

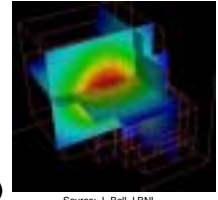
Titanium: A Java Dialect for High Performance Computing

Katherine Yelick

U.C. Berkeley and LBNL

Motivation: Target Problems

- ◆ Many modeling problems in astrophysics, biology, material science, and other areas require
 - Enormous range of spatial and temporal scales
- ◆ To solve interesting problems, one needs:
 - Adaptive methods
 - Large scale parallel machines
- ◆ Titanium is designed for
 - Structured grids
 - Locally-structured grids (AMR)
 - Unstructured grids (in progress)



Source: J. Bell, LBNL

March 5, 2004

CS267 Lecture 12

2

Titanium Background

- ◆ Based on Java, a cleaner C++
 - Classes, automatic memory management, etc.
 - Compiled to C and then machine code, **no JVM**
- ◆ Same parallelism model at UPC and CAF
 - SPMD parallelism
 - Dynamic Java threads are not supported
- ◆ Optimizing compiler
 - Analyzes global synchronization
 - Optimizes pointers, communication, memory

March 5, 2004

CS267 Lecture 12

3

Summary of Features Added to Java

- ◆ Multidimensional arrays: iterators, subarrays, copying
- ◆ Immutable (“value”) classes
- ◆ Templates
- ◆ Operator overloading
- ◆ Scalable SPMD parallelism replaces threads
- ◆ Global address space with local/global reference distinction
- ◆ Checked global synchronization
- ◆ Zone-based memory management (regions)
- ◆ Libraries for collective communication, distributed arrays, bulk I/O, performance profiling

March 5, 2004

CS267 Lecture 12

4

Outline

- ◆ Titanium Execution Model
 - SPMD
 - Global Synchronization
 - Single
- ◆ Titanium Memory Model
- ◆ Support for Serial Programming
- ◆ Performance and Applications
- ◆ Compiler/Language Status

March 5, 2004

CS267 Lecture 12

5

SPMD Execution Model

- ◆ Titanium has the same execution model as UPC and CAF
- ◆ Basic Java programs may be run as Titanium programs, but all processors do all the work.
- ◆ E.g., parallel hello world

```
class HelloWorld {
    public static void main (String [] argv) {
        System.out.println("Hello from proc "
            + Ti.thisProc()
            + " out of "
            + Ti.numProcs());
    }
}
```
- ◆ Global synchronization done using `Ti.barrier()`

March 5, 2004

CS267 Lecture 12

6

Barriers and Single

- ◆ Common source of bugs is barriers or other collective operations inside branches or loops
 - `barrier, broadcast, reduction, exchange`
- ◆ A “single” method is one called by all procs
 - `public single static void allStep(...)`
- ◆ A “single” variable has same value on all procs
 - `int single timestep = 0;`
- ◆ Single annotation on methods is optional, but useful in understanding compiler messages
- ◆ Compiler proves that all processors call barriers together

March 5, 2004

CS267 Lecture 12

7

Explicit Communication: Broadcast

- ◆ Broadcast is a one-to-all communication
 - `broadcast <value> from <processor>`
- ◆ For example:
 - `int count = 0;`
 - `int allCount = 0;`
 - `if (Ti.thisProc() == 0) count = computeCount();`
 - `allCount = broadcast count from 0;`
- ◆ The processor number in the broadcast must be single; all constants are single.
 - All processors must agree on the broadcast source.
- ◆ The `allCount` variable could be declared single.
 - All will have the same value after the broadcast.

March 5, 2004

CS267 Lecture 12

8

More on Single

- ◆ Global synchronization needs to be controlled
 - `if (this processor owns some data) {`
 - `compute on it`
 - `barrier`
 - `}`
- ◆ Hence the use of “single” variables in Titanium
- ◆ If a conditional or loop block contains a barrier, all processors must execute it
 - conditions must contain only single variables
- ◆ Compiler analysis statically enforces freedom from deadlocks due to barrier and other collectives being called non-collectively "Barrier Inference" [Gay & Aiken]

March 5, 2004

CS267 Lecture 12

9

Single Variable Example

- ◆ Barriers and single in N-body Simulation
 - `class ParticleSim {`
 - `public static void main (String [] argv) {`
 - `int single allTimestep = 0;`
 - `int single allEndTime = 100;`
 - `for (; allTimestep < allEndTime; allTimestep++){`
 - `read remote particles, compute forces on mine`
 - `Ti.barrier();`
 - `write to my particles using new forces`
 - `Ti.barrier();`
 - `}`
 - `}`
 - `}`
- ◆ Single methods inferred by the compiler

March 5, 2004

CS267 Lecture 12

10

Outline

- ◆ Titanium Execution Model
- ◆ Titanium Memory Model
 - Global and Local References
 - Exchange: Building Distributed Data Structures
 - Region-Based Memory Management
- ◆ Support for Serial Programming
- ◆ Performance and Applications
- ◆ Compiler/Language Status

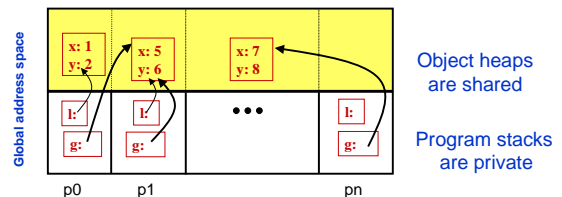
March 5, 2004

CS267 Lecture 12

11

Global Address Space

- ◆ Globally shared address space is partitioned
- ◆ References (pointers) are either local or global (meaning possibly remote)



March 5, 2004

CS267 Lecture 12

12

Use of Global / Local

- ◆ Global references (pointers) may point to remote locations
 - Reference are global by default
 - Easy to port shared-memory programs
- ◆ Global pointers are more expensive than local
 - True even when data is on the same processor
 - Costs of global:
 - ◆ space (processor number + memory address)
 - ◆ dereference time (check to see if local)
- ◆ May declare references as **local**
 - Compiler will automatically infer **local** when possible
 - This is an important performance-tuning mechanism

March 5, 2004

CS267 Lecture 12

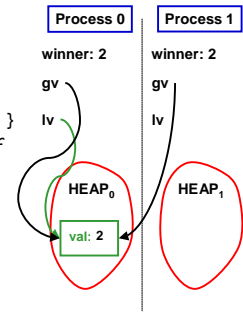
13

Global Address Space

- ◆ Processes allocate locally
- ◆ References can be passed to other processes

```

class C { public int val;... }
C gv; // global pointer
C local lv; // local pointer
if (Ti.thisProc() == 0) {
    lv = new C();
}
gv = broadcast lv from 0;
//data race
gv.val = Ti.thisProc()+1;
int winner = gv.val
    
```



March 5, 2004

CS267 Lecture 12

14

Aside on Titanium Arrays

- ◆ Titanium adds its own multidimensional array class for performance
- ◆ Distributed data structures are built using a 1D Titanium array
- ◆ Slightly different syntax, since Java arrays still exist in Titanium, e.g.:


```

int [1d] a;
a = new int [1:100];
a[1] = 2*a[1] - a[0] - a[2];
            
```
- ◆ Will discuss these more later...

March 5, 2004

CS267 Lecture 12

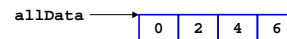
15

Explicit Communication: Exchange

- ◆ To create shared data structures
 - each processor builds its own piece
 - pieces are exchanged (for objects, just exchange pointers)
- ◆ Exchange primitive in Titanium


```

int [1d] single allData;
allData = new int [0:Ti.numProcs()-1];
allData.exchange(Ti.thisProc()*2);
            
```
- ◆ E.g., on 4 procs, each will have copy of allData:



March 5, 2004

CS267 Lecture 12

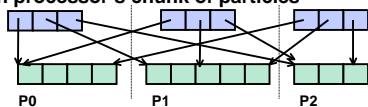
16

Distributed Data Structures

- ◆ Building distributed arrays:


```

Particle [1d] single [1d] allParticle =
    new Particle [0:Ti.numProcs-1][1d];
Particle [1d] myParticle =
    new Particle [0:myParticleCount-1];
allParticle.exchange(myParticle);
            
```
- ◆ Now each processor has array of pointers, one to each processor's chunk of particles



March 5, 2004

CS267 Lecture 12

17

Region-Based Memory Management

- ◆ An advantage of Java over C/C++ is:
 - Automatic memory management
- ◆ But garbage collection:
 - Has a reputation of slowing serial code
 - Does not scale well in a parallel environment
- ◆ Titanium approach:
 - Preserves safety - cannot deallocate live data
 - Garbage collection is the default (on most platforms)
 - Higher performance is possible using region-based explicit memory management
 - Takes advantage of memory management phases

March 5, 2004

CS267 Lecture 12

18

Region-Based Memory Management

- ◆ Need to organize data structures
- ◆ Allocate set of objects (safely)
- ◆ Delete them with a single explicit call (fast)

```
PrivateRegion r = new PrivateRegion();
for (int j = 0; j < 10; j++) {
    int[] x = new ( r ) int[j + 1];
    work(j, x);
}
try { r.delete(); }
catch (RegionInUse oops) {
    System.out.println("failed to delete");
}
}
```

March 5, 2004

CS267 Lecture 12

19

Outline

- ◆ Titanium Execution Model
- ◆ Titanium Memory Model
- ◆ Support for Serial Programming
 - Immutables
 - Operator overloading
 - Multidimensional arrays
 - Templates
- ◆ Performance and Applications
- ◆ Compiler/Language Status

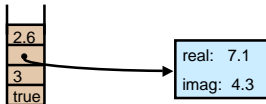
March 5, 2004

CS267 Lecture 12

20

Java Objects

- ◆ Primitive scalar types: boolean, double, int, etc.
 - implementations store these on the program stack
 - access is fast -- comparable to other languages
- ◆ Objects: user-defined and standard library
 - always allocated dynamically in the heap
 - passed by pointer value (object sharing)
 - has implicit level of indirection
 - simple model, but inefficient for small objects



March 5, 2004

CS267 Lecture 12

21

Java Object Example

```
class Complex {
    private double real;
    private double imag;
    public Complex(double r, double i) {
        real = r; imag = i; }
    public Complex add(Complex c) {
        return new Complex(c.real + real, c.imag + imag);
    }
    public double getReal { return real; }
    public double getImag { return imag; }
}
```

```
Complex c = new Complex(7.1, 4.3);
c = c.add(c);
class VisComplex extends Complex { ... }
```

March 5, 2004

CS267 Lecture 12

22

Immutable Classes in Titanium

- ◆ For small objects, would sometimes prefer
 - to avoid level of indirection and allocation overhead
 - pass by value (copying of entire object)
 - especially when immutable -- fields never modified
 - ◆ extends the idea of primitive values to user-defined types
- ◆ Titanium introduces immutable classes
 - all fields are implicitly **final** (constant)
 - **cannot inherit** from or be inherited by other classes
 - needs to have 0-argument constructor
- ◆ Examples: Complex, xyz components of a force
- ◆ Note: considering lang. extension to allow mutation

March 5, 2004

CS267 Lecture 12

23

Example of Immutable Classes

```
immutable class Complex {
    Complex () {real=0; imag=0;}
    ...
}
```

Annotations in the original image:

- new keyword (points to 'new')
- Zero-argument constructor required (points to 'Complex () {real=0; imag=0;}')
- Rest unchanged. No assignment to fields outside of constructors. (points to '...')

- ◆ Use of immutable complex values


```
Complex c1 = new Complex(7.1, 4.3);
Complex c2 = new Complex(2.5, 9.0);
c1 = c1.add(c2);
```
- ◆ Addresses performance and programmability
 - Similar to C structs in terms of performance
 - Support for Complex with a general mechanism

March 5, 2004

CS267 Lecture 12

24

Operator Overloading

- ◆ Titanium provides operator overloading
 - Convenient in scientific code
 - Feature is similar to that in C++

```
class Complex {
    ...
    public Complex op+(Complex c) {
        return new Complex(c.real + real, c.imag + imag);
    }
}
```

```
Complex c1 = new Complex(7.1, 4.3);
Complex c2 = new Complex(5.4, 3.9);
Complex c3 = c1 + c2;
```

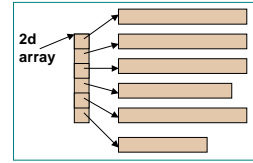
March 5, 2004

CS267 Lecture 12

25

Arrays in Java

- ◆ Arrays in Java are objects
- ◆ Only 1D arrays are directly supported
- ◆ Multidimensional arrays are arrays of arrays
- ◆ General, but slow
- ◆ Subarrays are important in AMR (e.g., interior of a grid)
 - Even C and C++ don't support these well
 - Hand-coding (array libraries) can confuse optimizer
- ◆ Can build multidimensional arrays, but we want
 - Compiler optimizations and nice syntax



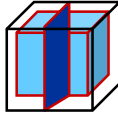
March 5, 2004

CS267 Lecture 12

26

Multidimensional Arrays in Titanium

- ◆ New multidimensional array added
 - Supports subarrays without copies
 - ◆ can refer to rows, columns, slabs interior, boundary, even elements...
 - Indexed by Points (tuples of ints)
 - Built on a rectangular set of Points, RectDomain
 - Points, Domains and RectDomains are built-in immutable classes, with useful literal syntax
- ◆ Support for AMR and other grid computations
 - domain operations: intersection, shrink, border
 - bounds-checking can be disabled after debugging



March 5, 2004

CS267 Lecture 12

27

Unordered Iteration

- ◆ Motivation:
 - Memory hierarchy optimizations are essential
 - Compilers sometimes do these, but hard in general
- ◆ Titanium has explicitly unordered iteration
 - Helps the compiler with analysis
 - Helps programmer avoid indexing details

```
foreach (p in r) { ... A[p] ... }
```

 - ◆ p is a Point (tuple of ints), can be used as array index
 - ◆ r is a RectDomain or Domain
- ◆ Additional operations on domains to transform
- ◆ Note: foreach is not a parallelism construct

March 5, 2004

CS267 Lecture 12

28

Point, RectDomain, Arrays in General

- ◆ Points specified by a tuple of ints


```
Point<2> lb = [1, 1];
Point<2> ub = [10, 20];
```
- ◆ RectDomains given by 3 points:
 - lower bound, upper bound (and optional stride)

```
RectDomain<2> r = [lb : ub];
```
- ◆ Array declared by num dimensions and type


```
double [2d] a;
```
- ◆ Array created by passing RectDomain


```
a = new double [r];
```

March 5, 2004

CS267 Lecture 12

29

Simple Array Example

- ◆ Matrix sum in Titanium


```
Point<2> lb = [1,1];
Point<2> ub = [10,20];
RectDomain<2> r = [lb:ub];

double [2d] a = new double [r];
double [2d] b = new double [1:10,1:20];
double [2d] c = new double [lb:ub:[1,1]];

for (int i = 1; i <= 10; i++)
    for (int j = 1; j <= 20; j++)
        c[i,j] = a[i,j] + b[i,j];

foreach(p in c.domain()) { c[p] = a[p] + b[p]; }
```

 - No array allocation here
 - Syntactic sugar
 - Optional stride
 - Equivalent loops

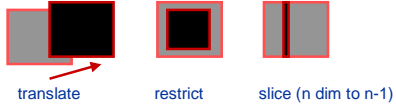
March 5, 2004

CS267 Lecture 12

30

More Array Operations

- ◆ Titanium arrays have a rich set of operations



- ◆ None of these modify the original array, they just create another view of the data in that array
- ◆ You create arrays with a RectDomain and get it back later using `A.domain()` for array A
 - A Domain is a set of points in space
 - A RectDomain is a rectangular one
- ◆ Operations on Domains include `+`, `-`, `*` (union, different intersection)

March 5, 2004

CS267 Lecture 12

31

MatMul with Titanium Arrays

```
public static void matMul(double [2d] a,
                        double [2d] b,
                        double [2d] c) {
    foreach (ij in c.domain()) {
        double [1d] aRowi = a.slice(1, ij[1]);
        double [1d] bColj = b.slice(2, ij[2]);
        foreach (k in aRowi.domain()) {
            c[ij] += aRowi[k] * bColj[k];
        }
    }
}
```

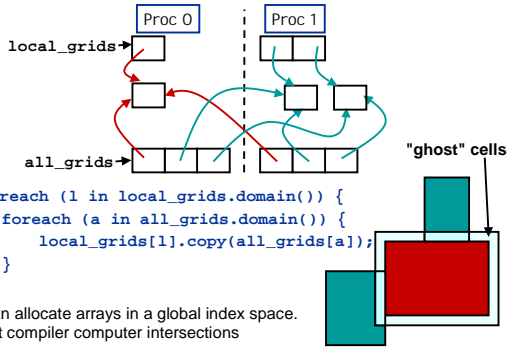
Current performance: comparable to 3 nested loops in C

March 5, 2004

CS267 Lecture 12

32

Example: Setting Boundary Conditions



- Can allocate arrays in a global index space.
- Let compiler compute intersections

March 5, 2004

CS267 Lecture 12

33

Templates

- ◆ Many applications use containers:
 - Parameterized by dimensions, element types,...
 - Java supports parameterization through inheritance
 - ◆ Can only put Object types into containers
 - ◆ Inefficient when used extensively
- ◆ Titanium provides a template mechanism closer to C++
 - Can be instantiated with non-object types (double, Complex) as well as objects
- ◆ Example: Used to build a distributed array package
 - Hides the details of exchange, indirection within the data structure, etc.

March 5, 2004

CS267 Lecture 12

34

Example of Templates

```
template <class Element> class Stack {
    . . .
    public Element pop() {...}
    public void push( Element arrival ) {...}
}

template Stack<int> list = new template Stack<int>();
list.push( 1 ); // ← Not an object
int x = list.pop(); // ← Strongly typed, No dynamic cast
```

- ◆ Addresses programmability and performance

March 5, 2004

CS267 Lecture 12

35

Using Templates: Distributed Arrays

```
template <class T, int single arity>
public class DistArray {
    RectDomain <arity> single rd;
    T [arity d][arity d] subMatrices;
    RectDomain <arity> [arity d] single subDomains;
    ...
    /* Sets the element at p to value */
    public void set (Point <arity> p, T value) {
        getHomingSubMatrix (p) [p] = value;
    }
}

template DistArray <double, 2> single A =
    new template
    DistArray<double, 2> ( [[0,0]:[aHeight, aWidth]] );
```

March 5, 2004

CS267 Lecture 12

36

Outline

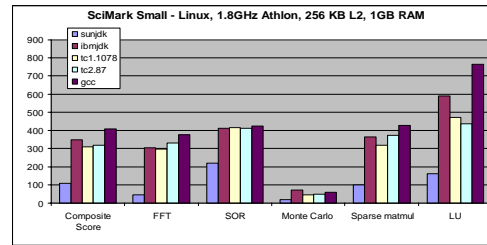
- ◆ Titanium Execution Model
- ◆ Titanium Memory Model
- ◆ Support for Serial Programming
- ◆ Performance and Applications
 - Serial Performance on pure Java (SciMark)
 - Parallel Applications
 - Compiler status & usability results
- ◆ Compiler/Language Status

March 5, 2004

CS267 Lecture 12

37

Java Compiled by Titanium Compiler



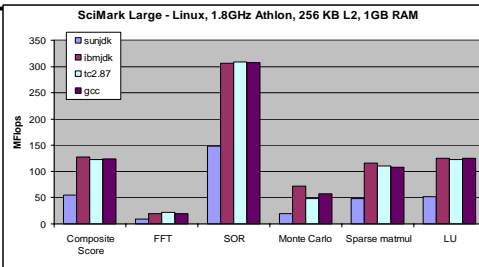
- Sun JDK 1.4.1_01 (HotSpot(TM) Client VM) for Linux
- IBM J2SE 1.4.0 (Classic VM cxia32140-20020917a, jitc JIT) for 32-bit Linux
- Titaniumc v2.87 for Linux, gcc 3.2 as backend compiler -O3, no bounds check
- gcc 3.2, -O3 (ANSI-C version of the SciMark2 benchmark)

March 5, 2004

CS267 Lecture 12

38

Java Compiled by Titanium Compiler



- Same as previous slide, but using a larger data set
- More cache misses, etc.

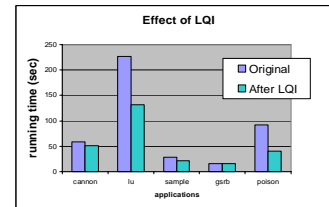
March 5, 2004

CS267 Lecture 12

39

Local Pointer Analysis

- ◆ Global pointer access is more expensive than local
- ◆ Compiler analysis can frequently infer that a given global pointer always points locally
 - Replace global pointer with a local one
 - Local Qualification Inference (LQI)
 - Data structures must be well partitioned



March 5, 2004

CS267 Lecture 12

40

Applications in Titanium

- ◆ Benchmarks and Kernels
 - Scalable Poisson solver for infinite domains
 - NAS PB: MG, FT, IS, CG
 - Unstructured mesh kernel: EM3D
 - Dense linear algebra: LU, MatMul
 - Tree-structured n-body code
 - Finite element benchmark
- ◆ Larger applications
 - Gas Dynamics with AMR
 - Heart and Cochlea simulation (ongoing)
 - Genetics: micro-array selection
 - Ocean modeling with AMR (in progress)

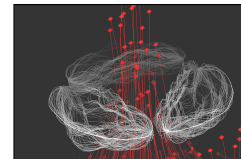
March 5, 2004

CS267 Lecture 12

41

Heart Simulation: Immersed Boundary Method

- ◆ Problem: compute blood flow in the heart
 - Modeled as an elastic structure in an incompressible fluid.
 - ◆ The “immersed boundary method” [Peskin and McQueen].
 - ◆ 20 years of development in model
 - Many other applications: blood clotting, inner ear, paper making, embryo growth, and more
- ◆ Can be used for design of prosthetics
 - Artificial heart valves
 - Cochlear implants



March 5, 2004

CS267 Lecture 12

42

Performance of IB Code

- ◆ IBM SP performance (seaborg)
- ◆ Performance on a PC cluster at Caltech

March 5, 2004 CS267 Lecture 12 43

Error on High-Wavenumber Problem

- ◆ Charge is
 - 1 charge of concentric waves
 - 2 star-shaped charges.
- ◆ Largest error is where the charge is changing rapidly. Note:
 - discretization error
 - faint decomposition error
- ◆ Run on 16 procs

March 5, 2004 CS267 Lecture 12 44

Scalable Parallel Poisson Solver

- ◆ MLC for Finite-Differences by Balls and Colella
- ◆ Poisson equation with infinite boundaries
 - arise in astrophysics, some biological systems, etc.
- ◆ Method is scalable
 - Low communication (<5%)
- ◆ Performance on
 - SP2 (shown) and T3E
 - scaled speedups
 - nearly ideal (flat)
- ◆ Currently 2D and non-adaptive

March 5, 2004 CS267 Lecture 12 45

AMR Gas Dynamics

- ◆ Hyperbolic Solver [McCorquodale and Colella]
 - Implementation of Berger-Colella algorithm
 - Mesh generation algorithm included
- ◆ 2D Example (3D supported)
 - Mach-10 shock on solid surface at oblique angle
- ◆ Future: 3D Ocean Model based on Chombo algorithms
 - [Wen and Colella]

March 5, 2004 CS267 Lecture 12 46

Outline

- ◆ Titanium Execution Model
- ◆ Titanium Memory Model
- ◆ Support for Serial Programming
- ◆ Performance and Applications
- ◆ Compiler/Language Status

March 5, 2004 CS267 Lecture 12 47

Titanium Compiler Status

- ◆ Titanium runs on almost any machine
 - Requires a C compiler and C++ for the translator
 - Pthreads for shared memory
 - GASNet for distributed memory, which exists on
 - ◆ Quadrics (Elan), IBM/SP (LAPI), Myrinet (GM), Infiniband, UDP, Shem* (Altix and X1), Dolphin* (SCI), and MPI
 - ◆ Shared with Berkeley UPC compiler
- ◆ Recent language and compiler work
 - Indexed (scatter/gather) array copy
 - Non-blocking array copy*
 - Loop level cache optimizations
 - Inspector/Executor*

* Work is still in progress

March 5, 2004 CS267 Lecture 12 48

Programmability

- ◆ Immersed boundary method developed in ~1 year
 - Extended to support 2D structures ~1 month
 - Reengineered over ~6 months
- ◆ Preliminary code length measures
 - Simple torus model
 - ◆ Serial Fortran torus code is 17045 lines long (2/3 comments)
 - ◆ Parallel Titanium torus version is 3057 lines long.
 - Full heart model
 - ◆ Shared memory Fortran heart code is 8187 lines long
 - ◆ Parallel Titanium version is 4249 lines long.
 - Need to be analyzed more carefully, but not a significant overhead for distributed memory parallelism

March 5, 2004

CS267 Lecture 12

49

Titanium and UPC Project Ideas

- ◆ Past 267 project ideas
 - Tree-based N-Body code in Titanium
 - Finite element code in Titanium
- ◆ Future project ideas for Titanium and UPC
 - Splash benchmarks in either language
 - Missing NAS benchmarking in Titanium
 - Your favorite application
- ◆ What makes it interesting?
 - Understanding the performance and scalability
 - ◆ Why does it perform as it does?
 - ◆ Performance model
 - ◆ Effectiveness of optimizations in application, runtime, compiler?

March 5, 2004

CS267 Lecture 12

50

Titanium Group (Past and Present)

- ◆ Susan Graham
 - ◆ Katherine Yelick
 - ◆ Paul Hilfinger
 - ◆ Phillip Colella (LBNL)
 - ◆ Alex Aiken
 - ◆ Ben Liblit
 - ◆ Peter McQuorquodale (LBNL)
 - ◆ Sabrina Merchant
 - ◆ Carleton Miyamoto
 - ◆ Chang Sun Lin
 - ◆ Geoff Pike
 - ◆ Luigi Semenzato (LBNL)
 - ◆ Armando Solar-Lezama
 - ◆ Jimmy Su
 - ◆ Tong Wen (LBNL)
 - ◆ Siu Man Yau
 - ◆ and many undergraduate researchers
 - ◆ Greg Balls
 - ◆ Andrew Begel
 - ◆ Dan Bonachea
 - ◆ Kaushik Datta
 - ◆ David Gay
 - ◆ Ed Givelberg
 - ◆ Arvind Krishnamurthy
- <http://titanium.cs.berkeley.edu>

March 5, 2004

CS267 Lecture 12

51