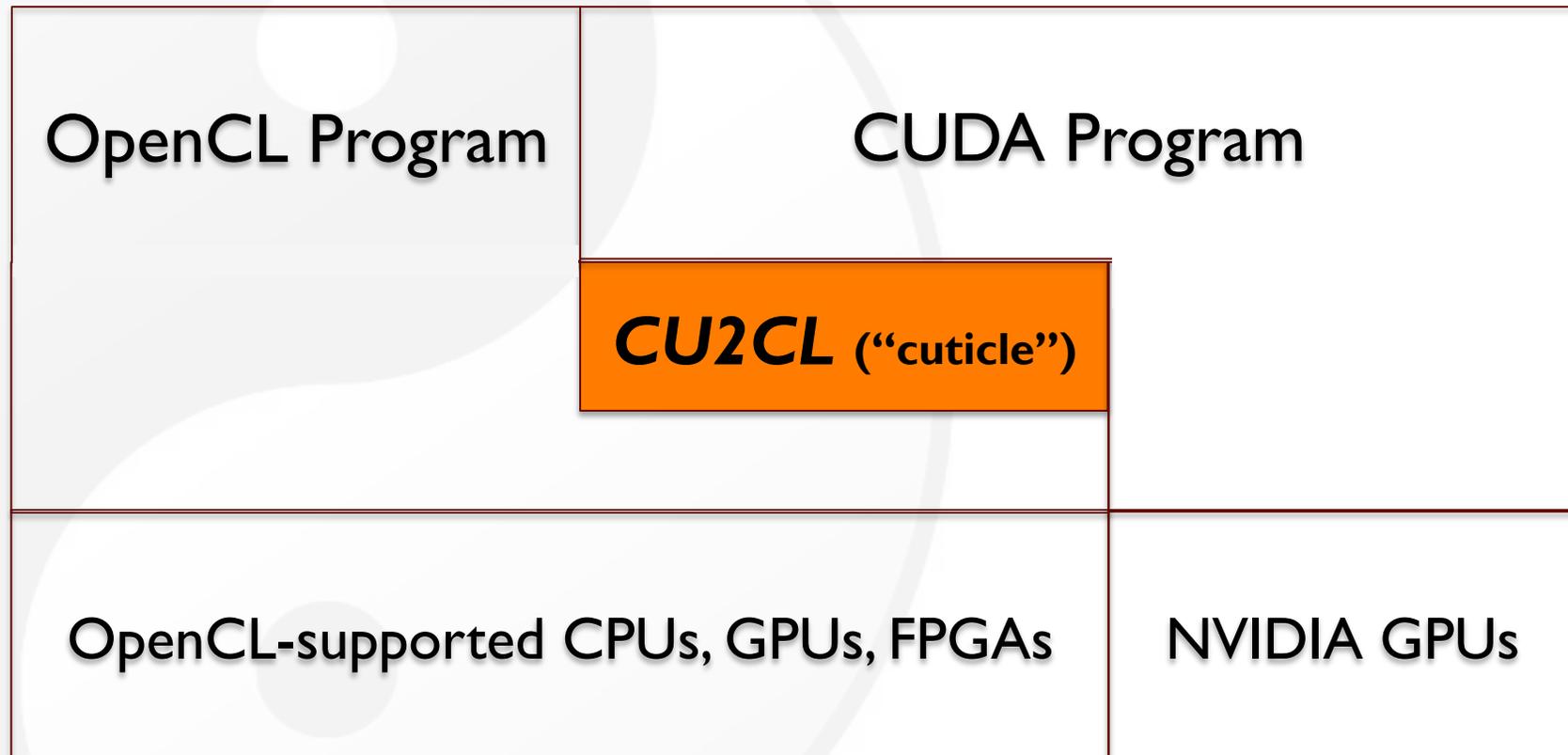
OpenACC

CU2CL: CUDA-to-OpenCL Source-to-Source Translator†

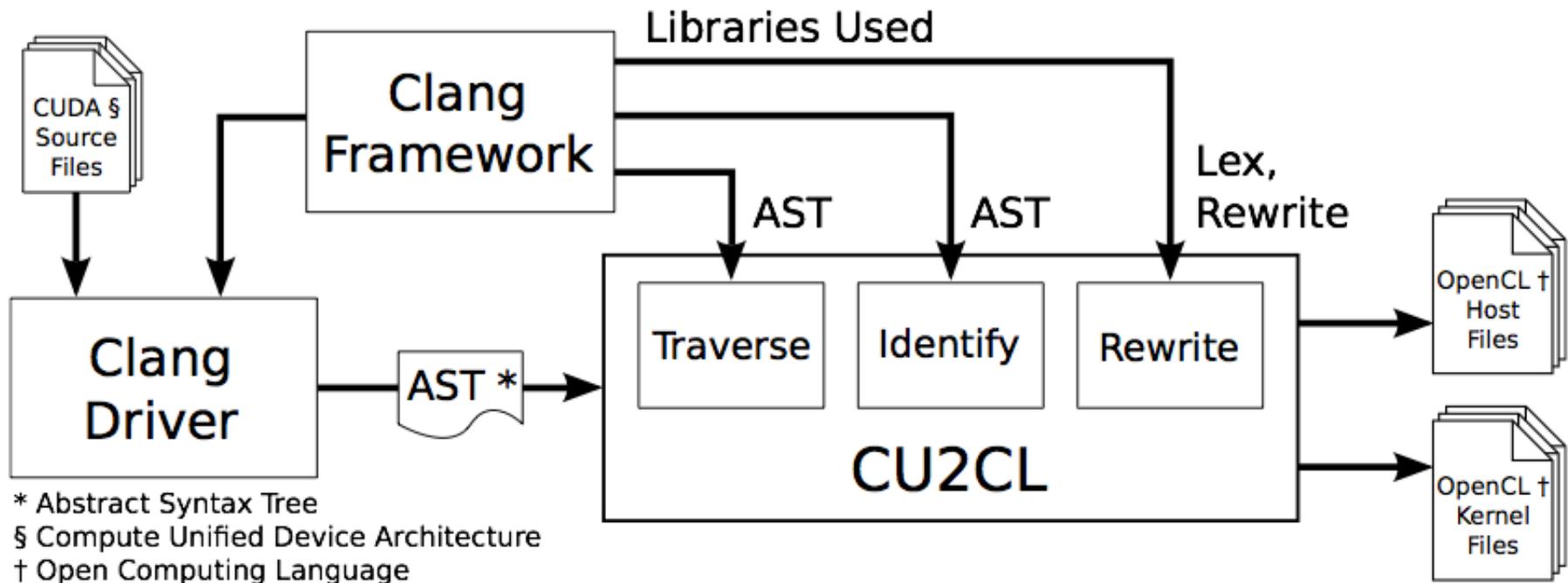
- Works as a Clang plug-in to leverage its production-quality compiler framework.
- Covers primary CUDA constructs found in CUDA C and CUDA run-time API.
- Delivers performance portability when OpenCL 1.2-equivalent CUDA code run on same NVIDIA GPU.
- Focuses on *functional portability* ... for now.

† “CU2CL: A CUDA-to-OpenCL Translator for Multi- and Many-core Architectures,” 17th IEEE Int’l Conf. on Parallel & Distributed Systems (ICPADS), Dec. 2011.

OpenCL: Write Once, Run Anywhere



CU2CL Translation and Performance

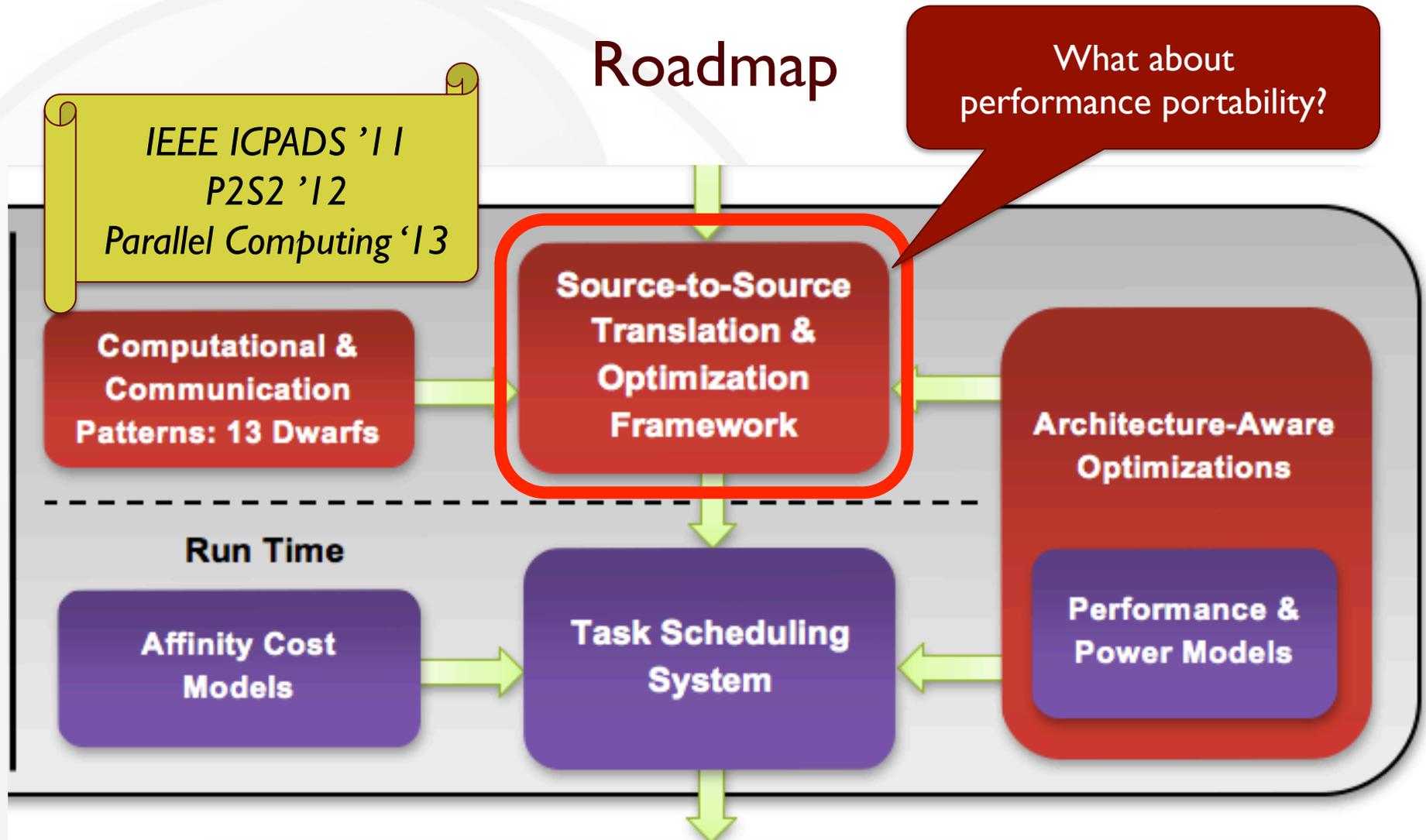


- Automatically translated OpenCL codes (via CU2CL) yield similar execution times to manually translated OpenCL codes (when running on the same device)

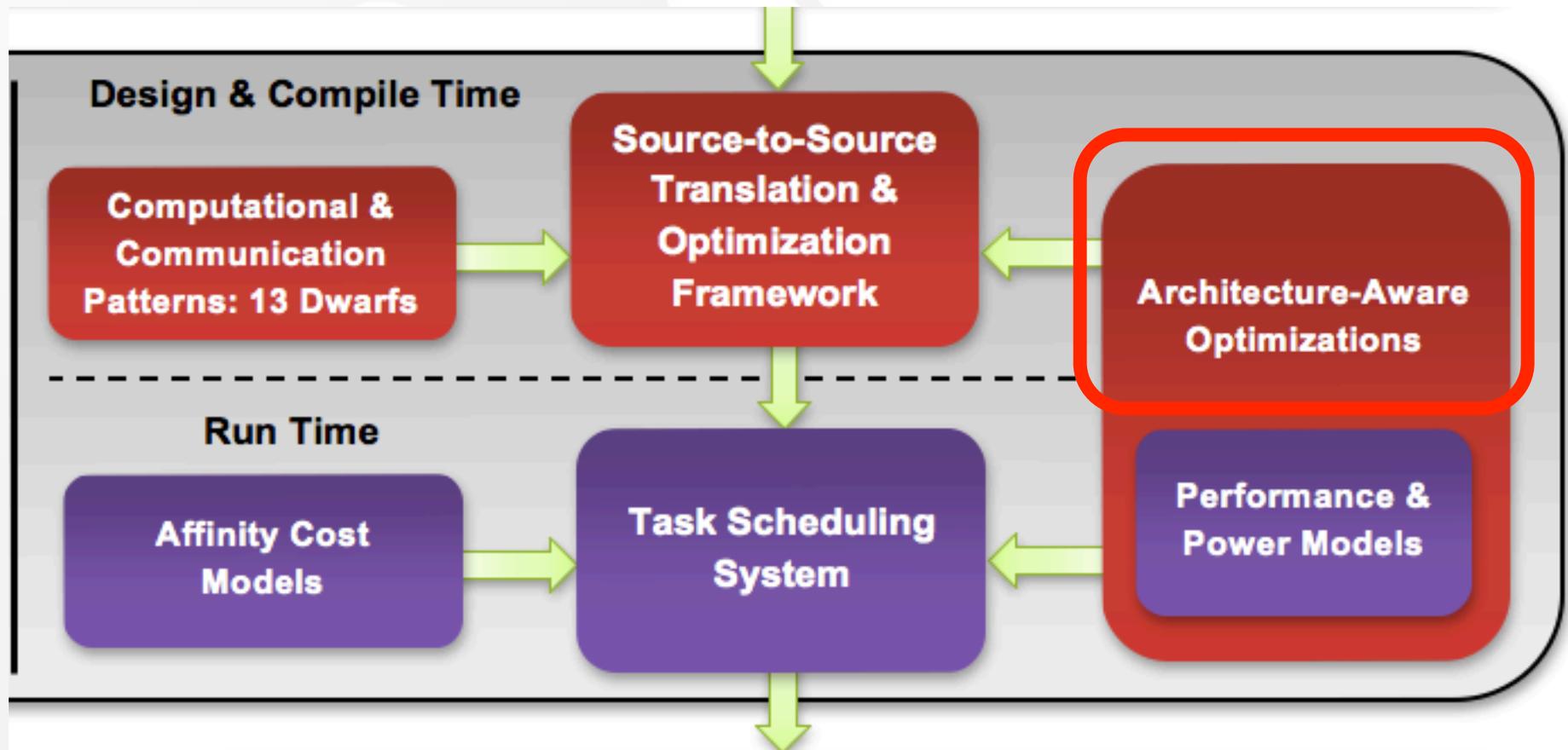
	Application	CUDA Lines	Lines Manually Changed	% Auto-Translated
CUDA SDK	bandwidthTest	891	5	98.9
	BlackScholes	347	14	96.0
	matrixMul	351	9	97.4
	vectorAdd	147	0	100.0
Rodinia	Back Propagation	313	24	92.3
	Hotspot	328	2	99.4
	Needleman-Wunsch	430	3	99.3
	SRAD	541	0	100.0
Delaware	Fen Zi: Molecular Dynamics	17,768	1,796	89.9
VT	GEM: Molecular Modeling	524	15	97.1
AMD	IZ PS: Neural Network	8,402	166	98.0

Performance, Programmability, Portability

Roadmap

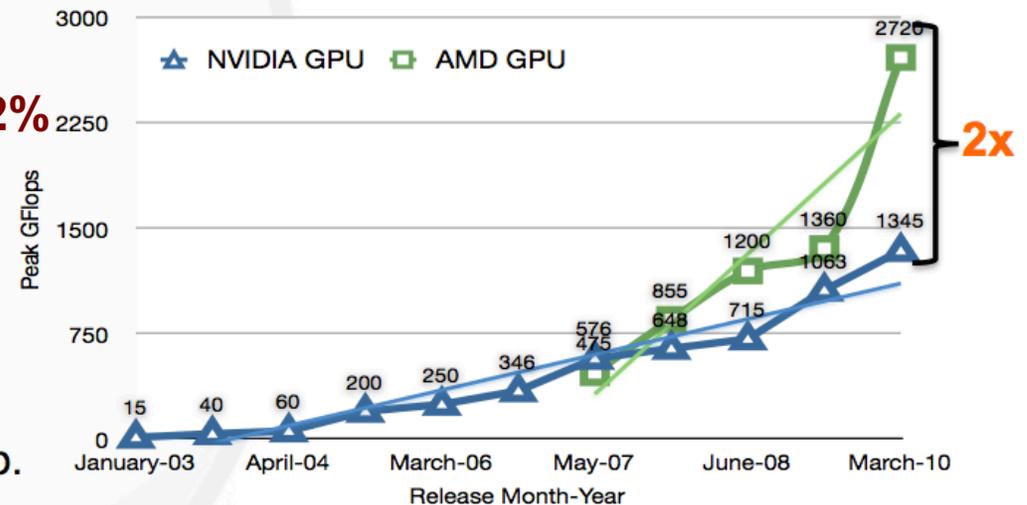
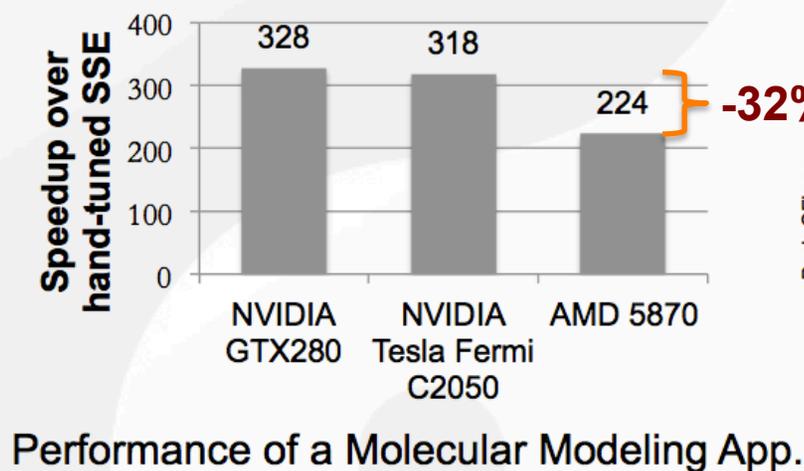


Roadmap



Computational Units *Not* Created Equal

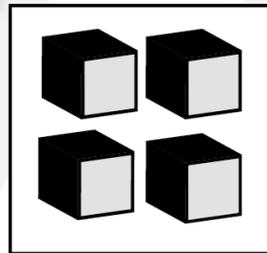
- “AMD CPU \neq Intel CPU” and “AMD GPU \neq NVIDIA GPU”
- Initial performance of a *CUDA-optimized* N-body dwarf



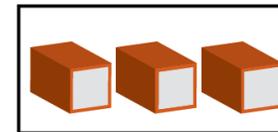
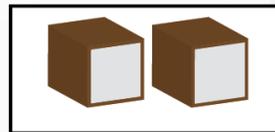
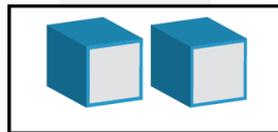
Traditional Approach to Optimizing Compiler

Architecture-Independent Optimization

General Optimizations



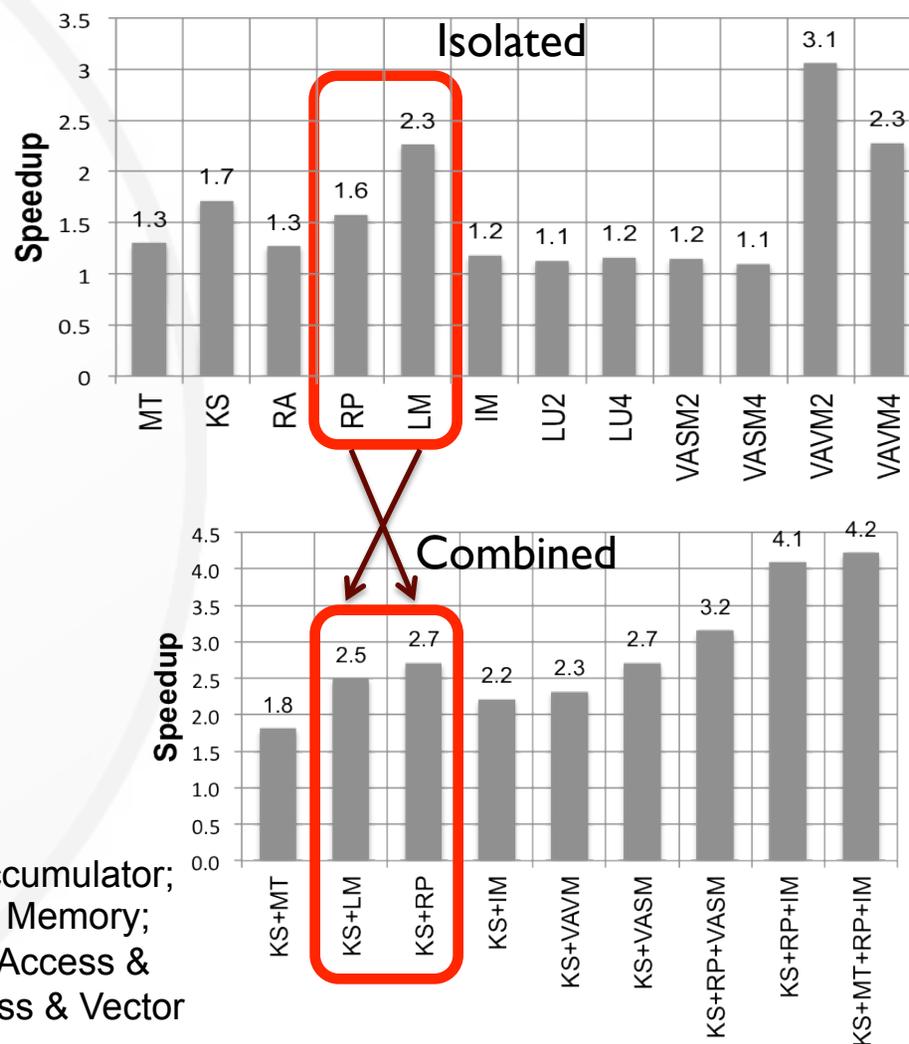
Architecture-Aware Optimization



- C. del Mundo, W. Feng. "Towards a Performance-Portable FFT Library for Heterogeneous Computing," in IEEE IPDPS '13. Phoenix, AZ, USA, May 2014. (Under review.)
- C. del Mundo, W. Feng. "Enabling Efficient Intra-Warp Communication for Fourier Transforms in a Many-Core Architecture", SC'13, Denver, CO, Nov. 2013.
- C. del Mundo, V. Adhinarayanan, W. Feng, "Accelerating FFT for Wideband Channelization." in IEEE ICC '13. Budapest, Hungary, June 2013.

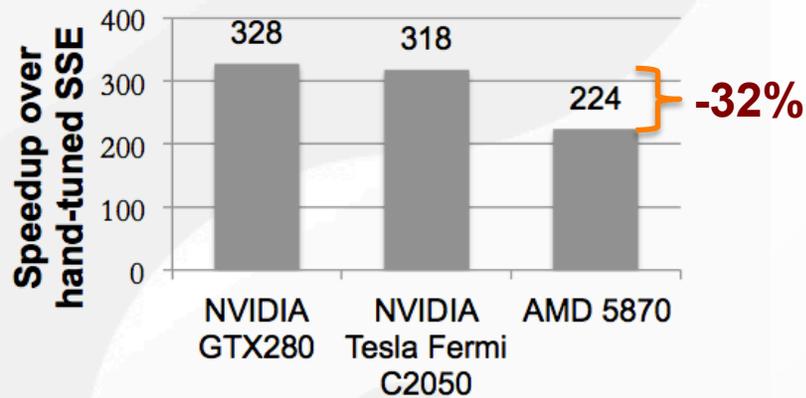
Optimization for N-body Molecular Modeling

- Optimization techniques on AMD GPUs
 - Removing conditions → kernel splitting
 - Local staging
 - Using vector types
 - Using image memory
- Speedup over basic OpenCL GPU implementation
 - Isolated optimizations
 - Combined optimizations



MT: Max Threads; **KS**: Kernel Splitting; **RA**: Register Accumulator;
RP: Register Preloading; **LM**: Local Memory; **IM**: Image Memory;
LU{2,4}: Loop Unrolling{2x,4x}; **VASM{2,4}**: Vectorized Access &
 Scalar Math{float2, float4}; **VAVM{2,4}**: Vectorized Access & Vector
 Math{float2, float4}

AMD-Optimized N-body Dwarf



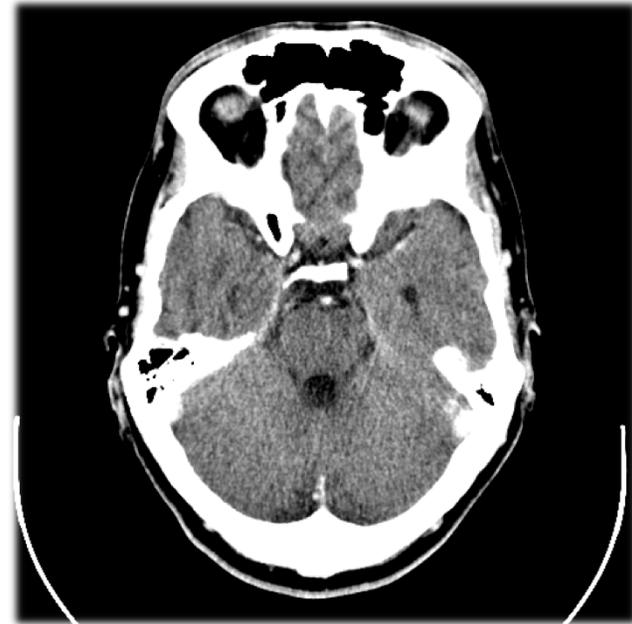
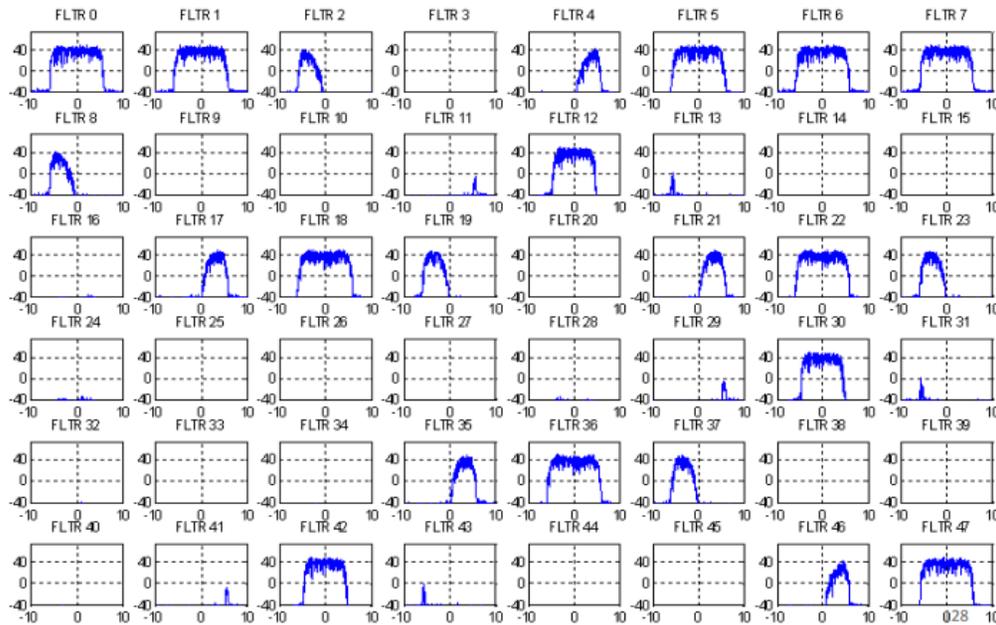
371x speed-up

- 12% better than NVIDIA GTX 280

Performance of a Molecular Modeling App.

FFT: Fast Fourier Transform

- A spectral method that is a critical building block across many disciplines



Summary of Optimizations

Codename	Name	Description
LM-CT	Local Memory (Computation, No Transpose)	Data elements are loaded into local memory for computation. The communication step is avoided by algorithm reorganization [8].
LM-CC	Local Memory (Computation and Communication)	All data elements are preloaded into local memory. Computation is performed in local memory, while registers are used for scratchpad communication.
LM-CM	Local Memory (Communication Only)	Data elements are loaded into local memory only for communication. Threads swap data elements solely in local memory. This optimization requires only $N \times sz(\text{float})$ local memory by transposing each dimension of a floatn vector one dimension at a time.
CM- $\{K,L\}$	Constant Memory $\{\text{Kernel, Literal}\}$	The twiddle multiplication stage can be pre-computed on the host and stored in constant memory for fast look up. This saves two transcendental single-precision operations at the cost of a cached memory access. CM-K refers to the usage of constant memory as a kernel argument, while CM-L refers to a static global declaration in the OpenCL kernel.
RP	Register Preloading	All data elements are first preloaded onto the register file of the respective GPU. Computation is facilitated solely on registers.
CGAP	Coalesced Global Access Pattern	Threads in a wavefront access memory contiguously, e.g. the kth thread accesses memory element, k.
CSE	Common Subexpression Elimination	A traditional optimization that collapses identical expressions in order to save computation. This optimization may increase register live time, therefore, increasing register pressure.
IL	(Function) Inlining	A function's code body is inserted in place of a function call. It is used primarily for functions that are frequently called.
LU	Loop Unrolling	A loop is explicitly rewritten as an identical sequence of statements without the overhead of loop variable comparisons.
VASM $\{2,4,8,16\}$	Vector Access, Scalar Math float $\{2,4,8,16\}$	Data elements are loaded as the listed vector type. Arithmetic operations are scalar (float \times float).
VAVM $\{2,4,8,16\}$	Vector Access, Vector Math float $\{2,4,8,16\}$	Data elements are loaded as the listed vector type. The arithmetic operations are vectorized with the sizes listed, (floatn \times floatn).
SHFL	Shuffle	The transpose stage in FFT is performed entirely in registers eliminating the use of local memory. This optimization is only specific to NVIDIA Tesla K20c

Optimizations

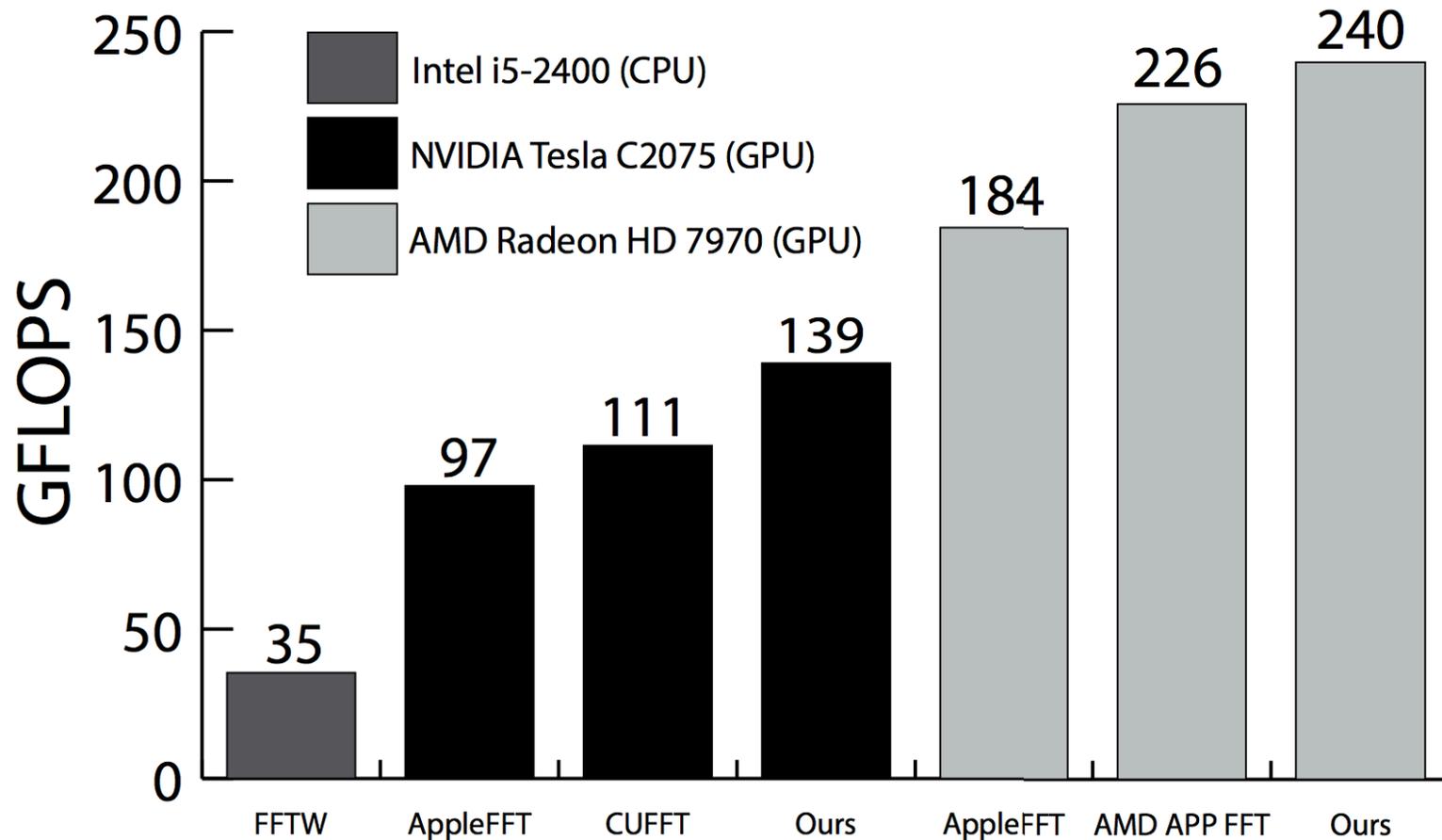
- **RP: Register Preloading**
 - All data elements are first preloaded into the register file before use.
 - Computation facilitated solely on registers
- **LM-CM: Local Memory (Communication Only)**
 - Data elements are loaded into local memory only for communication
 - Threads swap data elements solely in local memory
- **CGAP: Coalesced Global Access Pattern**
 - Threads access memory contiguously
- **VASM{2|4}: Vector Access, Scalar Math, float{2|4}**
 - Data elements are loaded as the listed vector type.
 - Arithmetic operations are scalar (float \times float).
- **CM-K: Constant Memory for Kernel Arguments**
 - The twiddle multiplication state of FFT is precomputed on the CPU and stored in GPU constant memory for fast look-up.

Architecture-Optimized FFT

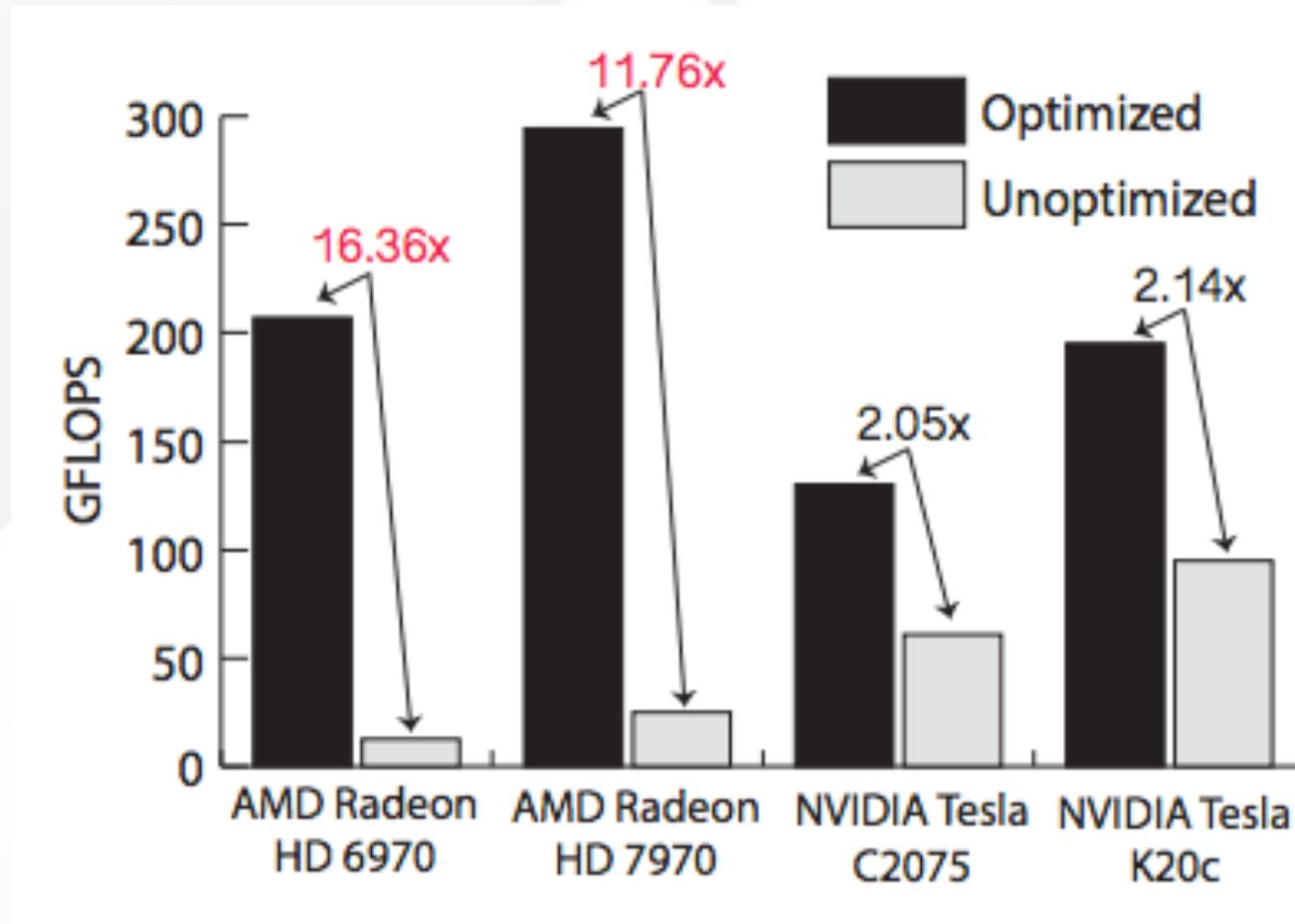
Device	Sample Size	Baseline (GFLOPS)	Optimal (GFLOPS)	Speedup	Speedup over FFTW	Optimal Set
Intel i5-2400 (FFTW)	16		36			
	64		43			
	256		48			
Radeon HD 6970	16	12	174	14.5x	4.8x	RP + LM-CM + CGAP + VASM4 + CM-K
	64	14	257	18.4x	6.0x	RP + LM-CM + CGAP + VASM4 + CM-K
	256	11	346	31.5x	7.2x	RP + LM-CM + CGAP + VASM2 + CM-K
Radeon HD 7970	16	36	240	6.7x	6.7x	RP + LM-CM + CGAP + VASM4 + CM-K
	64	23	366	15.9x	8.5x	RP + LM-CM + CGAP + VASM2 + CM-K
	256	24	437	18.2x	9.1x	RP + LM-CM + CGAP + VASM2 + CM-K
Tesla C2075	16	37	139	3.7x	3.9x	RP + LM-CM + CGAP + VASM2 + CM-K
	64	69	200	2.9x	4.7x	RP + LM-CM + CGAP + VASM4 + CM-K
	256	60	177	3.0x	3.7x	RP + LM-CM + CGAP + VASM2 + CM-K
Tesla K20c	16	54	183	3.4x	5.1x	RP + LM-CM + CGAP + VASM2 + CM-K
	64	99	265	2.7x	6.2x	RP + LM-CM + CGAP + VASM4 + CM-K
	256	95	280	2.9x	5.8x	RP + LM-CM + CGAP + VASM2 + CM-K

Architecture-Optimized FFT

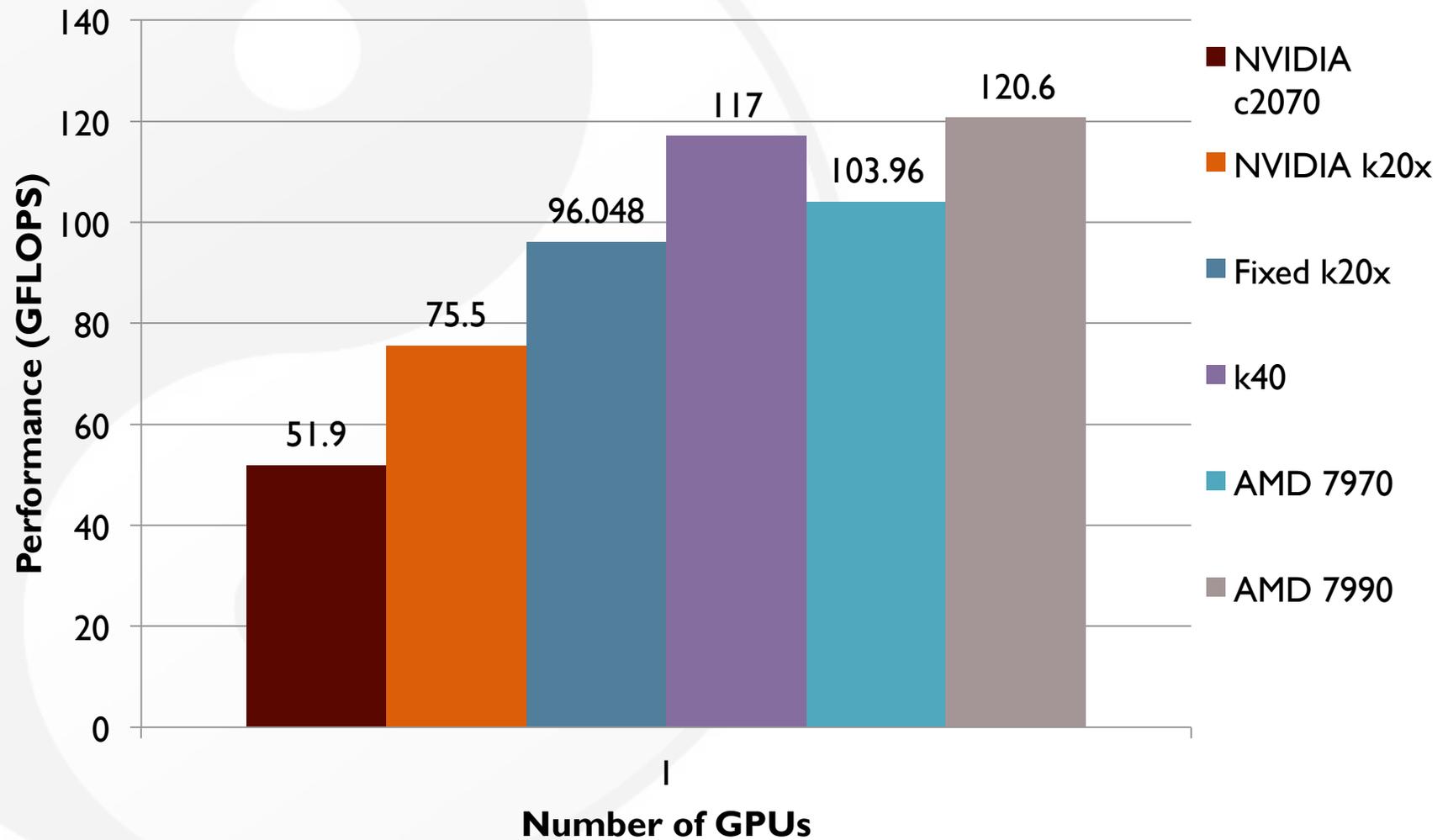
(Batched, Single Precision, 1-D, 16-pt)



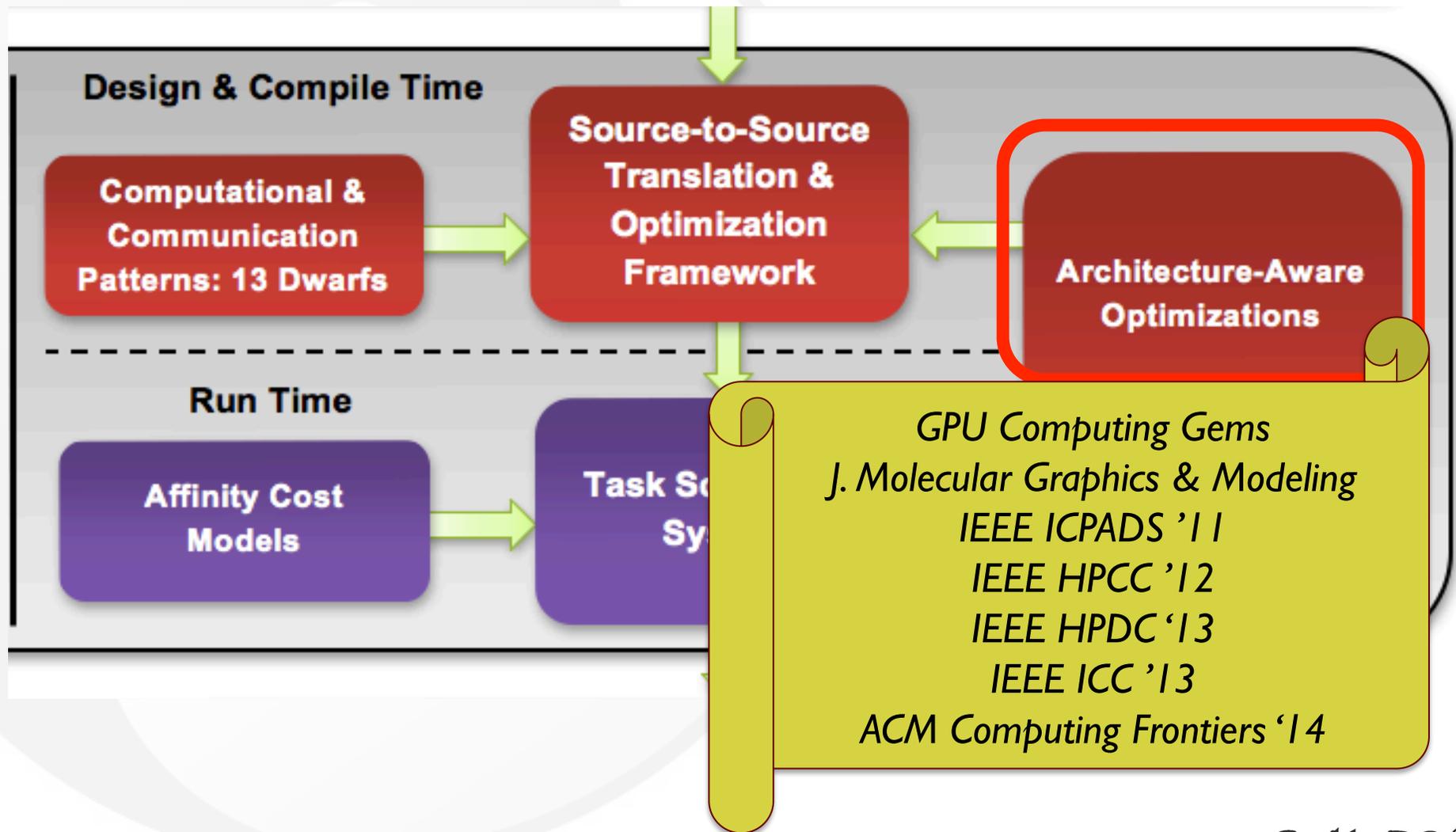
Architecture-Optimized 2D FFT (256 × 256)



Architecture-Optimized Lid-Driven Cavity

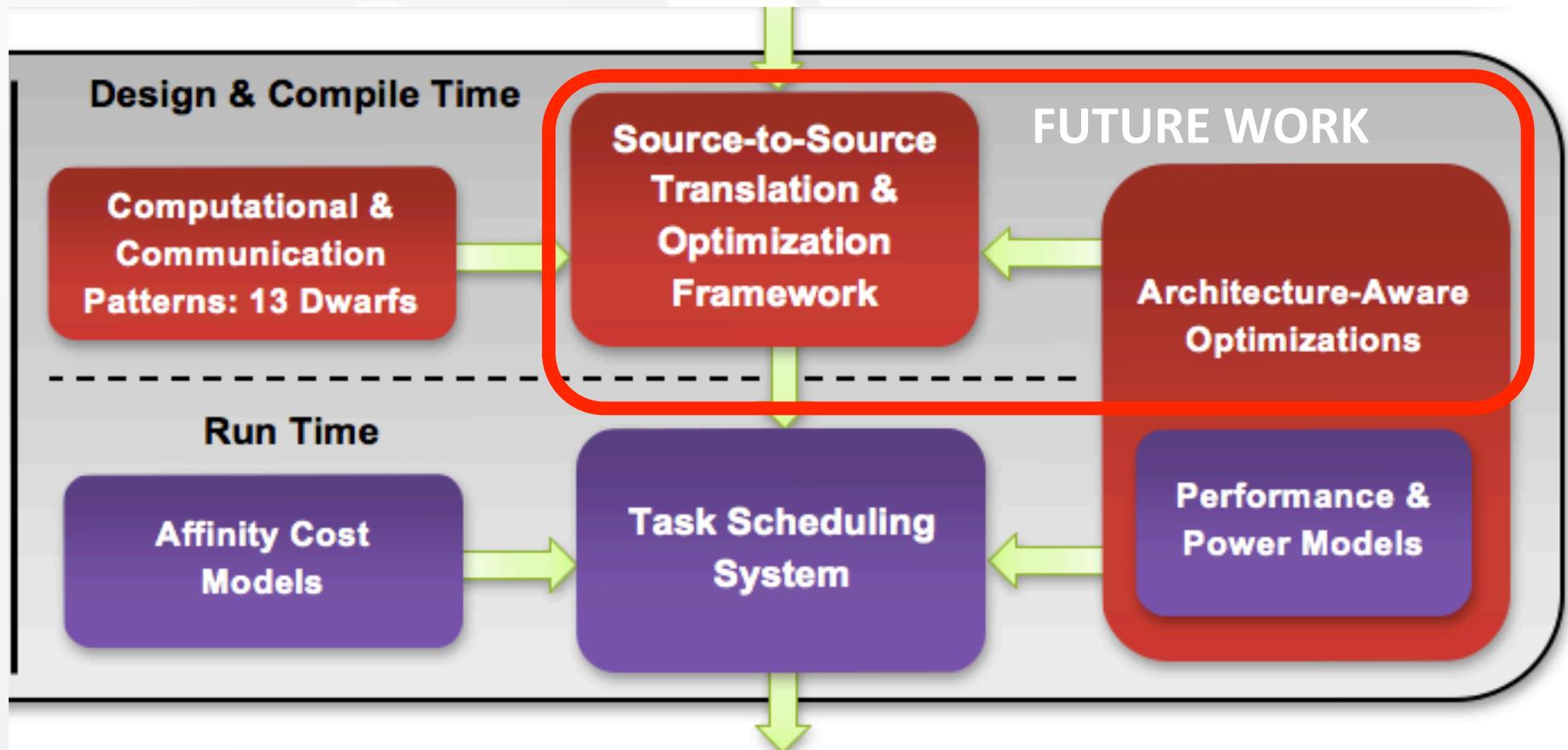


Roadmap

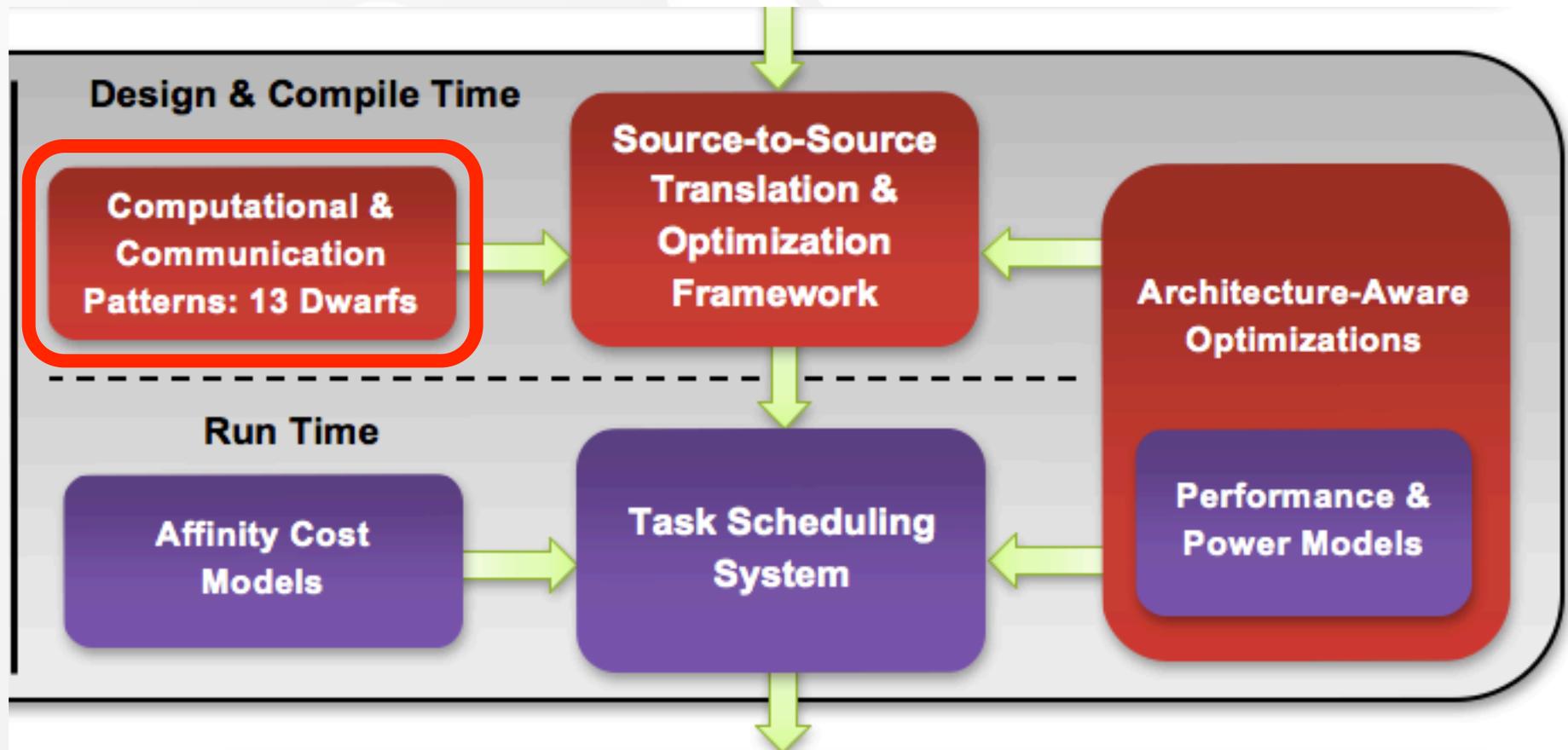


Performance, Programmability, Portability

Roadmap



Roadmap



Paying For Performance

- “The free lunch is over..” †
 - Programmers can no longer expect substantial increases in single-threaded performance.
 - The burden falls on developers to exploit parallel hardware for performance gains.
- How do we lower the cost of concurrency?

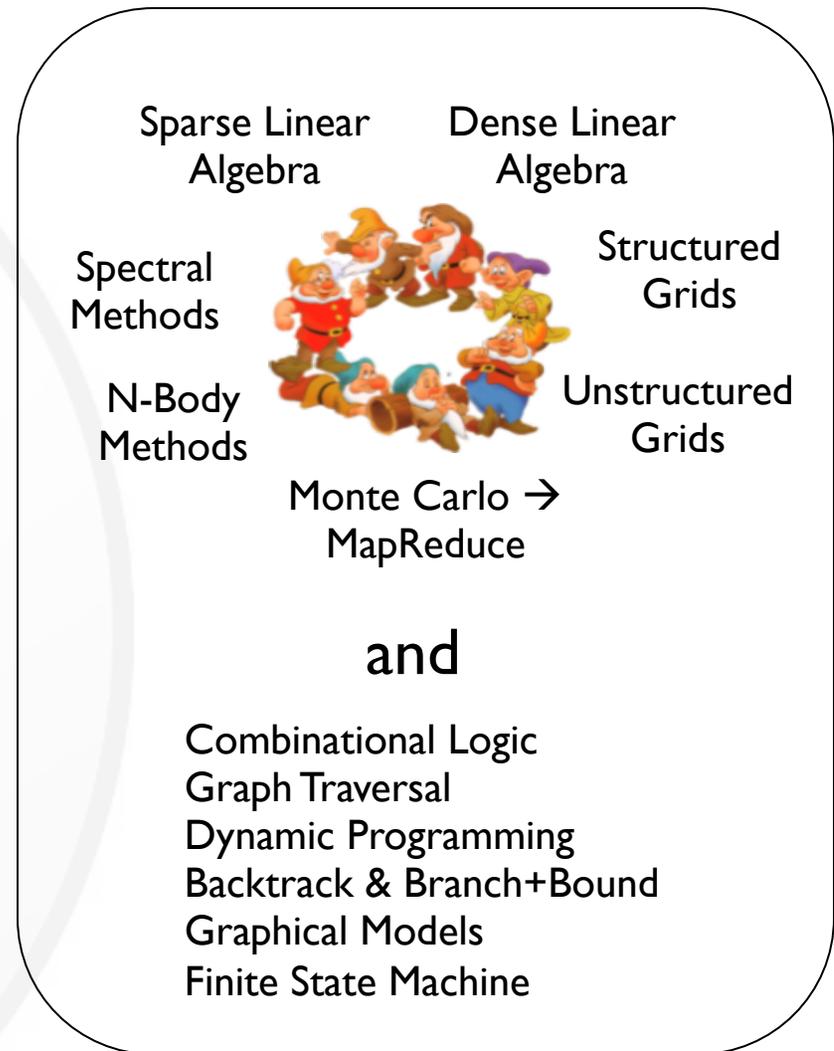
† H. Sutter, “The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software,” *Dr. Dobbs’ Journal*, 30(3), March 2005. (Updated August 2009.)

The Berkeley View †

- Traditional Approach
 - Applications that target existing hardware and programming models
- Berkeley Approach
 - Hardware design that keeps future applications in mind
 - Basis for future applications?
13 computational dwarfs

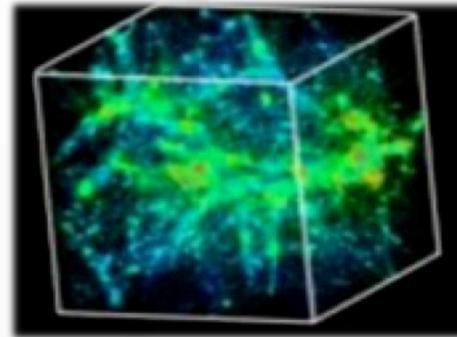
A computational dwarf is a pattern of communication & computation that is common across a set of applications.

† Asanovic, K., et al. *The Landscape of Parallel Computing Research: A View from Berkeley*. Tech. Rep. UCB/EECS-2006-183, University of California, Berkeley, Dec. 2006.

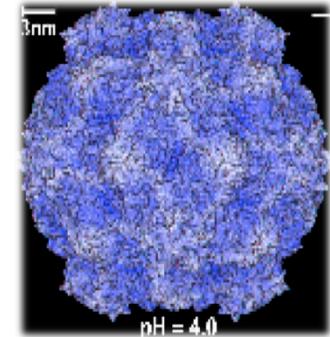


Example of a Computational Dwarf: N-Body

- Computational Dwarf: *Pattern of computation & communication ... that is common across a set of applications*
- N-Body problems are studied in
 - Cosmology, particle physics, biology, and engineering
- All have similar structures
- An N-Body benchmark can provide meaningful insight to people in all these fields
- Optimizations may be generally applicable as well



RoadRunner Universe:
Astrophysics



GEM:
Molecular Modeling

First Instantiation: OpenDwarfs

(formerly “OpenCL and the 13 Dwarfs”)

- Goal
 - Provide common algorithmic methods, i.e., dwarfs, in a language that is “write once, run anywhere” (CPU, GPU, or even FPGA), i.e., OpenCL

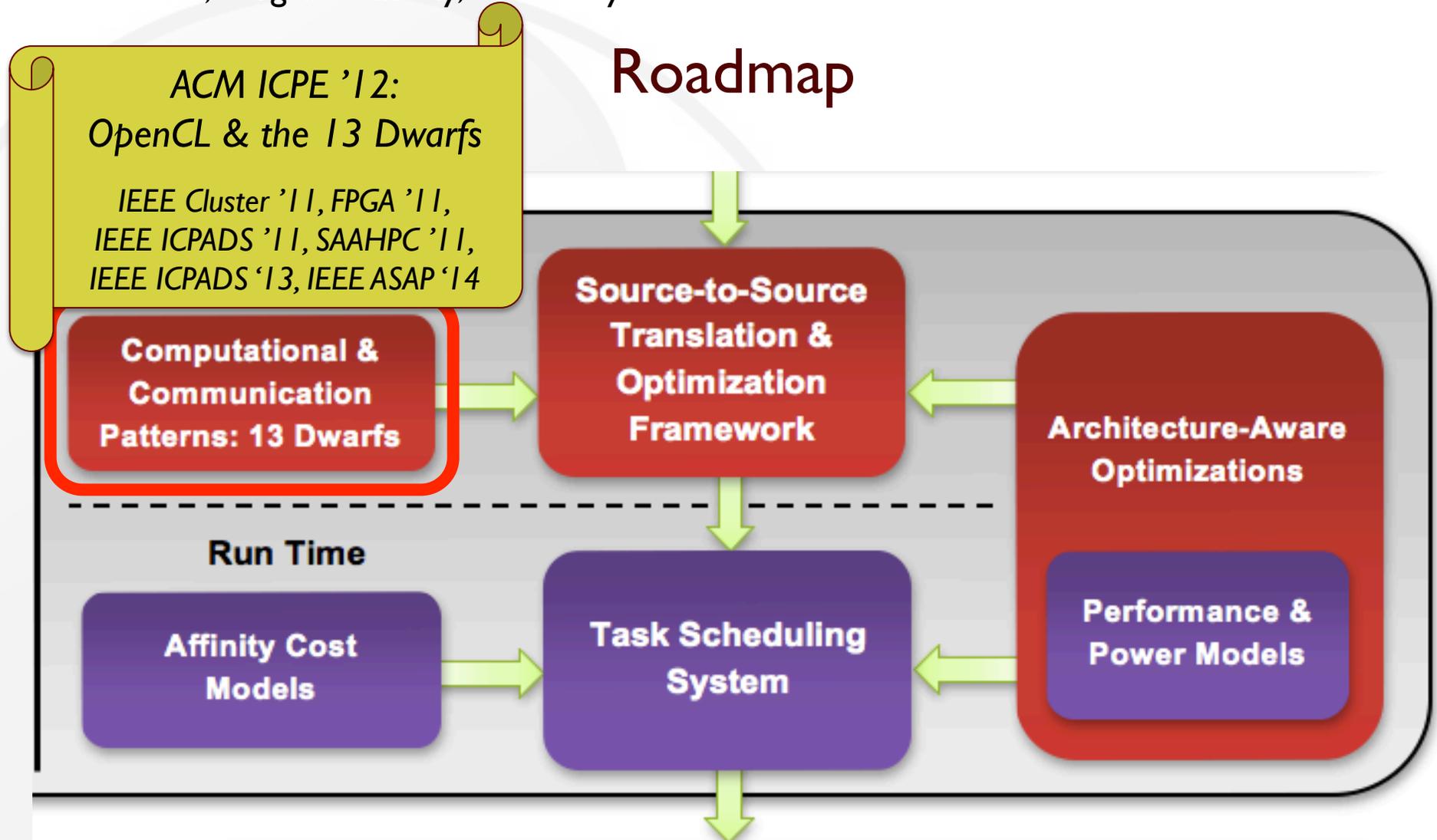


- Part of a larger umbrella project (2008-2018), funded by the NSF Center for High-Performance Reconfigurable Computing (CHREC)



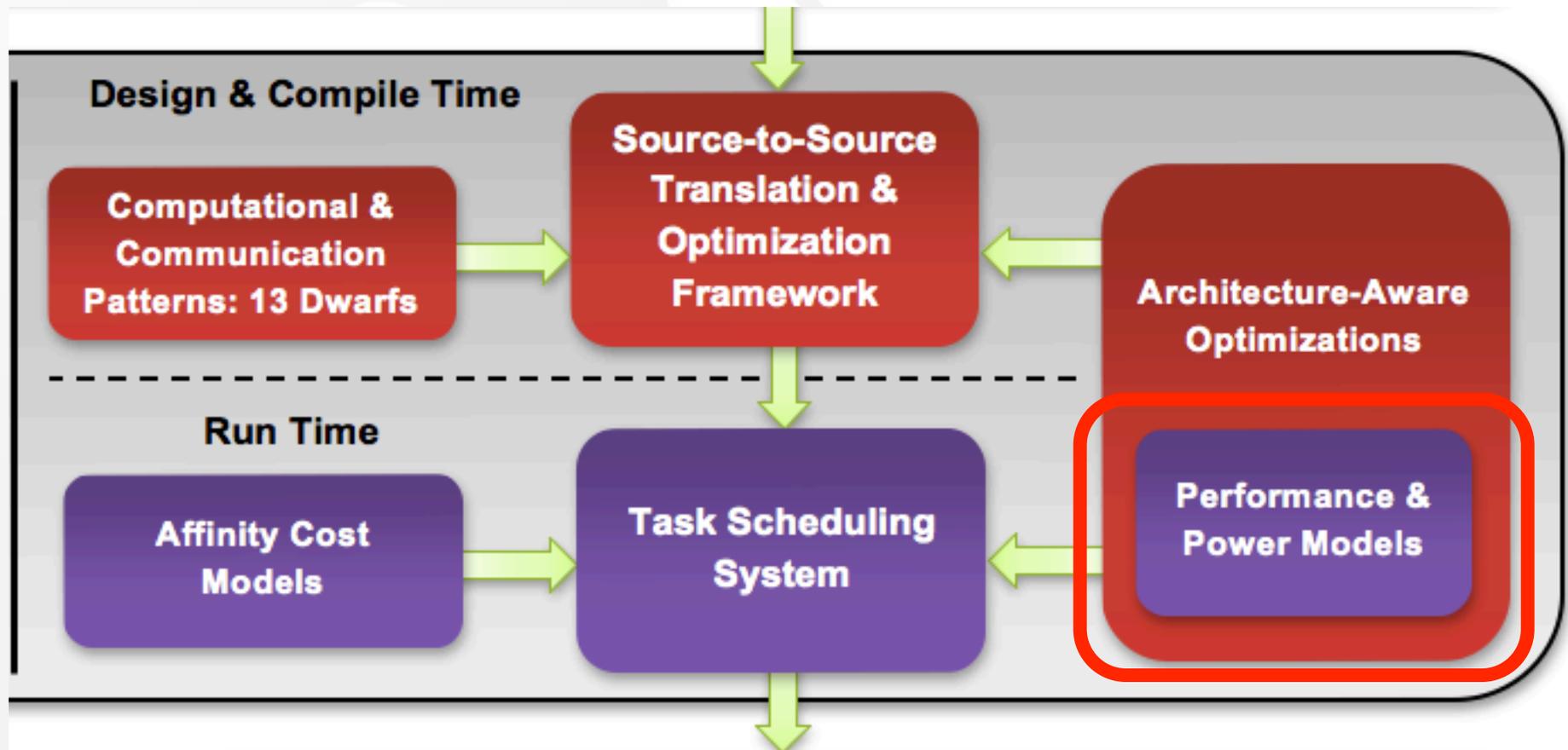
Performance, Programmability, Portability

Roadmap



Performance, Programmability, Portability

Roadmap



Performance & Power Modeling

- **Goals**
 - Robust framework
 - Very high accuracy (Target: < 5% prediction error)
 - Identification of portable predictors for performance and power
 - Multi-dimensional characterization
 - Performance → sequential, intra-node parallel, inter-node parallel
 - Power → component level, node level, cluster level

Problem Formulation:

LP-Based Energy-Optimal DVFS Schedule

- Definitions
 - A DVFS system exports n $\{ (f_i, P_i) \}$ settings.
 - T_i : total execution time of a program running at setting i
- Given a program with deadline D , find a DVS schedule (t_1^*, \dots, t_n^*) such that
 - If the program is executed for t_i seconds at setting i , the total energy usage E is minimized, the deadline D is met, and the required work is completed.

$$\min E = \sum_i P_i \cdot t_i$$

subject to

$$\begin{aligned}\sum_i t_i &\leq D \\ \sum_i t_i / T_i &= 1 \\ t_i &\geq 0\end{aligned}$$

Single-Coefficient β Performance Model

- Our Formulation

- Define the relative performance slowdown δ as

$$T(f) / T(f_{MAX}) - 1$$

- Re-formulate two-coefficient model as a single-coefficient model:

$$\frac{T(f)}{T(f_{max})} = \beta \cdot \frac{f_{max}}{f} + (1 - \beta)$$

with

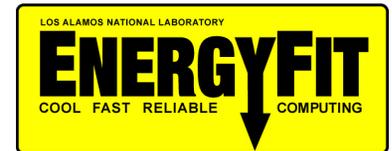
$$\beta = \frac{W_{cpu}}{W_{cpu} + T_{mem} \cdot f_{max}}$$

- The coefficient β is computed at run-time using a regression method on the past MIPS rates reported from the built-in PMU.

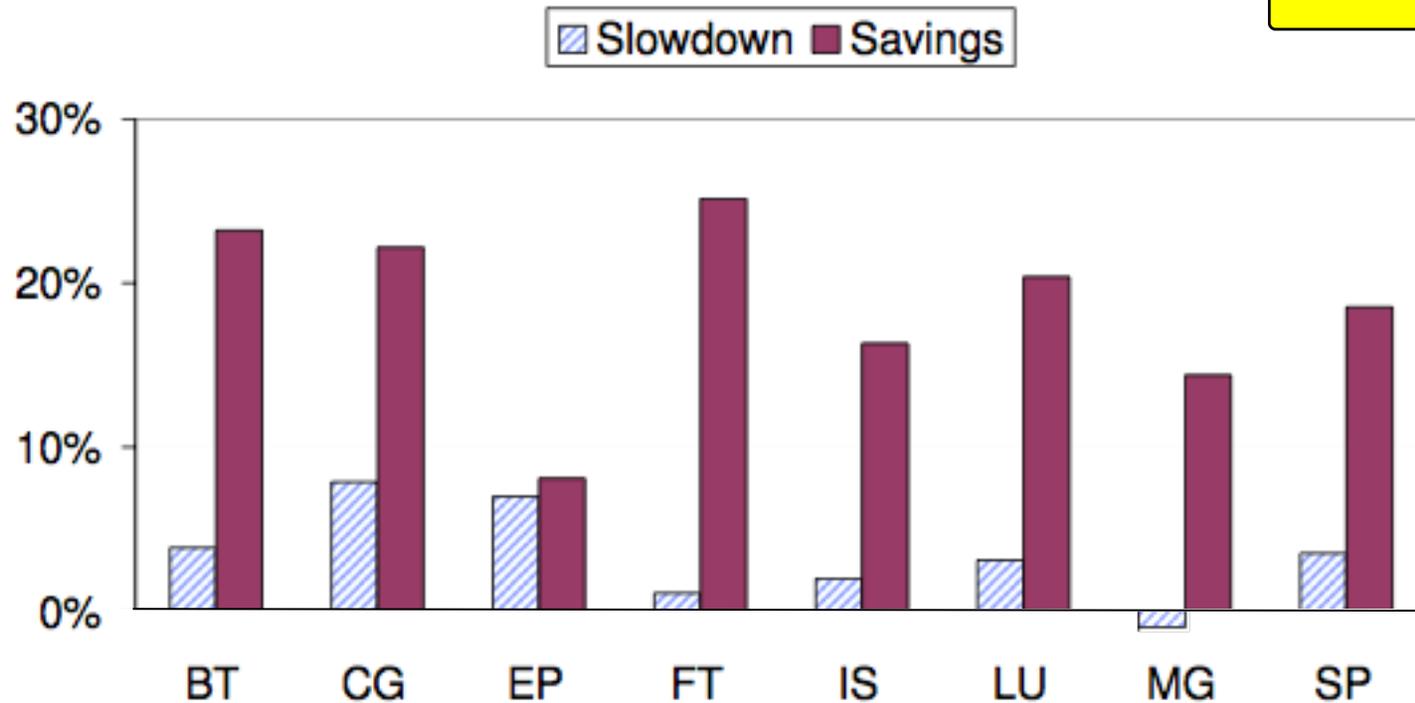
$$\beta = \frac{\sum_i \left(\frac{f_{max}}{f_i} - 1 \right) \left(\frac{\text{mips}(f_{max})}{\text{mips}(f_i)} - 1 \right)}{\sum_i \left(\frac{f_{max}}{f_i} - 1 \right)^2}$$

C. Hsu and W. Feng.
“A Power-Aware Run-Time System for High-Performance Computing,” *SC/05*, Nov. 2005.

β – Adaptation on NAS Parallel Benchmarks



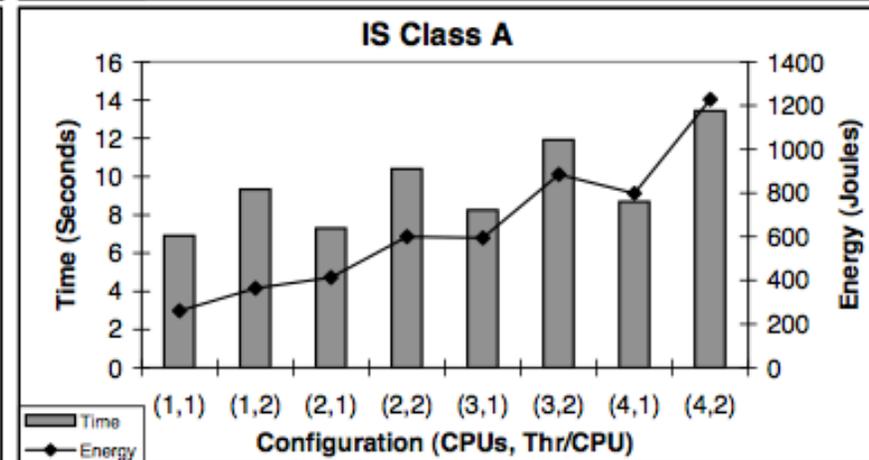
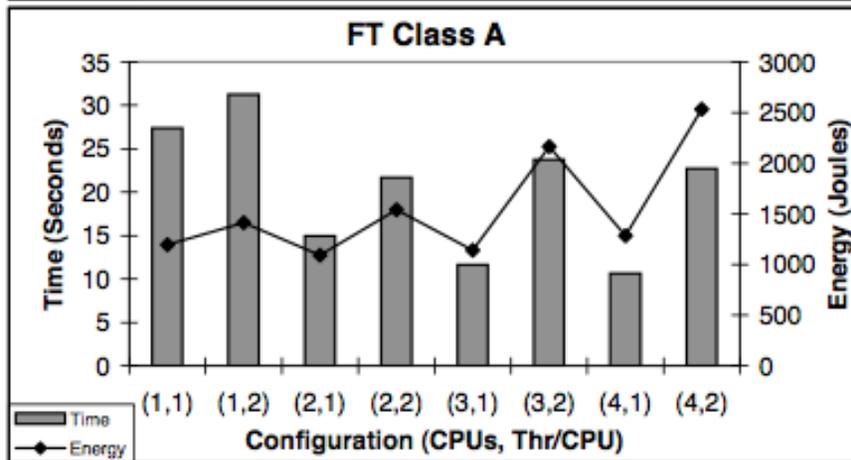
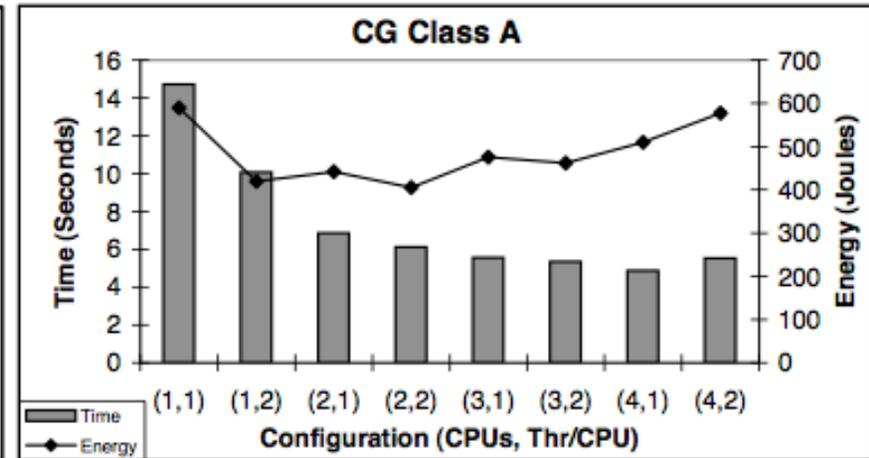
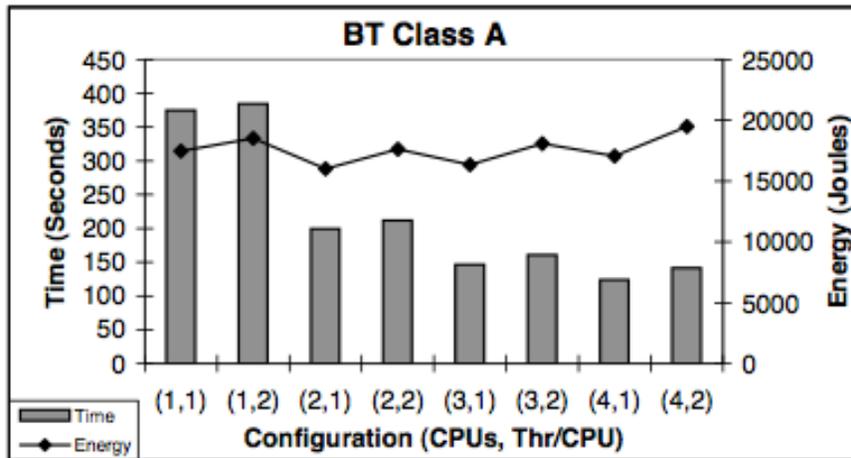
NAS/NPB3.2-MPI, C.16



C. Hsu and W. Feng.
“A Power-Aware Run-Time System
for High-Performance Computing,”
SC|05, Nov. 2005.

Energy reduction (15%)
Performance improvement ↗

Need for Better Performance & Power Modeling



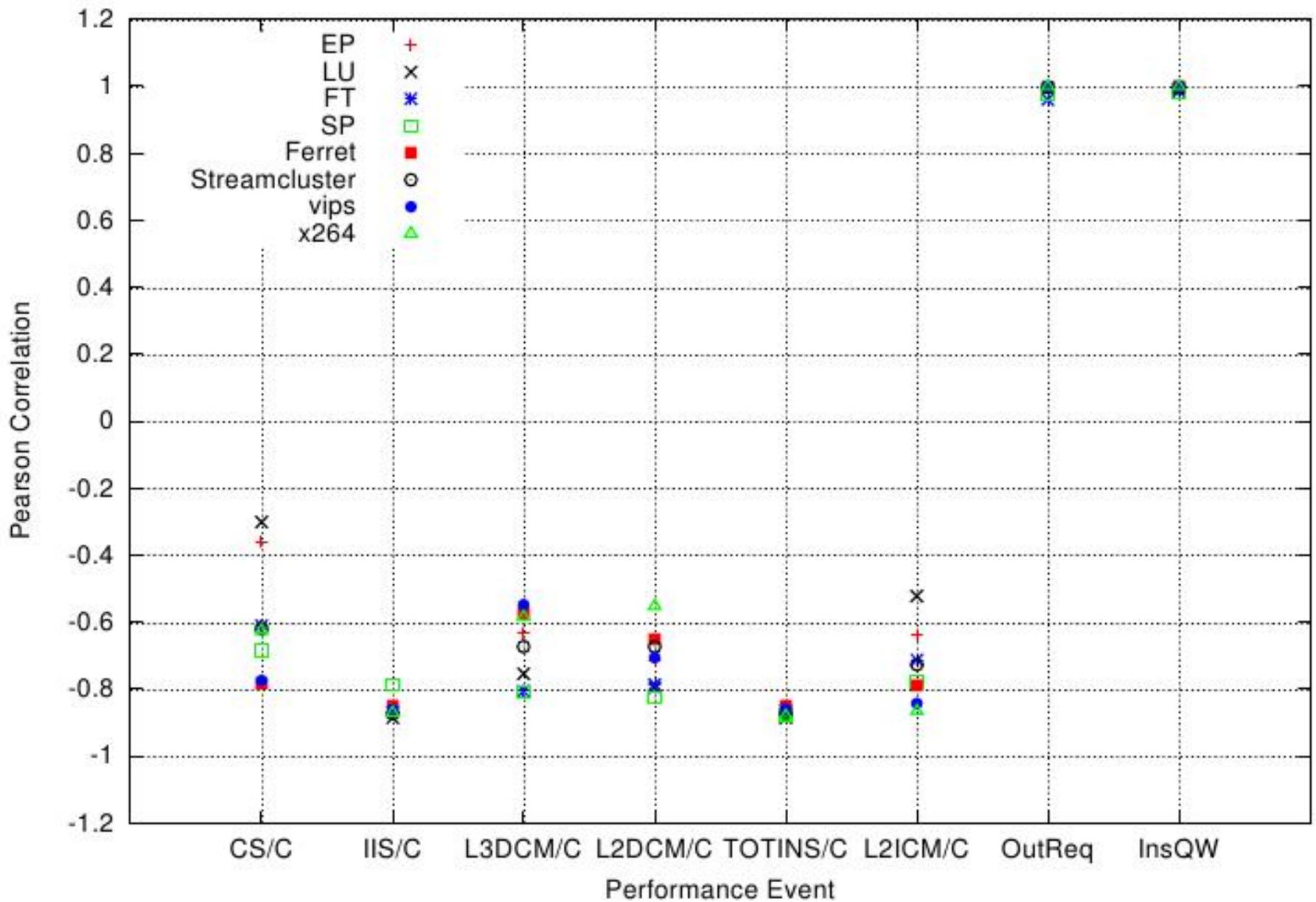
Source: Virginia Tech

Search for Reliable Predictors

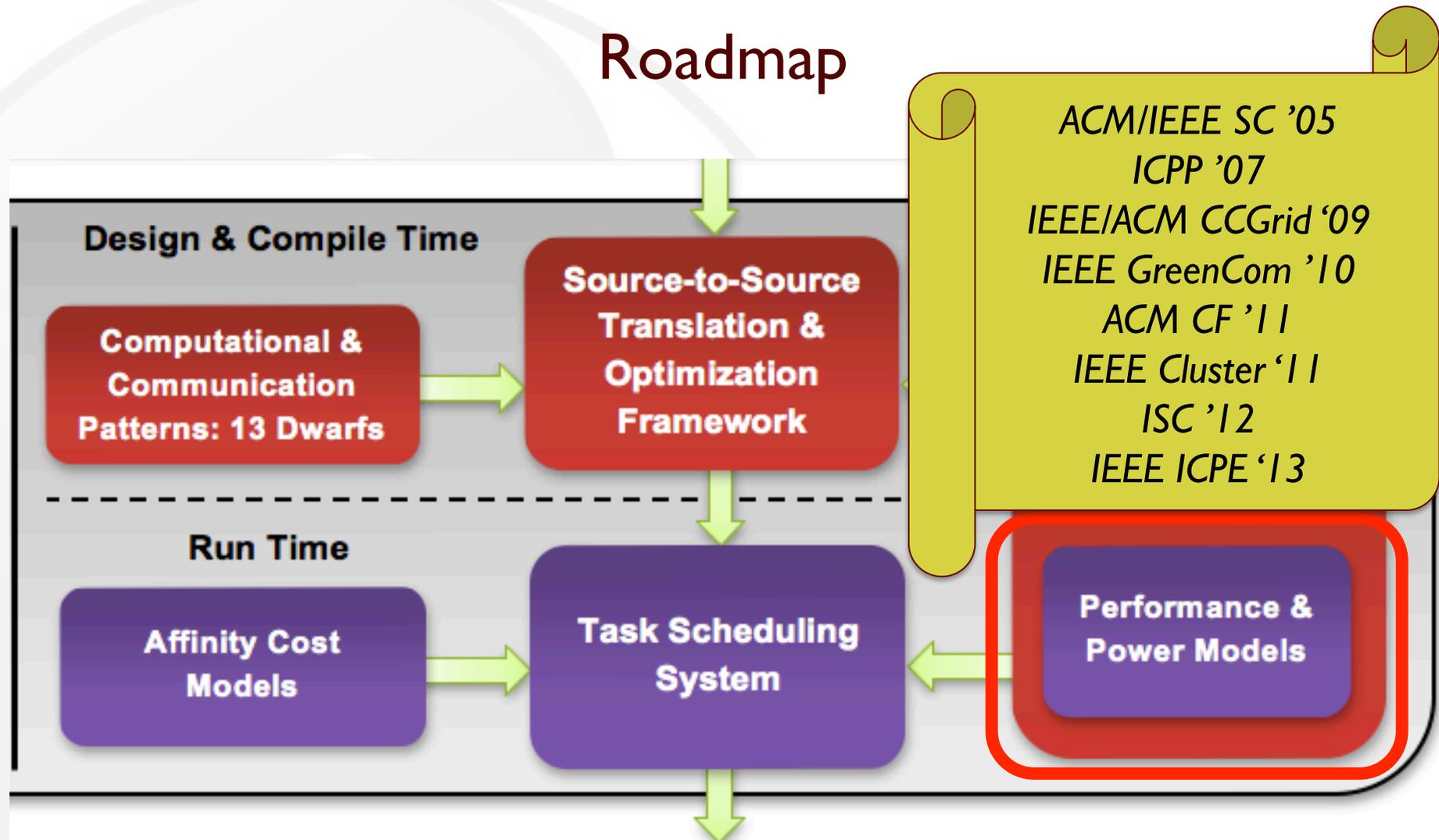
- Re-visit performance counters used for prediction
 - Applicability of performance counters across generations of architecture
- Performance counters monitored
 - NPB and PARSEC benchmarks
 - Platform: Intel Xeon E5645 CPU (Westmere-EP)

Context switches (CS)	Instruction issued (IIS)
L3 data cache misses (L3 DCM)	L2 data cache misses (L2 DCM)
L2 instruction cache misses (L2 ICM)	Instruction completed (TOTINS)
Outstanding bus request cycles (OutReq)	Instruction queue write cycles (InsQW)

Pearson Correlation For Performance (Execution Time) and Performance Event



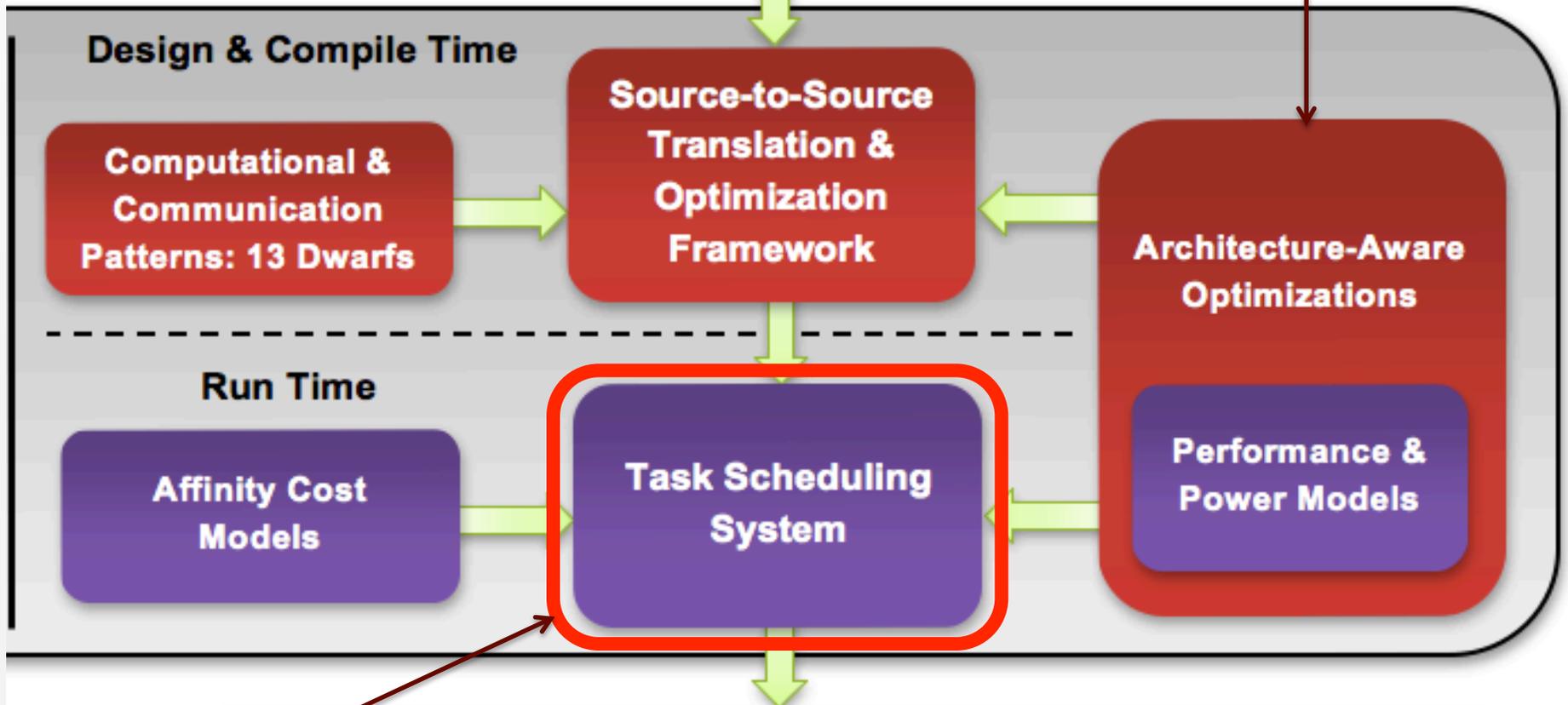
Roadmap



Performance, Programmability, Portability

Roadmap

Performance Portability



Performance Portability

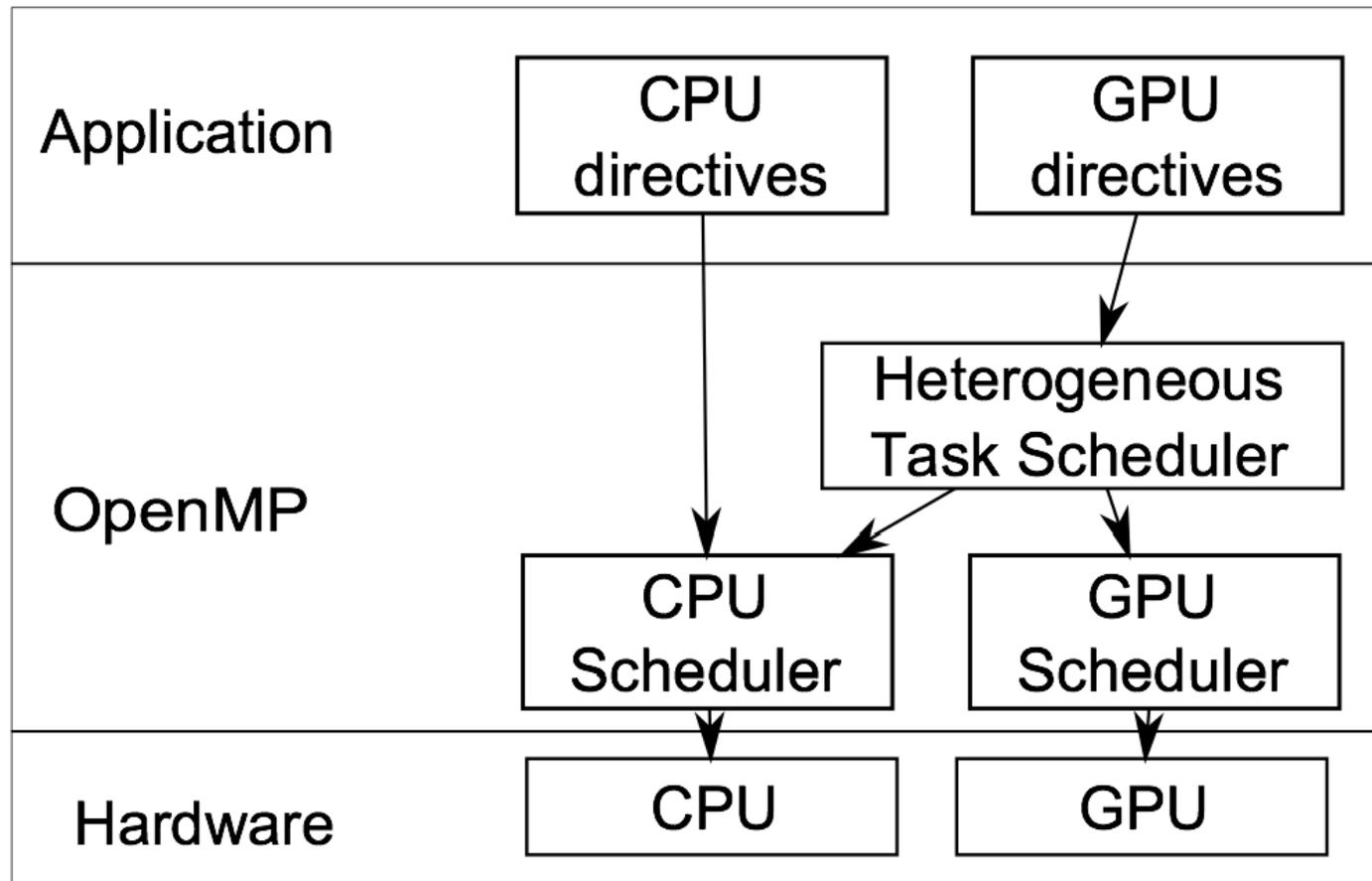
What is Heterogeneous Task Scheduling?

- Automatically spreading tasks across heterogeneous compute resources
 - CPUs, GPUs, APUs, FPGAs, DSPs, and so on
- Specify tasks at a higher level (currently OpenMP extensions)
- Run them across available resources automatically

Goal

- A run-time system that intelligently uses what is available resource-wise and optimize for performance portability
 - Each user should *not* have to implement this for themselves!

Heterogeneous Task Scheduling



How to Heterogeneous Task Schedule (HTS)

- Accelerated OpenMP offers heterogeneous task scheduling with
 - Programmability
 - Functional portability, given underlying compilers
 - Performance portability
- How?
 - A simple extension to Accelerated OpenMP syntax for **programmability**
 - Automatically dividing parallel tasks across arbitrary heterogeneous compute resources for **functional portability**
 - CPUs
 - GPUs
 - APUs
 - Intelligent runtime task scheduling for **performance portability**

Programmability: Why Accelerated OpenMP?

```
#pragma omp parallel for          \  
    shared(in1,in2,out,pow)  
for (i=0; i<end; i++){  
    out[i] = in1[i]*in2[i];  
    pow[i] = pow[i]*pow[i];  
}
```

Traditional OpenMP

```
#pragma acc_region_loop          \  
    acc_copyin(in1[0:end],in2[0:end])\  
    acc_copyout(out[0:end])      \  
    acc_copy(pow[0:end])  
for (i=0; i<end; i++){  
    out[i] = in1[i] * in2[i];  
    pow[i] = pow[i]*pow[i];  
}
```

OpenMP Accelerator Directives

```
#pragma acc_region_loop          \  
    acc_copyin(in1[0:end],in2[0:end]) \  
    acc_copyout(out[0:end])        \  
    acc_copy(pow[0:end])           \  
    hetero(<cond>[,<scheduler>[,<ratio>\  
            [,<div>]]])  
for (i=0; i<end; i++){  
    out[i] = in1[i] * in2[i];  
    pow[i] = pow[i]*pow[i];  
}
```

Our Proposed Extension

Programmability: Code Transformation

- Manual
 - Add 20 lines
 - Must manually split problem and reassemble results
- Automatic
 - Add I clause
 - Splitting and assembly are automatic

```
#pragma omp acc_region_loop private(i) \
    firstprivate(nco,no,ncl) default(none) \
    acc_copyin(fc[0:ncl*nco]) present(fo) \
    acc_copyout(m[0:no])
// hetero(1,dynamic)
for (i=0; i<no; i++) {
    m[i] = findc(no,ncl,nco,fo,fc,i);
}
```

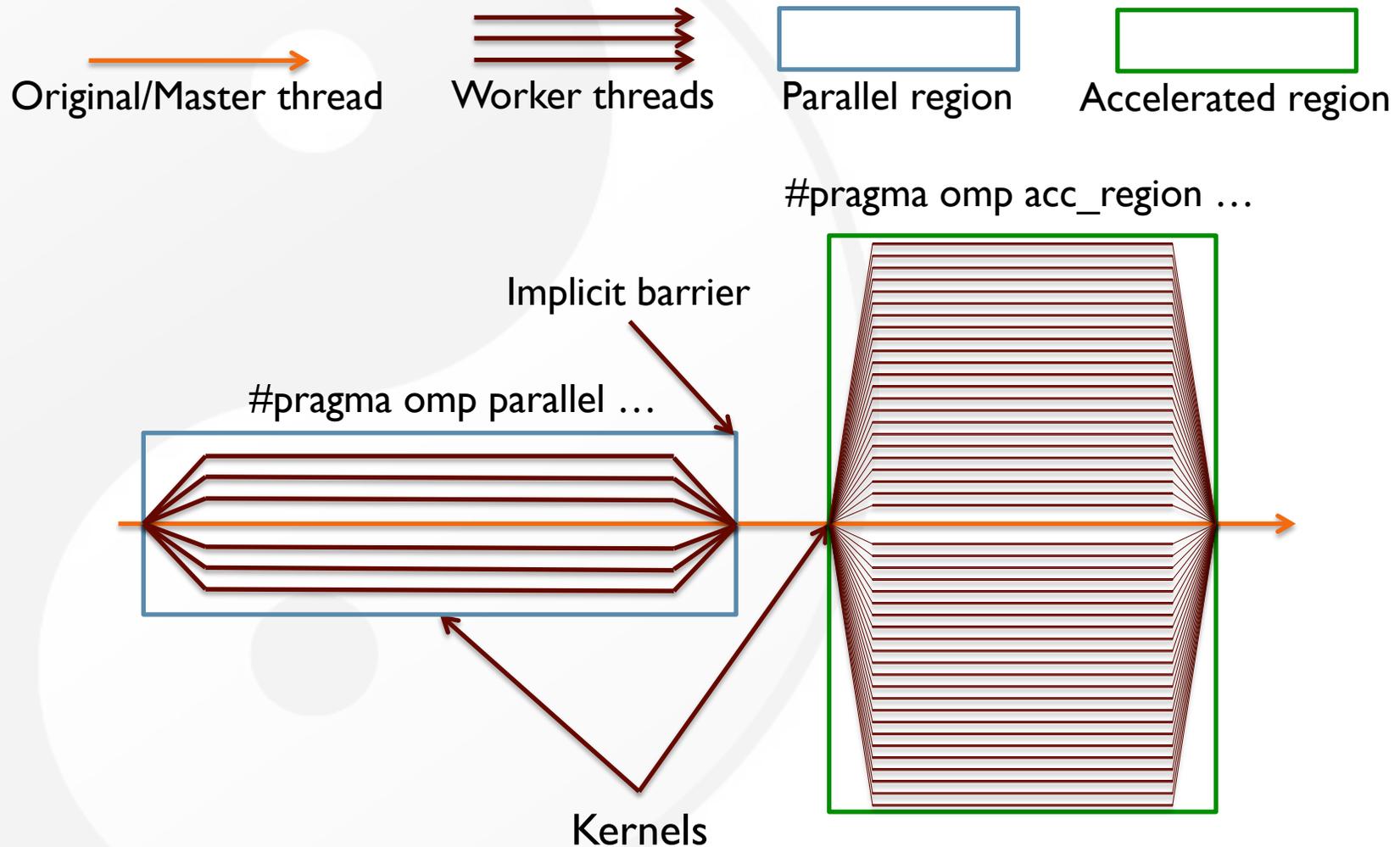
Our Proposed Extension

```
splitter * s = split_init(no, SPLIT_DYNAMIC, NULL, NULL);
int *m_c = (int*)malloc(sizeof(int)*no);
for(int d_it=0; d_it < s->d_end; d_it++)
{
    s = split_next(no, d_it);

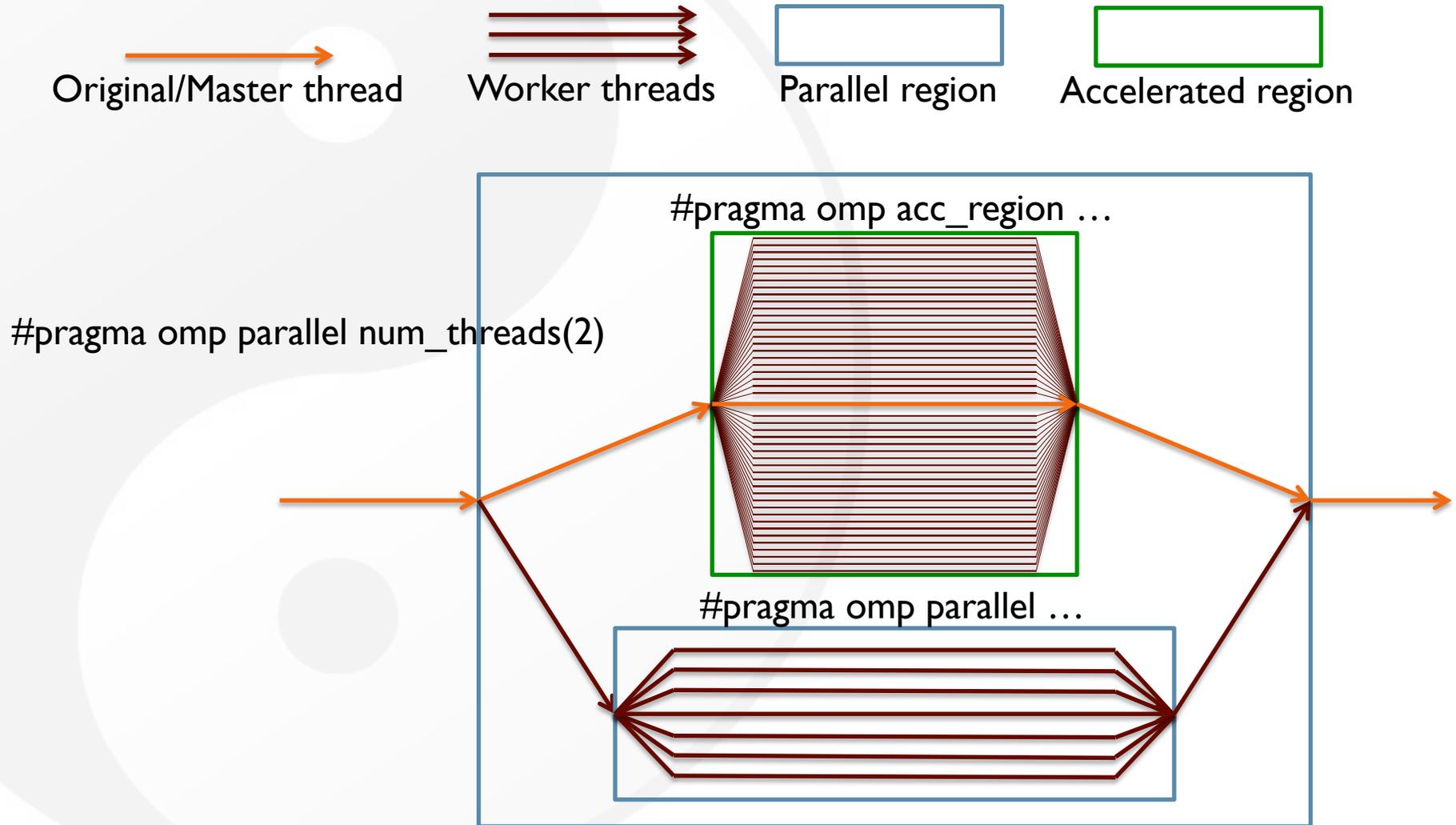
    #pragma omp parallel num_threads(2)
    {
        if(omp_get_thread_num()>0)
            { //CPU OpenMP code
                split_cpu_start(s);
                #pragma omp parallel shared(fo,fc,m_c,s) \
                    num_threads(omp_get_thread_limit()-1) \
                    firstprivate(no,ncl,nco) private(i)
                {
                    #pragma omp for
                    for (i=s->cts; i<s->cte; i++) {
                        m_c[i] = findc(no,ncl,nco,fo,fc,i);
                    }
                }
                split_cpu_end(s);
            }else{ //GPU OpenMP code
                split_gpu_start(s);
                int gts = s->gts, gte = s->gte;
                #pragma omp acc_region_loop private(i) \
                    firstprivate(nco,no,ncl,gts,gte) \
                    acc_copyin(fc[0:ncl*nco]) \
                    acc_copyout(m[0:no]) \
                    present(fo) default(none)
                for (i=gts; i<gte; i++) {
                    m[i] = findc(no,ncl,nco,fo,fc,i);
                }
                split_gpu_end(s);
            }
    }
}
memcpy(m+s->d_ccs,m_c+s->d_ccs,
    (s->d_cce-s->d_ccs)*sizeof(int));
free(m_c);
```

Manual

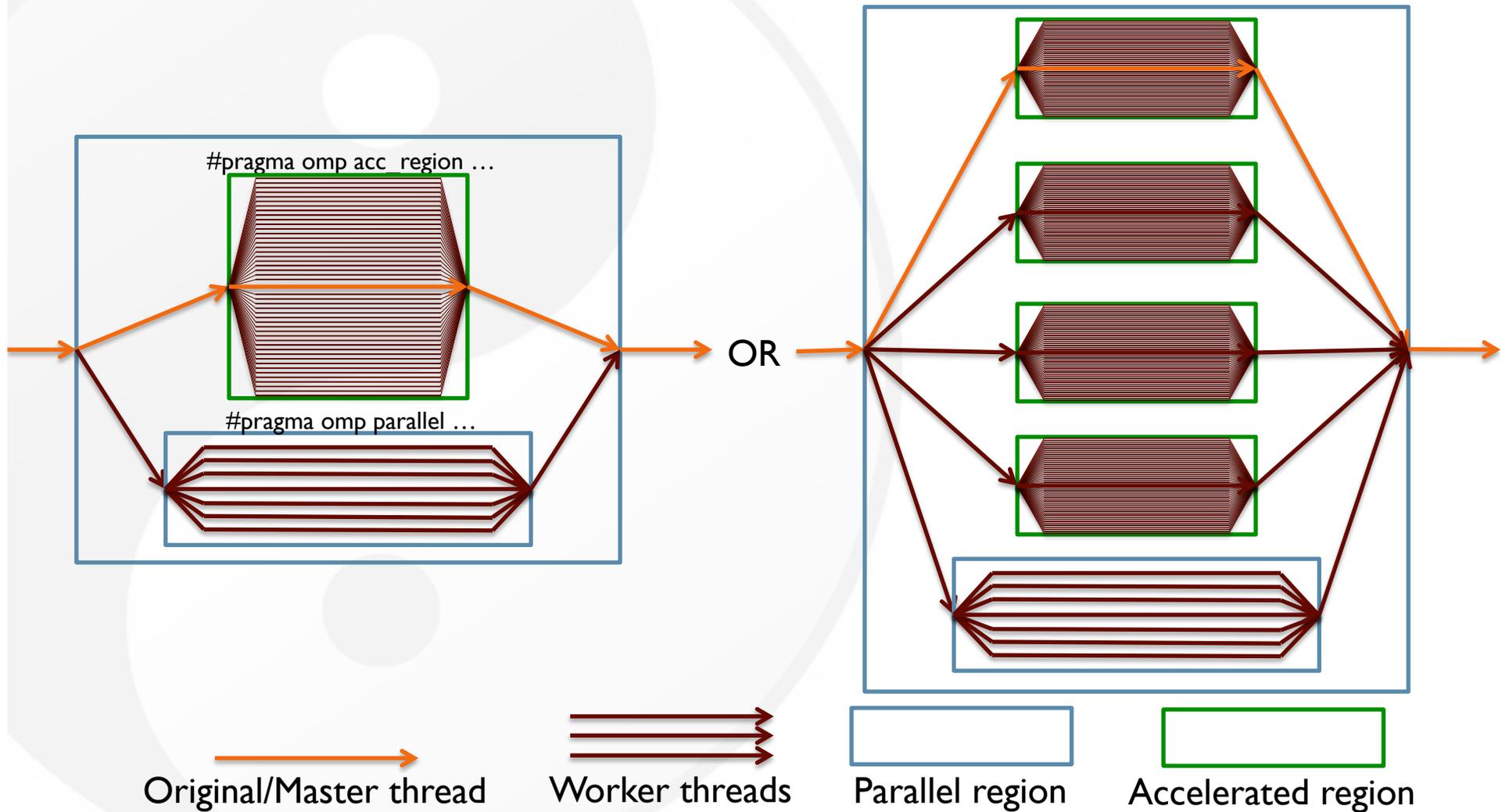
OpenMP Accelerator Behavior



DESIRED OpenMP Accelerator Behavior



Work-share a Region Across the Whole System



CoreTSAR: Scheduling and Load-Balancing by Adaptation

- Measure computational suitability at runtime
- Compute new distribution of work through a linear optimization approach
- Re-distribute work before each pass

I = total iterations available

i_j = iterations for compute unit j

f_j = fraction of iterations for compute unit j

p_j = recent time/iteration for compute unit j

n = number of compute devices

t_j^+ (or t_j^-) = time over (or under) equal

$$\min\left(\sum_{j=1}^{n-1} t_j^+ + t_j^-\right) \quad (7)$$

$$\sum_{j=1}^n f_j = 1 \quad (8)$$

$$f_2 * p_2 - f_1 * p_1 = t_1^+ - t_1^- \quad (9)$$

$$f_3 * p_3 - f_1 * p_1 = t_2^+ - t_2^- \quad (10)$$

⋮

$$f_n * p_n - f_1 * p_1 = t_{n-1}^+ - t_{n-1}^- \quad (11)$$

Heterogeneous Scheduling: Issues and Solutions

- Issue: Launching overhead is high on GPUs
 - Using a work-queue, many GPU kernels may need to be run
- Solution: Schedule only at the beginning of a region
 - The overhead is only paid once or a small number of times
- Issue: Without a work-queue, how do we balance load?
 - The performance of each device must be predicted
- Solution: Allocate different amounts of work
 - For the first pass, predict a reasonable initial division of work. We use a ratio between the number of CPU and GPU cores for this.
 - For subsequent passes, use the performance of previous passes to predict following passes.

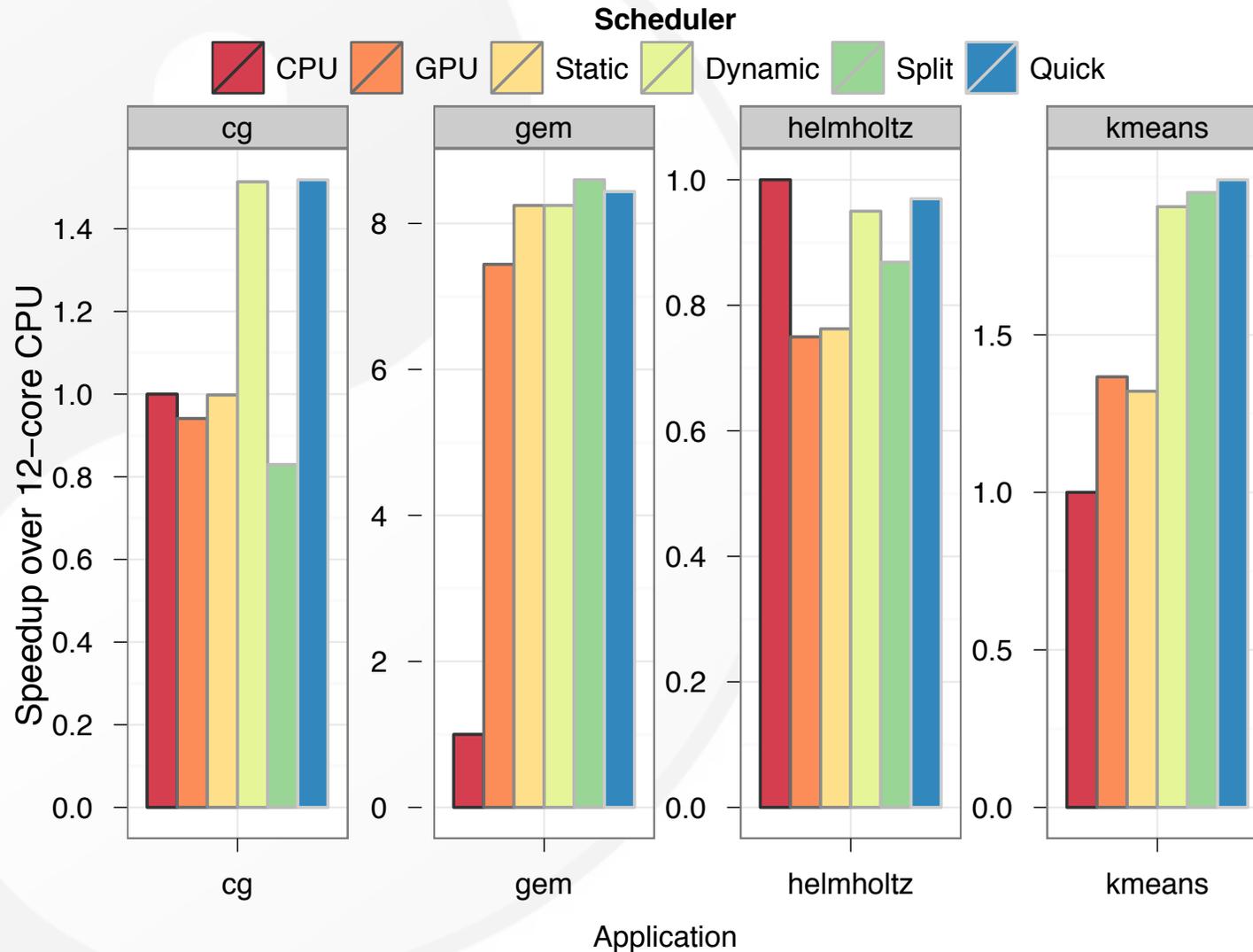
Benchmarks

- **NAS CG – Many passes (1,800 for C class)**
 - The Conjugate Gradient benchmark from the NAS Parallel Benchmarks
- **GEM – One pass**
 - Molecular Modeling, computes the electrostatic potential along the surface of a macromolecule
- **K-Means – Few passes**
 - Iterative clustering of points
- **Helmholtz – Few passes, GPU unsuitable**
 - Jacobi iterative method implementing the Helmholtz equation

Experimental Setup

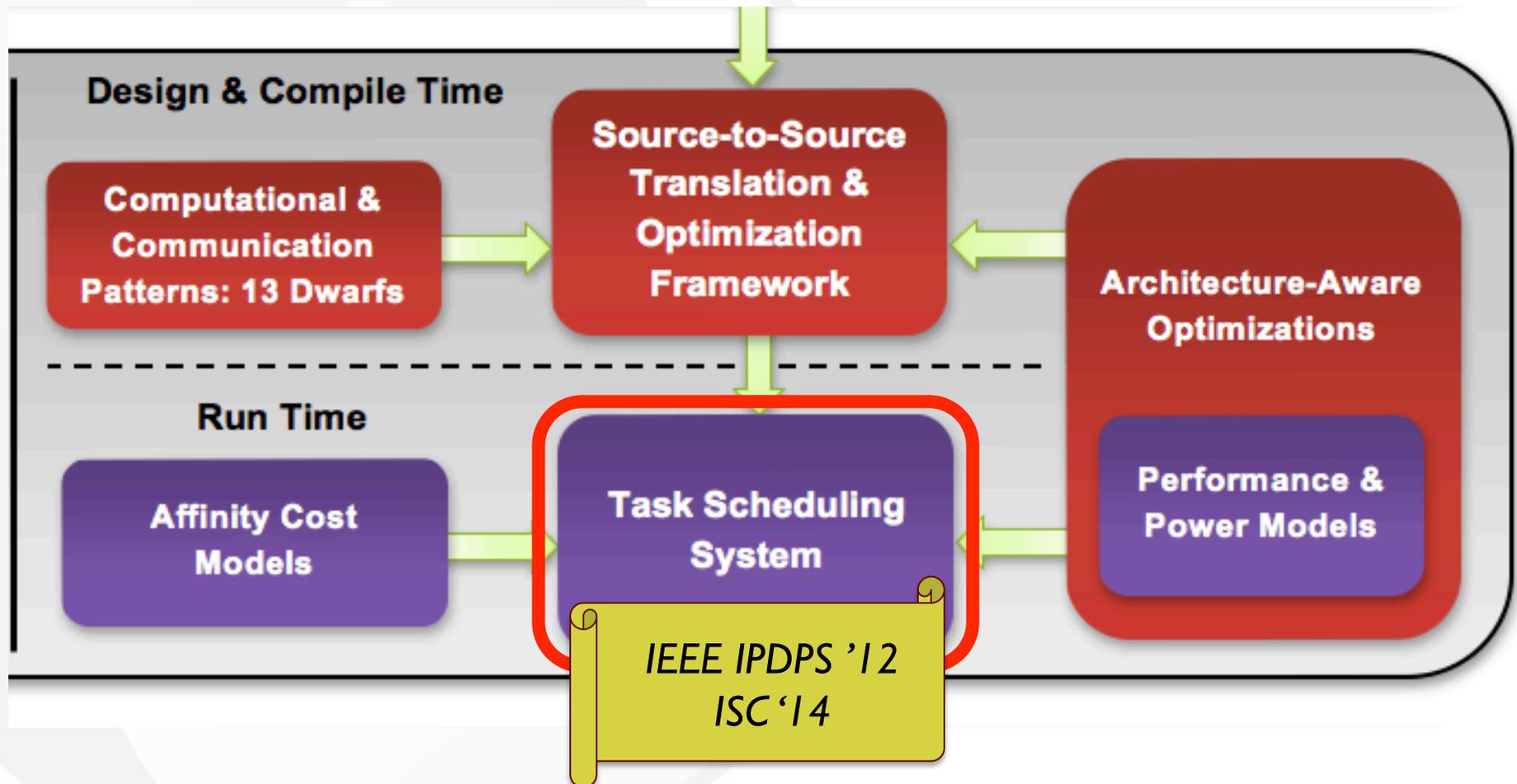
- **System**
 - 12-core AMD Opteron 6174 CPU
 - NVIDIA Tesla C2050 GPU
 - Linux 2.6.27.19
 - CCE compiler with OpenMP accelerator extensions
- **Procedures**
 - All parameters default, unless otherwise specified
 - Results represent 5 or more runs

CoreTSAR Results Across Schedulers

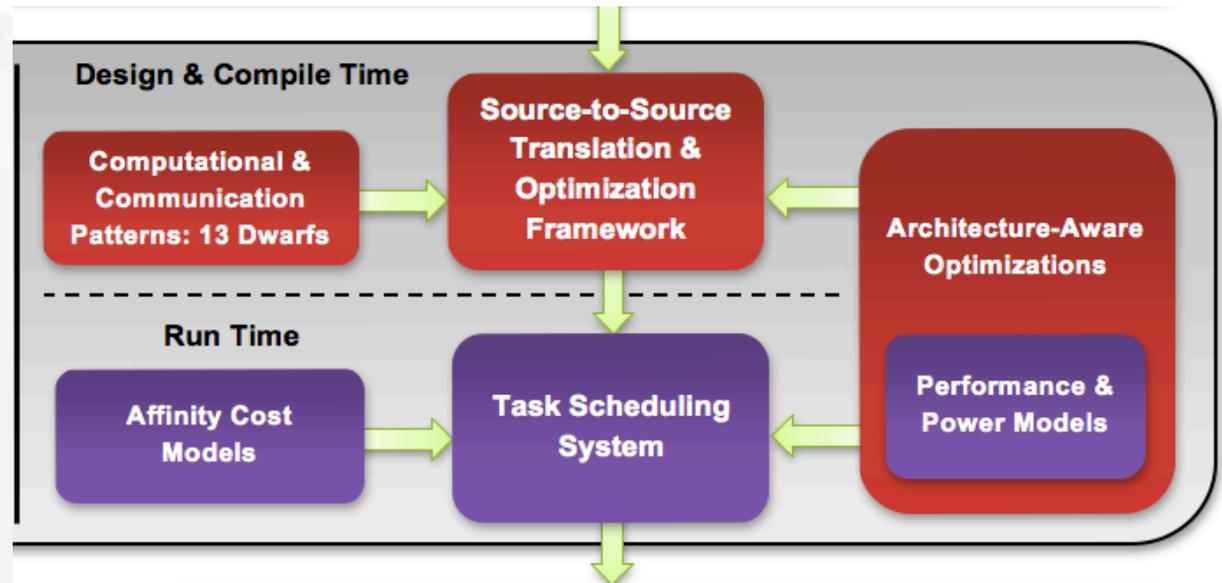


Performance, Programmability, Portability

Roadmap



Intra-Node
**An Ecosystem for
Heterogeneous
Parallel Computing**

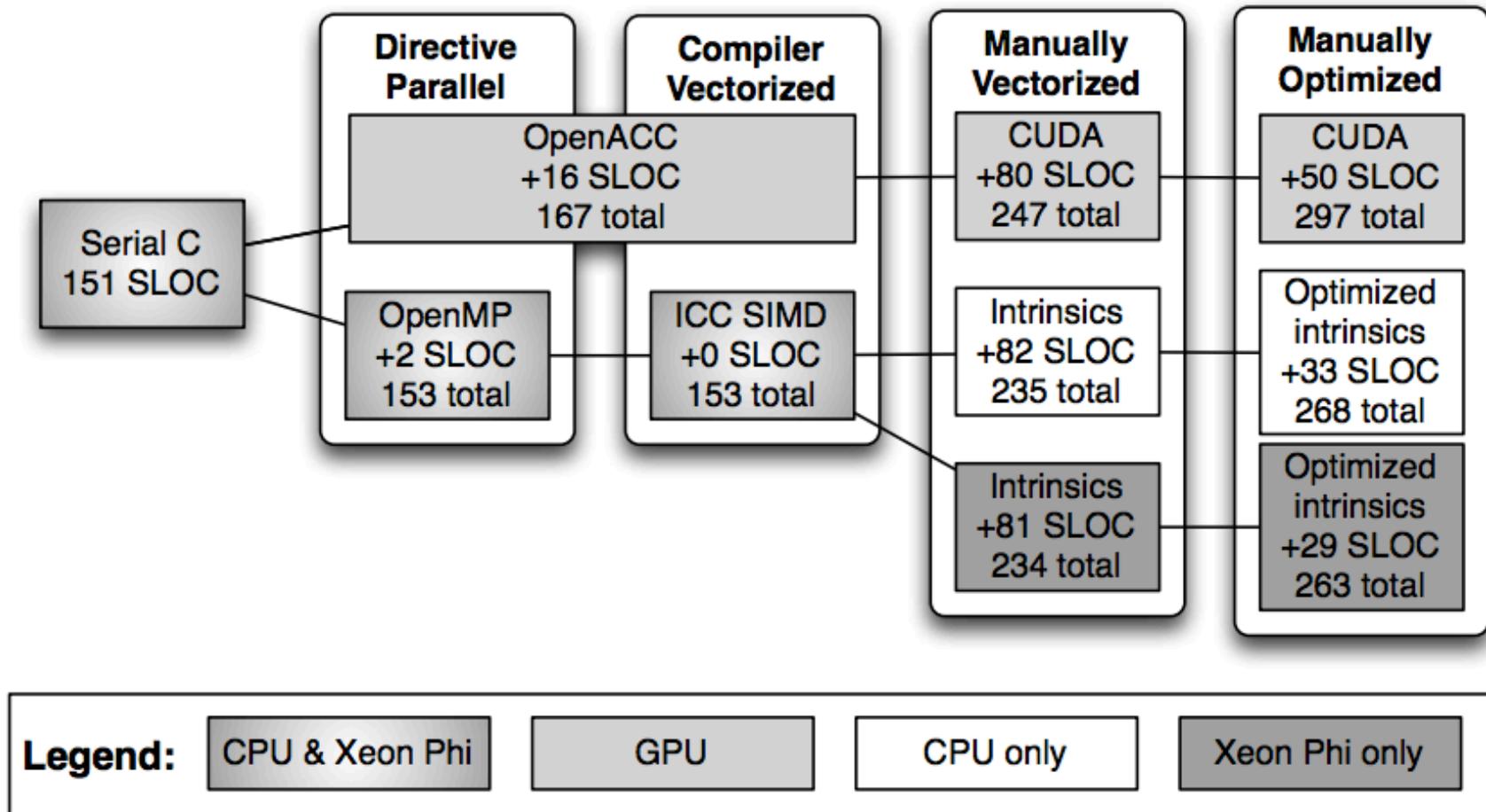


Much work still to be done.

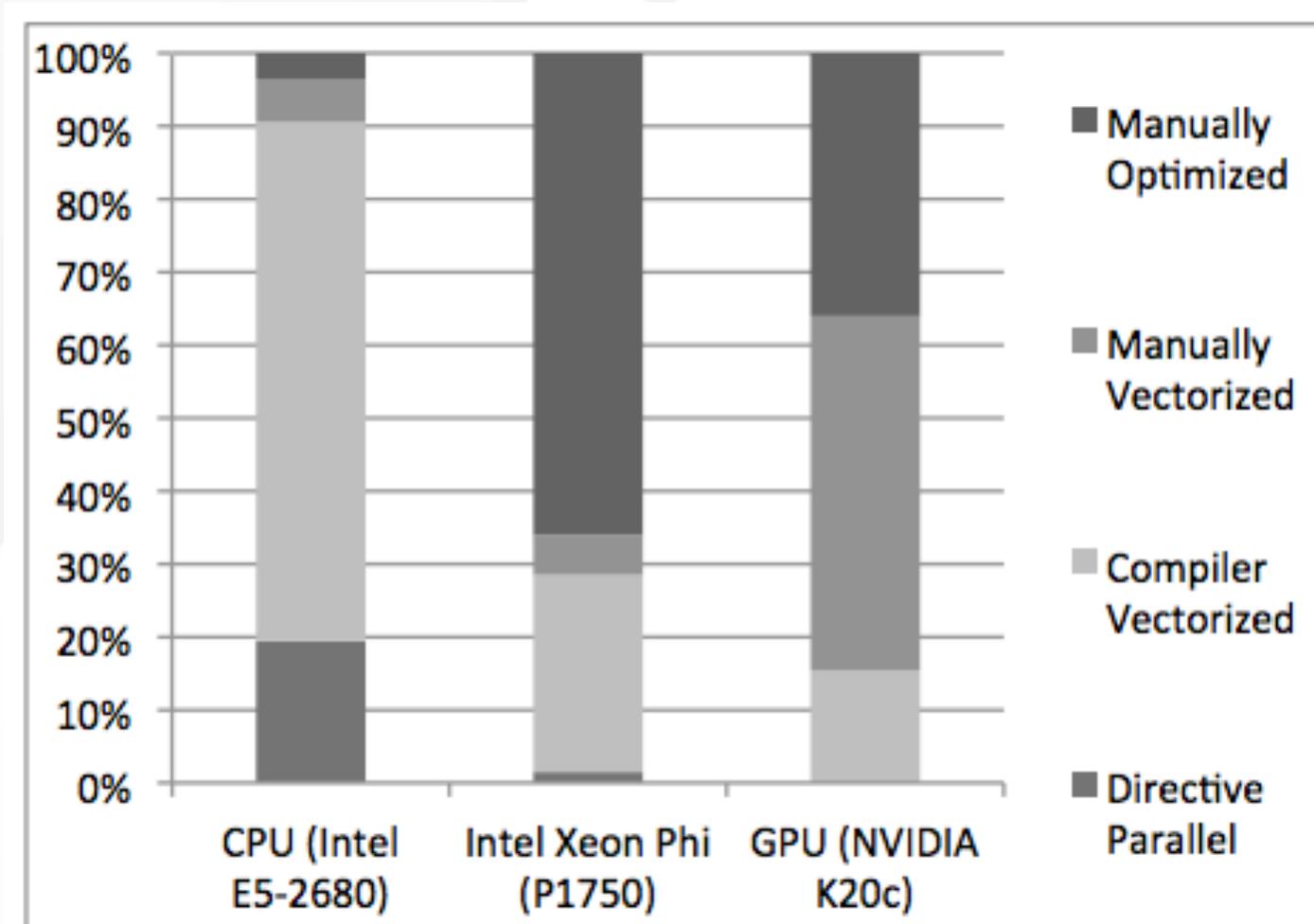
- OpenCL and the 13 Dwarfs
- Source-to-Source Translation
- Architecture-Aware Optimization
- Performance & Power Modeling
- Affinity-Based Cost Modeling
- Heterogeneous Task Scheduling

Beta release pending
CU2CL only & no optimization
Only manual optimizations
Preliminary & pre-multicore
Empirical results; modeling in progress
Preliminary with OpenMP

Performance, Programmability, and Portability (3 Ps)



Performance, Programmability, and Portability (3 Ps)

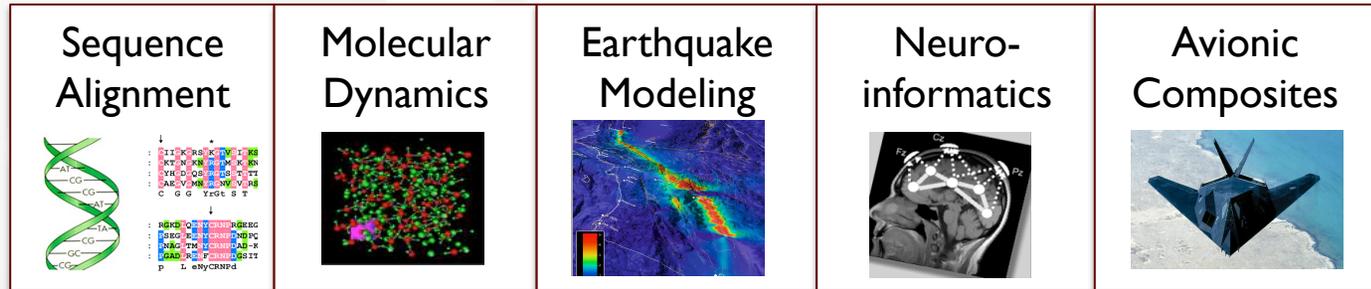


Intra-Node

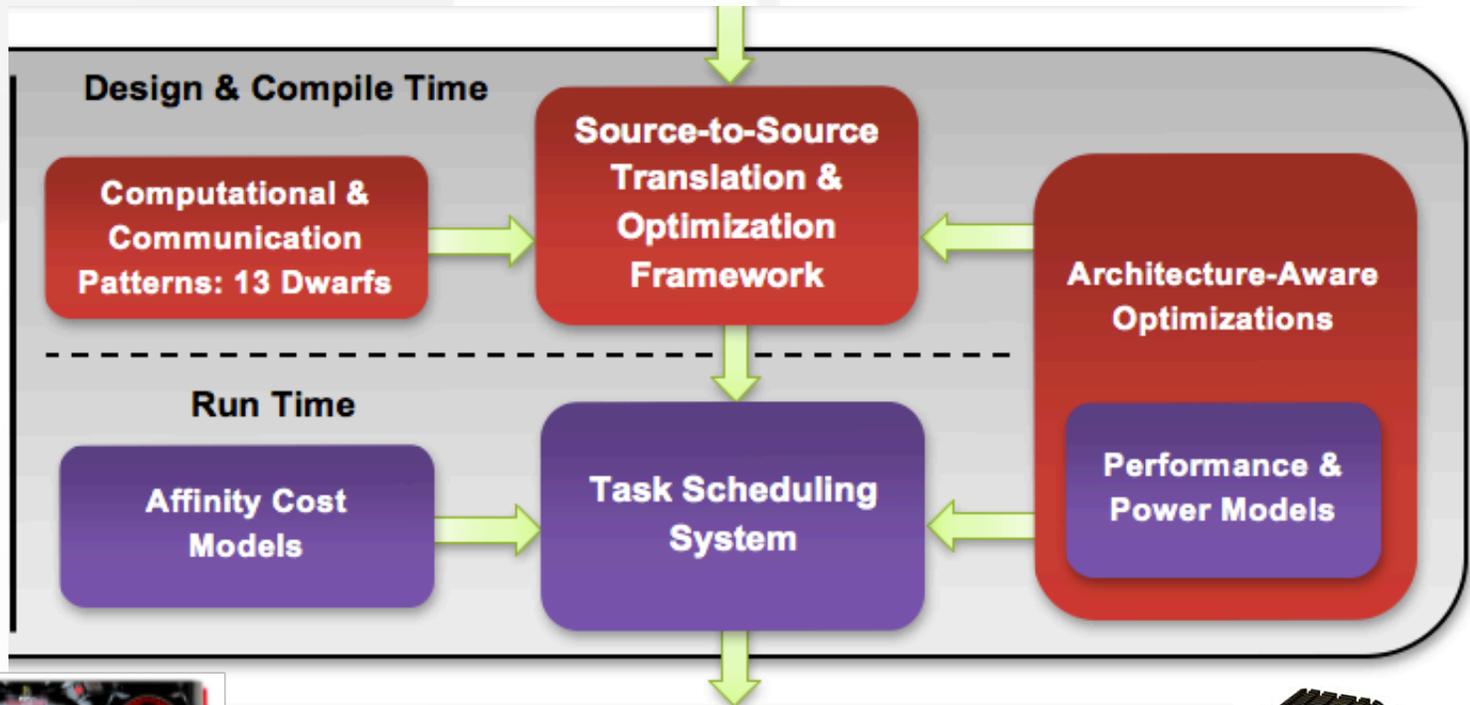
An Ecosystem for Heterogeneous Parallel Computing

^

Applications



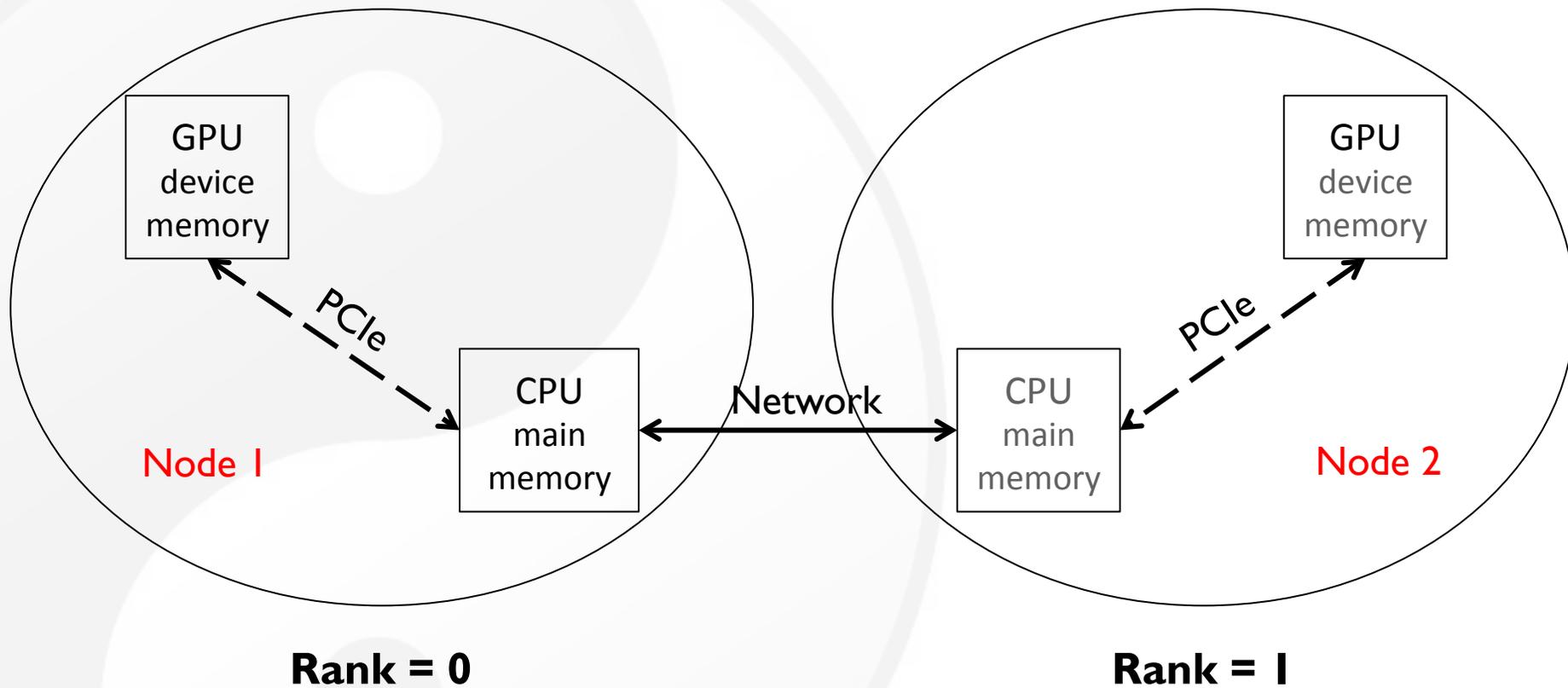
Software Ecosystem



Heterogeneous Parallel Computing (HPC) Platform



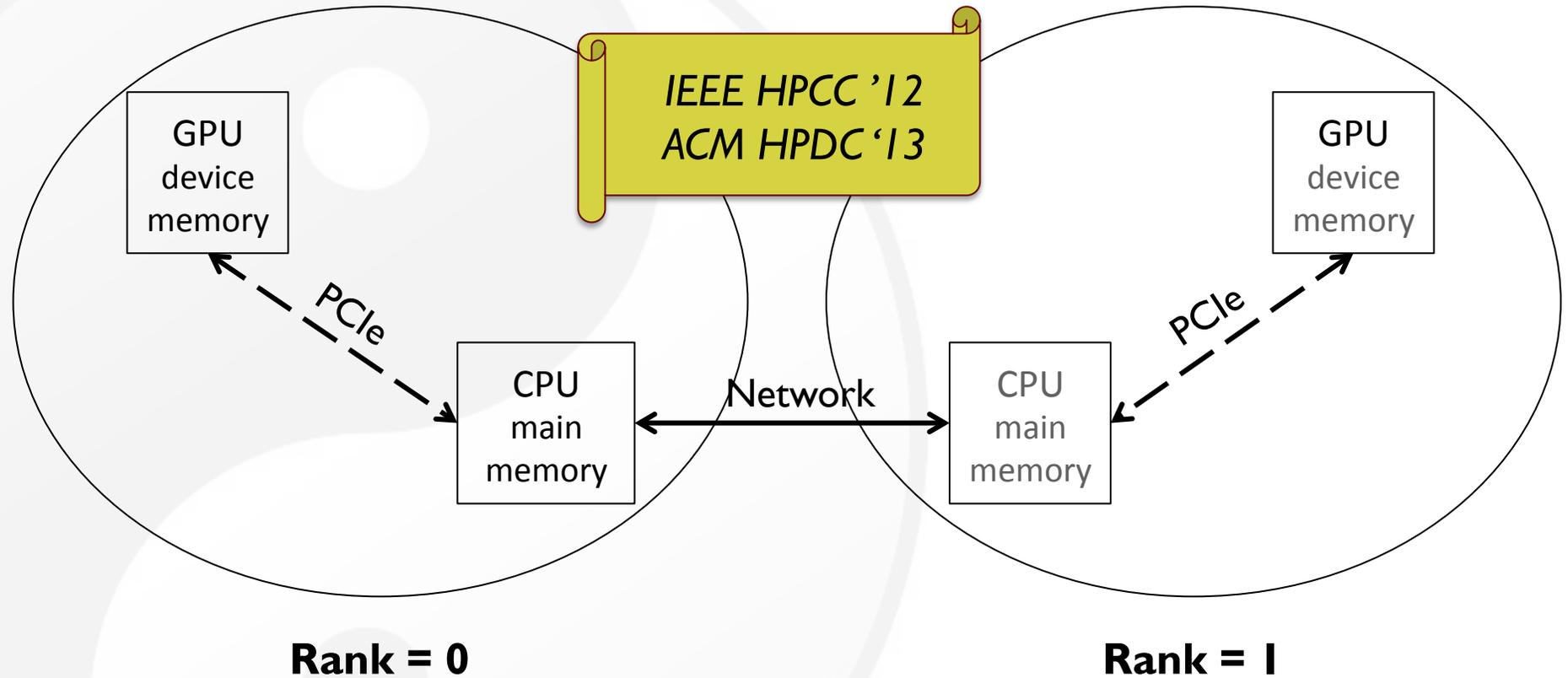
Programming CPU-GPU Clusters (e.g., MPI+CUDA)



```
if(rank == 0)
{
  cudaMemcpy(host_buf, dev_buf, D2H)
  MPI_Send(host_buf, .. ..)
}
```

```
if(rank == 1)
{
  MPI_Recv(host_buf, .. ..)
  cudaMemcpy(dev_buf, host_buf, H2D)
}
```

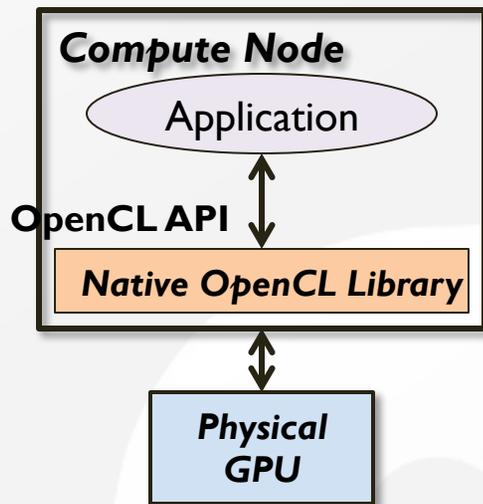
Goal of Programming CPU-GPU Clusters (MPI + Any Acc)



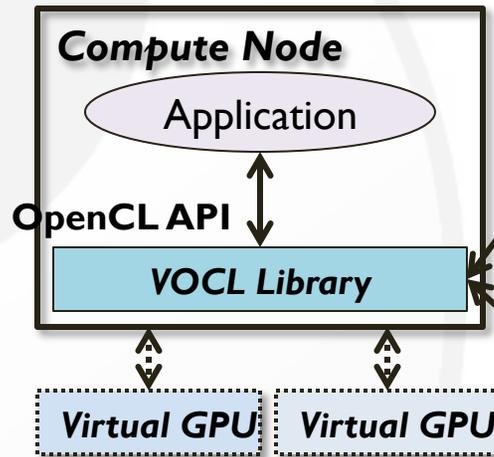
```
if(rank == 0)
{
  MPI_Send(any_buf, .. ..);
}
```

```
if(rank == 1)
{
  MPI_Recv(any_buf, .. ..);
}
```

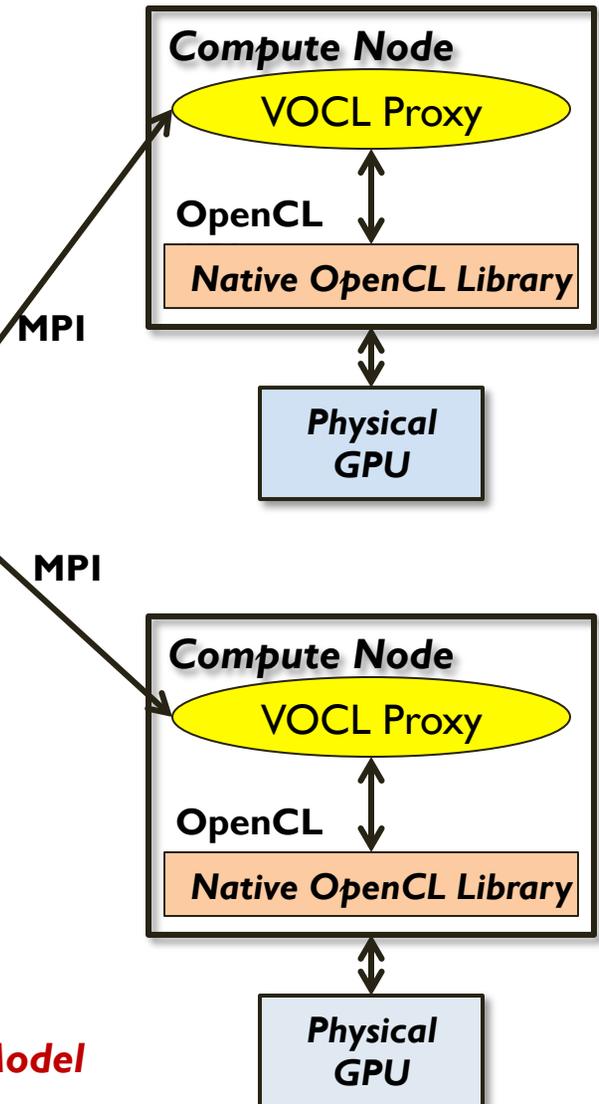
“Virtualizing” GPUs ...



Traditional Model



Our Model



Faculty & Staff



Wu-chun ("Wu") Feng

Dept. of CS, ECE, VBI at Virginia Tech and
Dept. of Cancer Biology & Translational Science
Inst. at Wake Forest U.



Paul Sathre

Postmasters Research
Associate,
Dept. of CS.



Nataliya E. Timoshevskaya

Postdoctoral Research
Associate,
Virginia Tech.



Mark K. Gardner

Office of IT and Affiliate Faculty
Dept. of CS.



Harold Trease

Sr. Research Scientist,
Dept. of CS.



Hao Wang

Postdoctoral Research Associate,
Dept. of CS.

Students



Ashwin Aji

(Ph.D.)
NVIDIA Graduate
Fellowship



Vignesh Adhinarayanan

(Ph.D.)



L. R. Sriram Chivukula

(M.S.)



Kaixi Hou

(Ph.D.)



Yilong Jin

(Ph.D.)



Rubasri Kalidas

(M.S.)



Umar Kalim

(Ph.D.)



Konstantinos Krommydas

(Ph.D.)



Lokendra Panwar

(M.S.)



Sumedha Puppala

(M.S.)



Salvatore Pezzino

(M.S.)



Thomas Scogland

(Ph.D.)
NDSEG Fellowship



Mariam Umar

(Ph.D.)



Balaji Subramaniam

(Ph.D.)



Xiaodong Yu

(Ph.D.)



Da Zhang

(Ph.D.)



Jing Zhang

(Ph.D.)

Collaborators (Current & Past)



Peter Athanas

Dept. of ECE, Virginia
Tech.



Pavan Balaji

Argonne National
Laboratory.



Keith Bisset

Virginia Bioinformatics
Institute,
Virginia Tech.



Eric Brown

Office of IT,
Virginia Tech.



Yong Cao

Dept. of CS,
Virginia Tech.



Bronis de Supinski

Lawrence Livermore



Jack Edwards

Dept. of MAE,



Xiaohui (Helen) Gu

Dept. of CS,



Chung-Hsing Hsu



Hong Luo

Dept. of MAE,

Funding Acknowledgements



VirginiaTech

AMD



IBM



AMD

HARRIS



XILINX



SUPERMICRO



Microsoft

JUNIPER NETWORKS

CISCO



Microsoft featured our "Big Data in the Cloud" grant in U.S. Congressional Testimony

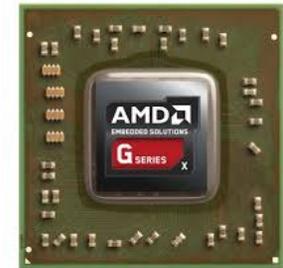
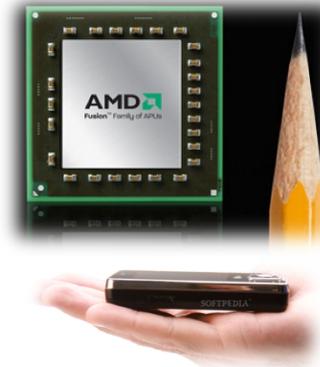
Conclusion

- An ecosystem for heterogeneous parallel computing



#96 on
TOP500
(11/11)

Highest-ranked commodity supercomputer
in U.S. on the **Green500** (11/11)



- Enabling software that tunes parameters of hardware devices
... with respect to **performance, programmability, and portability**
... via a benchmark suite of dwarfs (i.e., motifs) and mini-apps

Wu Feng, wfeng@vt.edu, 540-231-1192



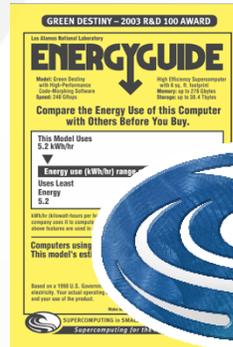
<http://synergy.cs.vt.edu/>



<http://www.chrec.org/>



<http://www.mpiblast.org/>



SUPERCOMPUTING
in SMALL SPACES

<http://sss.cs.vt.edu/>



<http://www.green500.org/>



<http://myvice.cs.vt.edu/>

"Accelerators 'R Us"

<http://accel.cs.vt.edu/>