Domain-Specific Languages for Heterogeneous Computing Platforms

Pat Hanrahan
Stanford Pervasive Parallelism Laboratory
(Supported by Sun, AMD, NVIDIA, IBM, Intel, NEC, HP)

Parallel @ Illinois DLS
April 15, 2010

Power Efficiency is Key to Exascale

Source: DARPA Exascale Hardware and Software Studies
DE Shaw Research: Anton

Molecular dynamics computer

100 times more power efficient

D. E. Shaw et al. Supercomputer 2009, Best Paper and Gordon Bell Prize

Sequential vs. Throughput Processors

20 times greater throughput for same area and power
½ the sequential performance

<table>
<thead>
<tr>
<th></th>
<th>2 out of order</th>
<th>10 in-order</th>
</tr>
</thead>
<tbody>
<tr>
<td># CPU cores</td>
<td>4 per clock</td>
<td>2 per clock</td>
</tr>
<tr>
<td>Instructions per issue</td>
<td>4-wide SSE</td>
<td>16-wide</td>
</tr>
<tr>
<td>VPU lanes per core</td>
<td>4 MB</td>
<td>4 MB</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>4 per clock</td>
<td>2 per clock</td>
</tr>
<tr>
<td>Single-stream</td>
<td>8 per clock</td>
<td>160 per clock</td>
</tr>
<tr>
<td>Vector throughput</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Argument for Heterogeneity [Moore, AMD]

Power efficiency
- CPUs more efficient for sequential workloads
- GPUs more efficient for data-parallel workloads

Amdahl’s Law
- Real applications have a mixture of sequential and parallel code
- Parallelism often limited by sequential code

Therefore,
- Optimal platform involves both sequential cores plus data-parallel cores

“Fusion” Architectures

Emergence of a hybrid processor
- 2-8 CPUs
- 16-64 GPUs
- Hardware for video compression/decompression
- ...

Plans announced by AMD and Intel
Already being deployed in mobile computers and smart phones
Apple iPhone 3GS

Samsung ARM Cortex A8

S5PC100 Block Diagram
Imagination PowerVR SGX535

SGX520 3.5M Tri/S, 125M Pix/s @ 100 Mhz

Apple A4 in the iPad

Contains CPU and GPU and …
Multiple Parallel Platforms

Cluster
- Distributed memory
- System area network

Multi-core SMP (e.g. 32 core, 4-socket systems)
- Shared memory

Many-core GPU (e.g. Cell, Fermi)
- SIMD / SIMT architecture
- Local memory on chip / Separate GPU memory
- Accelerator connected via PCI-E

Multiple Parallel Programming Models

Cluster
- MPI

Multi-core SMP (e.g. 32 core, 4-socket systems)
- Threads/locks, OpenMP

Many-core GPU (e.g. Cell, Fermi)
- CUDA, OpenCL
Complex Heterogeneous Platforms

Combined into heterogeneous/hybrid machines

LANL IBM Roadrunner
  (Opteron + Cell)

ORNL Cray 20 PFLOPs
  (Opteron + Fermi)

Is it Possible to Write One Program and Run it on all these Machines?
Traditional Answers

1. Commit to a standard programming paradigm and emulate on different architectures
   - For example, MPI is widely used; emulate on an SMP? on a GPU?
   - Combine MPI plus another; MPI + OpenCL?

2. General-purpose parallel programming language
   - For example, new DARPA HPCS languages X10, Chapel, Fortress

Alternative Approach:

Domain-Specific Libraries and Languages
Domain-Specific Languages

Definition: A language or library that exploits domain knowledge for productivity and efficiency

Widely used in many application areas
- matlab / R
- SQL / map-reduce / Microsoft’s LINQ
- OpenGL/D3D and Cg/HLSL
- ...

DSLs are a hot topic now
- Programming language community (C#, Scala)
- Web programming environments (Ruby)

Graphics Libraries

glPerspective(45.0);
for( ... ) {
    glTranslate(1.0,2.0,3.0);
    glBegin(GL_TRIANGLES);
        glVertex(...);
        glVertex(...);
        ...
    glEnd();
}
glSwapBuffers();
OpenGL “Grammar”

<Scene> = <BeginFrame> <Camera> <World> <EndFrame>

<Camera> = glMatrixMode(GL_PROJECTION) <View>
<View> = glPerspective | glOrtho

<World> = <Objects>*
<Object> = <Transforms>* <Geometry>
<Transforms> = glTranslatef | glRotatef | ...
<Geometry> = glBegin <Vertices> glEnd
<Vertices> = [glColor] [glNormal] glVertex

Advantages

Productivity
- Graphics library is easy to use

Portability
- Runs on wide range of GPUs
Advantages

Productivity
Portability
Performance

- Vertices/Fragments are independent and coherent
- Rasterization can be done using SIMD hardware
- Efficient framebuffer scatter-ops
- Textures are read-only; texture filtering hw
- Specialized scheduler for pipeline
- ...

*Allows for super-optimized implementations*

Advantages

Productivity
Portability
Performance

Encourage innovation

- Allows vendors to radically optimize hardware architecture to achieve efficiency
- Allows vendors to introduce new low-level programming models and abstractions
Beyond Graphics?

Physical simulation
   Liszt – PDEs on meshes (fluid flow and finite element)
   Random[T] – Monte Carlo and UQ

Data analysis
   OptiML – Machine learning
   Kore – Nested data parallelism

Liszt

Z. DeVito, M. Medina, M. Barrientos,
E. Elsen, N. Joubert,
J. Alonso, E. Darve, F. Ham, P. Hanrahan

“...the most technically advanced and perhaps greatest pianist of all time... made playing complex pieces on the piano seem effortless...”
Characterize the operability limits of a hypersonic propulsion system using predictive computations. Primary focus is the unstart phenomena triggered by thermal choking in a hydrogen-fueled scramjet.

- State-of-the-art unstructured RANS solver
  - Main tool for system-level simulation
Typical Joe C Code Kernel

for (int ifa = 0; ifa < nfa; ifa++) {
    double x_fa_approx[3] = {0.,0.,0.};
    for (int nof = noofa_i[ifa];
        nof < noofa_i[ifa + 1]; nof++) {
        for (int i = 0; i < 3; i++)
            x_fa_approx[i] = x_no[nofa_v[nof]][i];
    }
    for (int i = 0; i < 3; i++)
        x_fa_approx[i] /= (double) (noofa_i[ifa + 1] - 1);
    for (int nof = noofa_i[ifa]; nof < noofa_i[ifa + 1]; nof++) {
        int ino1 = nof; ino2 = noofa_v[nof];
        double v1[3],v2[3];
        for (int i = 0; i < 3; i++)
            v1[i] = x_no[ino1][i] - x_fa_approx[i];
        for (int i = 0; i < 3; i++)
            v2[i] = x_no[ino2][i] - x_fa_approx[i];
        fa_normal[ifa][1] += 0.5 * (v1[2] * v2[0] - v1[0] * v2[2]);
        fa_normal[ifa][2] += 0.5 * (v1[0] * v2[1] - v1[1] * v2[0]);
    }
}

Minimal Abstraction

for (int ifa = 0; ifa < nfa; ifa++) {
    double x_fa_approx[3] = {0.,0.,0.};
    for (int nof = noofa_i[ifa];
        nof < noofa_i[ifa + 1]; nof++) {
        for (int i = 0; i < 3; i++)
            x_fa_approx[i] = x_no[nofa_v[nof]][i];
    }
    for (int i = 0; i < 3; i++)
        x_fa_approx[i] /= (double) (noofa_i[ifa + 1] - 1);
    for (int nof = noofa_i[ifa]; nof < noofa_i[ifa + 1]; nof++) {
        int ino1 = nof; ino2 = noofa_v[nof];
        double v1[3],v2[3];
        for (int i = 0; i < 3; i++)
            v1[i] = x_no[ino1][i] - x_fa_approx[i];
        for (int i = 0; i < 3; i++)
            v2[i] = x_no[ino2][i] - x_fa_approx[i];
        fa_normal[ifa][1] += 0.5 * (v1[2] * v2[0] - v1[0] * v2[2]);
        fa_normal[ifa][2] += 0.5 * (v1[0] * v2[1] - v1[1] * v2[0]);
    }
}
Joe in Liszt is Higher-Level

val pos = new Field[Vertex,double3]  
val A = new SparseMatrix[Vertex,Vertex]

for( c <- cells(mesh) ) {  
  val center = avg(pos(c.vertices))  
  for( f <- faces(c) ) {  
    val face_dx = avg(pos(f.vertices)) – center  
    for ( e <- f edgesCCW c ) {  
      val v0 = e.tail  
      val v1 = e.head  
      val v0_dx = pos(v0) – center  
      val v1_dx = pos(v1) – center  
      val face_normal = v0_dx cross v1_dx  
      // calculate flux for face …  
      A(v0,v1) += …  
      A(v1,v0) -= …

Built-in Features

Objects
- Mesh, cells, faces, edges, vertices
- Fields
- Linear operators as matrices (sparse and dense)
- Short vectors for positions and normals
- Sets (unordered) and lists (ordered)

Solvers
- Sparse matrix solvers (e.g. ANL PETSc, trilinos, …)
Optimize Using Domain Knowledge

Knowledge about topological relationships on the mesh are built into the compiler

We can use program analysis to

- Perform domain decomposition
- Identify and communicate ghost cells
- Optimize data layout for caches and/or vector machines with local memories
var rho: Field[Cell, double]

for( f <- mesh.faces ) {
  val rhoOutside(f) = calc_flux( f, rho(f.outside))
  + calc_flux( f, rho(f.inside ))
}

Program Analysis of Neighborhoods
var rho: Field[Cell, double]

for(f <- mesh.faces) {
  val outside  = f.outside
  var rhoOutside(f) = 0.5 * rho(c)
  for(c <- outside.cells) {
    rhoOutside(f) += 0.25 * rho(c)
  }
}

---

**Program Analysis of Neighborhoods**

Node 0

Owned Cells

Ghost Cells

---

**Domain Decomposition / Ghost Cells**

Liszt creates a graph of mesh adjacencies needed to run the algorithm

Graph is handed to ParMETIS to determine optimal partition

Communication of information in ghost cells is also automatically handed

Node 0

Owned Cells
**MPI Performance**

220K element mesh
Scalability limited by size of mesh, not by the Liszt implementation

**Programmer Productivity**

Lines of Code

- **Flow Solver**
- **Common Library**
- **Parallel Runtime**
How to Create
Domain-Specific Languages

“Little” Languages (UNIX)

The roll-your-own approach

Examples: sh, make, matlab, R, ...

Disadvantages:
- Continual requests for more generality
- Proliferation of syntax: “$x”
- Cannot use multiple DSLs in the same application
- Expensive to develop a complete system
Embedded DSL (EDSL)

An EDSL uses the embedding language’s compiler to implements parts of the language.

Examples: OpenGL, Lisp, C++ templates, C#, Haskell, Ruby, Scala

Advantages:
+ Consistent base syntax
+ Multiple DSLs may interoperate
+ Reusable infrastructure (compiling, debugging, …)

LINQ (Language-Integrated Query)

var personsNotInSeattle =
    from p in person
    where p.Address != "Seattle"
    orderby p.FirstName
    select p;

Designed by Microsoft to make it easier to write applications that use databases
Enhanced C# (3.0)

Supporting LINQ led to major additions to the language
- Implicitly typed variables
- Lambdas
- Anonymous classes
- Extensions (implicit type wrapper)
- Runtime code generation via expression trees

LINQ Enables Portable Parallelism

Multiple implementations
- SQL engines
- PLINQ: SMP
- DryadLINQ: Clusters
Multi-Stage Polymorphic Embedding

w/ M. Odersky, K. Olukotun

A DSL for Matrices

trait MatrixArith {
  type Rep[Mat]
  implicit def liftMatrix(x: Mat): Rep[Mat]
  implicit def matrixRepArith(x: Rep[Mat]) = new {
    def +(y: Rep[Mat]) = plus(x,y)
    def *(y: Rep[Mat]) = times(x,y)
  }
  def plus(x: Rep[Mat], y: Rep[Mat]): Rep[Mat]
  def times(x: Rep[Mat], y: Rep[Mat]): Rep[Mat]
}
Polymorphic Embedding

trait TestMatrix {
    def Example(a: Rep[Mat], b: Rep[Mat],
                c: Rep[Mat], d: Rep[Mat]): Rep[Mat] = {
        val x = a*b + a*c
        val y = a*c + a*d
        return x+y
    }
}

This code is in the DSL because it uses Rep[Mat]
Rep[Mat] is abstract; multiple concrete implementations
Thus, the code is polymorphic
We call this a polymorphic embedding of the DSL

Direct or Pure Embedding

trait MatrixArithDirect {
    type Rep[Mat] = Mat

    implicit def liftMatrix(x: Mat) = x

    def plus(x: Rep[Mat], y: Rep[Mat]): Rep[Mat] = x + y
    def times(x: Rep[Mat], y: Rep[Mat]): Rep[Mat] = x * y
}

object Test extends TestMatrix with MatrixArithDirect
DSL as an Expression Tree

trait MatrixArithExp {
    type Rep[Mat] = Exp[Mat]
    case class Plus(x: Exp[Mat], y: Exp[Mat])
        extends Op[Mat]
    case class Times(x: Exp[Mat], y: Exp[Mat])
        extends Op[Mat]
    def plus(x: Exp[Mat], y: Exp[Mat]) = Plus(x, y)
    def times(x: Exp[Mat], y: Exp[Mat]) = Times(x, y)
}

object Test extends TestMatrix with MatrixArithExp

Expression Tree: Exp[T]

trait Expressions = {
    abstract class Exp[T]
    case class Const[T](x: T) extends Exp[T]
    case class Sym[T](n: Int) extends Exp[T]

    def fresh[T]: Sym[T]

    abstract class Op[T]

    ...}
}
Extend with an Optimizer

trait MatrixArithExpOpt extends MatrixArithExp {
  override def plus(x: Exp[Mat]), y: Exp[Mat]) =
  (x,y) match {
    case (Times(a,b), Times(c,d))
      if (a==c) => Times( a, Plus( b,d ))
    case _ => super.plus(x,y)
  }
}

Polymorphic Embedding

Create abstract types that are representation independent
- Operations on Rep[T] define the DSL
- Concrete types created with traits (mixins)

Concrete types enable optimized implementations
- Direct embedding with T
  “lift” program on T to Rep[T]
- Alternative representations for specific hardware
  T = {CPU[T], GPU[T], SMP[T], …}
- Expression tree
  Can generate Exp[T] from the DSL
  Domain-specific and general-purpose analyzers/optimizers
Multi-Stage Polymorphic Embedding

Staging
- Rep[T] is dynamic
- T is static
- Metaprogramming implemented by converting dynamic to static
  - Dynamically compile code and link class library


Scala: Scalable Language

Designed to embed DSLs
- Concise syntax, implicit type conversions, ...

Functional programming
- Higher-order functions, lambdas, closures and continuations
- Encourages the use of immutable data structures
- Discourages programs with side-effects

Object-oriented programming
- Allows mutable data structures
- Strong type system, parameterized types, traits

Support for concurrency
Research Challenges

Lifting arbitrary subsets of the base language
Currently some operations cannot be lifted
e.g. if, while, var, def, lambda, …

Exposing compiler optimizations
- Common sub-expression elimination

Taming effects
- Side effects
- Exceptions

Mixing multiple DSLs
Debugging complex DSLs

The PPL Vision

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

Domain Specific Languages
- Rendering (Spark)
- Physics (List)
- Scripting
- Probabilistic (RandomT)
- Machine Learning (OptiML)

Embedding Language (Scala)

Common Parallel Runtime (Delite, Sequoia)
- Domain specific optimization
- Task & data parallelism
- Locality aware scheduling

DSL Infrastructure

Domain Specific Languages

Heterogeneous Hardware

Hardware Architecture
- OOO Cores
- SIMD Cores
- Threaded Cores
- Programmable Hierarchies
- Scalable Coherence
- Isolation & Atomicity
- Pervasive Monitoring
Summary

Need for power-efficiency causing heterogeneity

Domain-specific libraries and languages are productive, portable, and performant; also encourage innovation

Liszt is a DSL being developed for solving PDEs on meshes

Liszt uses domain-knowledge to map to heterogeneous platforms

Embedding DSLs is better than rolling-your-own

Challenge: Better EDSL technology and tools

Thank you
**Rake – Make in Ruby**

```
SRC = FileList['*.c']
OBJ = SRC.ext('o')

task :default => ['hello']
rule '.o' => '.c' do |t|
  sh "cc -c -o #{t.name} #{t.source}"
end
file "hello" => OBJ do
  sh "cc -o hello #{OBJ}"
end

# File dependencies go here ...
file 'main.o' => ['main.c', 'greet.h']
file 'greet.o' => ['greet.c']
```

**Results**

Implemented a version of Joe in Liszt

- Explicit Euler
- No turbulence
- Targets C++ layer directly

Calculates identical results as C++ version