CPOM Panel: Specialization in Hardware and Software

Kunle Olukotun
Pervasive Parallelism Laboratory
Stanford University
ppl.stanford.edu
Driving Forces: The 4 Ps

- Power
- Performance
- Productivity
- Portability
Computing System Power

\[ \text{Power} = \text{Energy}_{Op} \times \frac{\text{Ops}}{\text{second}} \]

FIXED
Heterogeneous Hardware

- Heterogeneous HW for energy efficiency
  - Multi-core, ILP, threads, data-parallel engines, custom engines
  - H.264 encode study

How do we get the benefits of heterogeneous hardware for all applications?

Source: Understanding Sources of Inefficiency in General-Purpose Chips (ISCA’10)
Green Flash: 100 PetaFLOPS System Concept with 25M CPUs

VLIW CPU (Tensilica core):
- 128b load-store + 2 DP MUL/ADD + integer op/ DMA per cycle:
- Synthesizable at 1GHz Hz in commodity 45nm
- 0.5mm² core, 1.7mm² with inst cache, data cache data RAM, DMA interface, 0.15mW/MHz
- Double precision SIMD FP : 4 ops/cycle (4 GFLOPs)
- Vectorizing compiler, lightweight communications library, cycle-accurate simulator, debugger GUI
- 8 channel DMA for streaming from on/off chip DRAM
- Nearest neighbor 2D communications grid

20 GFLOPS/Watt
~500 m²
~5MWatts
~ $100M

32 boards per rack
380 racks @ ~15KW

32 chip + memory clusters per board
(8.2 TFLOPS @ 450W)

8 DRAM per processor chip:
50 GB/s

64 processors per 45nm chip
512 GFLOPS @ 10W
**Current Heterogeneous Programming Challenge**

**Applications**
- Scientific Engineering
- Virtual Worlds
- Personal Robotics
- Data Informatics

**Too many different low-level programming models**

- Pthreads
- OpenMP
- CUDA
- OpenCL
- MPI
- PGAS
- Sun T2
- Nvidia Fermi
- Altera FPGA
- Cray Jaguar

**Virtual Worlds**

**Scientific Engineering**

**Personal Robotics**

**Data Informatics**
Architecture Up

Applications in Domain X

Arch. Specific Parallel Programming

Architecture Specific Languages

Direct Mapping

Abstraction

Architectures
Application Down vs. Architecture Up

Architectures in Domain X

Applications in Domain X

Arch. Specific Parallel Programming

Generalization

Capture App. Knowledge

Domain X Specific Language

Architectures

Architecture Specific Languages

Direct Mapping

Abstraction

DSL Compiler
Domain Specific Languages (DSLs)

Definition: A language or library with restrictive expressiveness that exploits domain knowledge for productivity and efficiency.

High-level, usually declarative, and deterministic.
Way Forward ⇒
Domain Specific Languages

Performance
(Heterogeneous Parallelism)

Domain Specific Languages

Productivity

Generality

SQL
MATLAB
C/C++
Java
Python
Ruby
**DSL Benefits**

**Productivity**
- Shield average programmers from the difficulty of parallel programming
- Focus on developing algorithms and applications and not on low level implementation details

**Performance**
- Match high level domain abstraction to generic parallel execution patterns
- Restrict expressiveness to more easily and fully extract available parallelism
- Use domain knowledge for static/dynamic optimizations

**Portability and forward scalability**
- DSL & Runtime can be evolved to take advantage of latest hardware features
- Applications remain unchanged
- Allows innovative HW without worrying about application portability
Many DSLs

Applications
- Scientific Engineering
- Virtual Worlds
- Personal Robotics
- Data informatics

Domain Specific Languages
- Statistics (R)
- Physics (Liszt)
- Data Analytics (OptiQL)
- Graph Alg. (Green Marl)
- Machine Learning (OptiML)

Heterogeneous Hardware

New Arch.
Common DSL Infrastructure: DSL Compiler Generator

Applications
- Scientific Engineering
- Virtual Worlds
- Personal Robotics
- Data informatics

Domain Specific Languages
- Statistics (R)
- Physics (Liszt)
- Data Analytics (OptiQL)
- Graph Alg. (Green Marl)
- Machine Learning (OptiML)

DSL Infrastructure
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler
- DSL Compiler

Heterogeneous Hardware
- New Arch.
Delite: DSL Compilation and Execution

// x : TrainingSet[Double]
// mu0, mu1 : Vector[Double]

val sigma = sum(0, m) { i =>
  if (x.labels(i) == false)
    (((x(i)-mu0).t) ** (x(i)-mu0))
  else
    (((x(i)-mu1).t) ** (x(i)-mu1))
}

DSL application
Delite Compiler Framework
Delite Runtime
Cross-layer Specialization and the 4 Ps

- Power
- Performance
- Productivity
- Portability

Application Specific Hardware

Domain Specific Languages