



LIQUID METAL

Taming Heterogeneity

Stephen Fink
IBM Research

INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE

gotocon.com

Liquid Metal Team (IBM T. J. Watson Research Center)



Josh Auerbach



David Bacon



Ioana Baldini



Perry Cheng



Stephen Fink

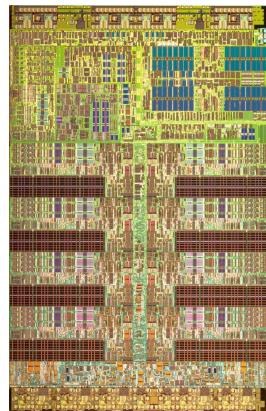


Rodric Rabbah

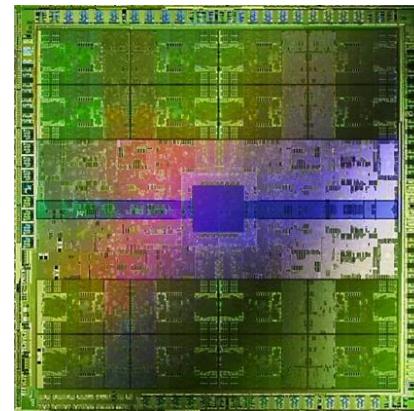


Sunil Shukla

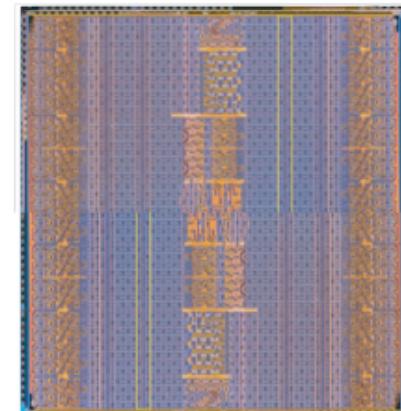
A Heterogeneous Landscape



Multicore



GPU

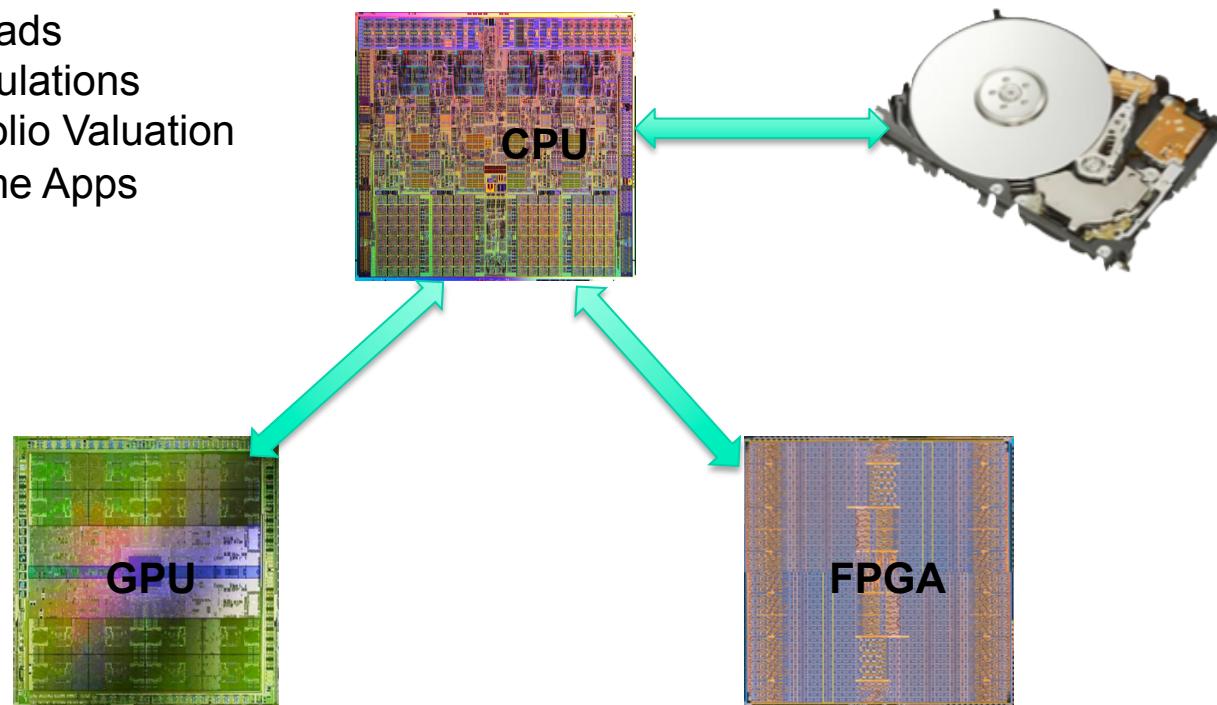


FPGA

Accelerator Platform

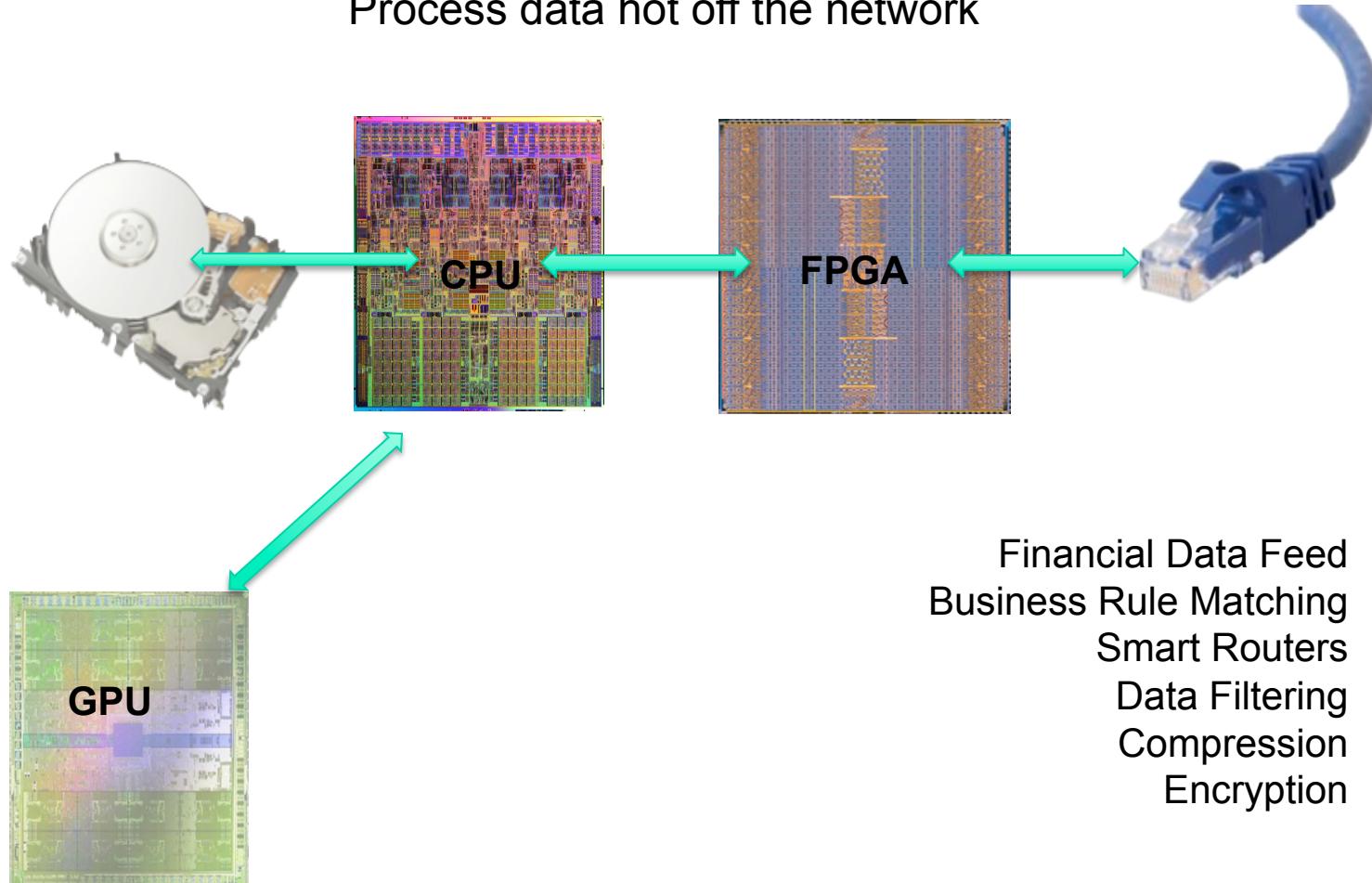
Computation starts from CPU

HPC Workloads
Seismic Simulations
Offline Portfolio Valuation
Non Real-time Apps



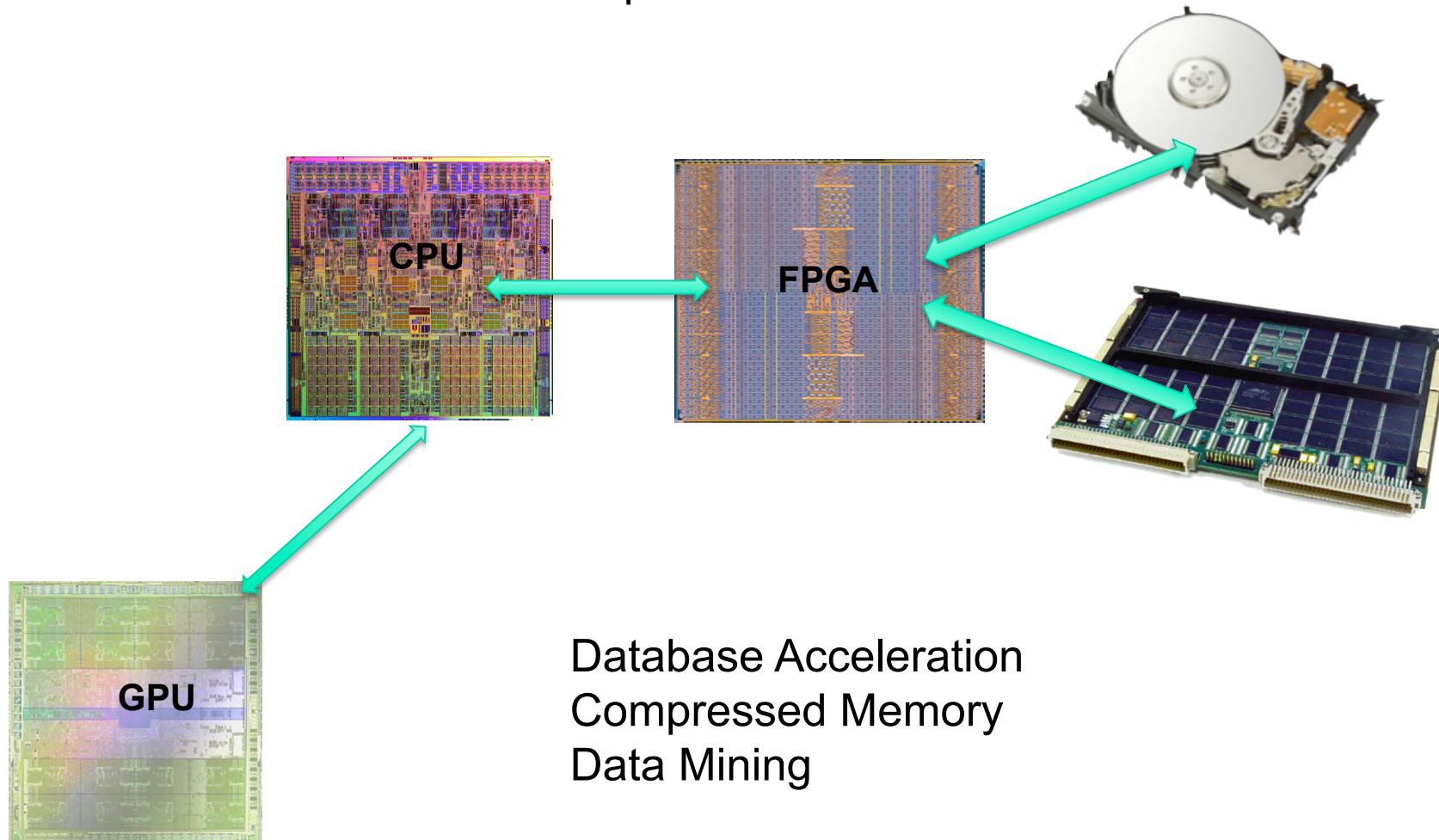
Network Appliance Platform

Process data hot off the network

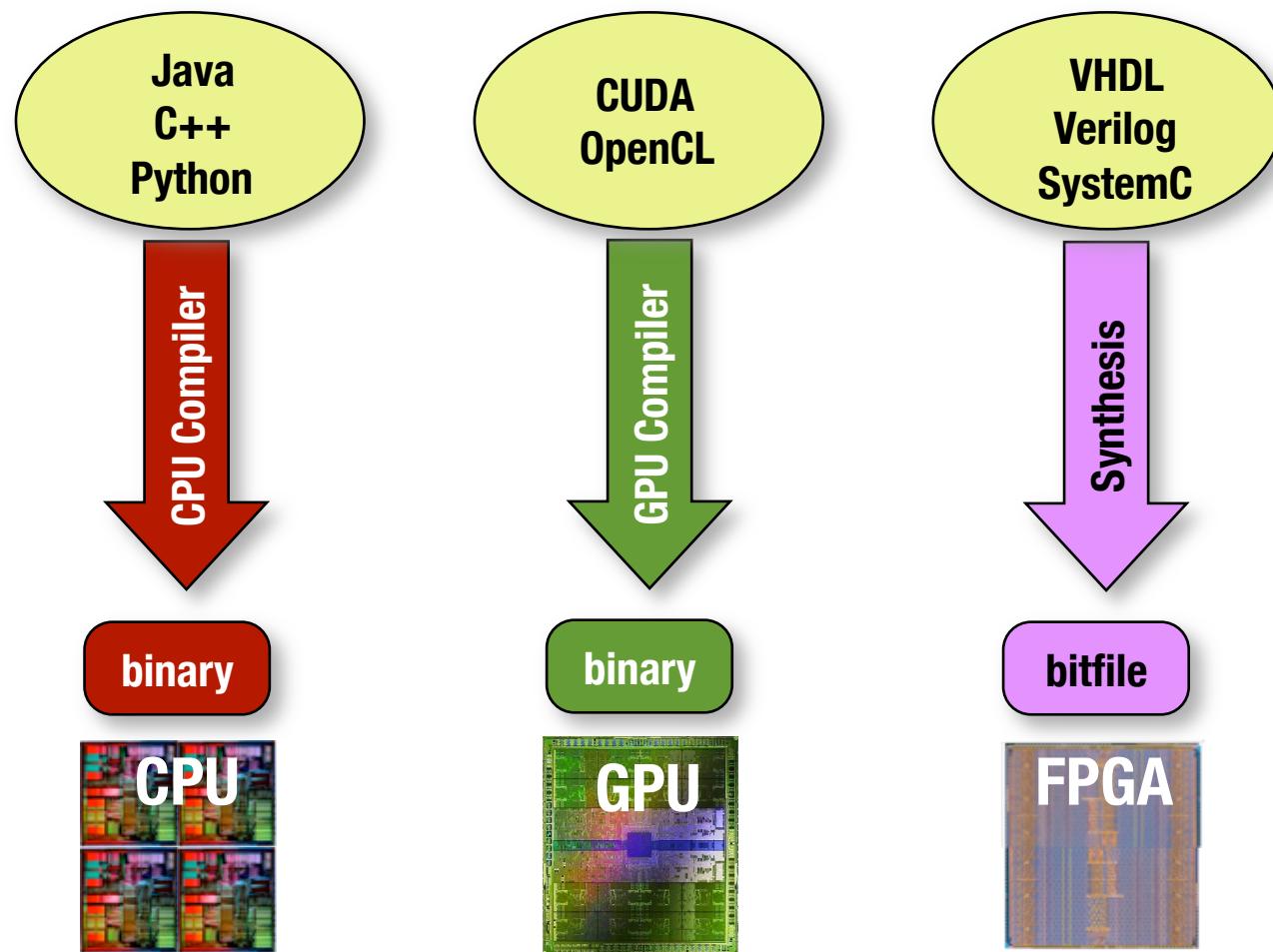


Storage Appliance Platform

Push computation near data

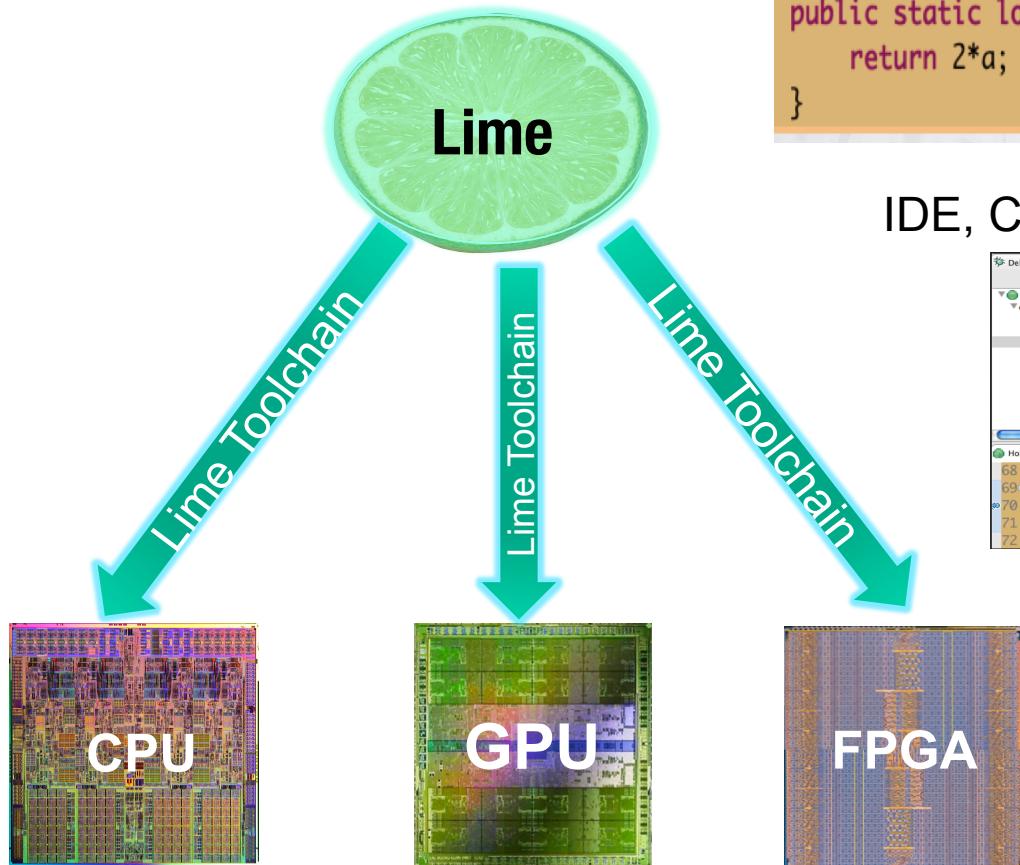


Heterogeneous Programming: Current Reality



Goal: Make Heterogeneous Platforms Accessible for Mainstream Use

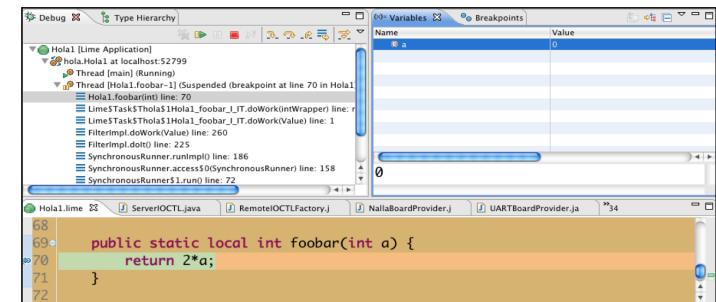
Vision:



High-Level Java-Derived Language

```
public static local int foobar(int a) {  
    return 2*a;  
}
```

IDE, Compiler, Debugger, Cloud Synthesis



Platform:
Hardware + Runtime

Export Model

Development for customer-owned platform



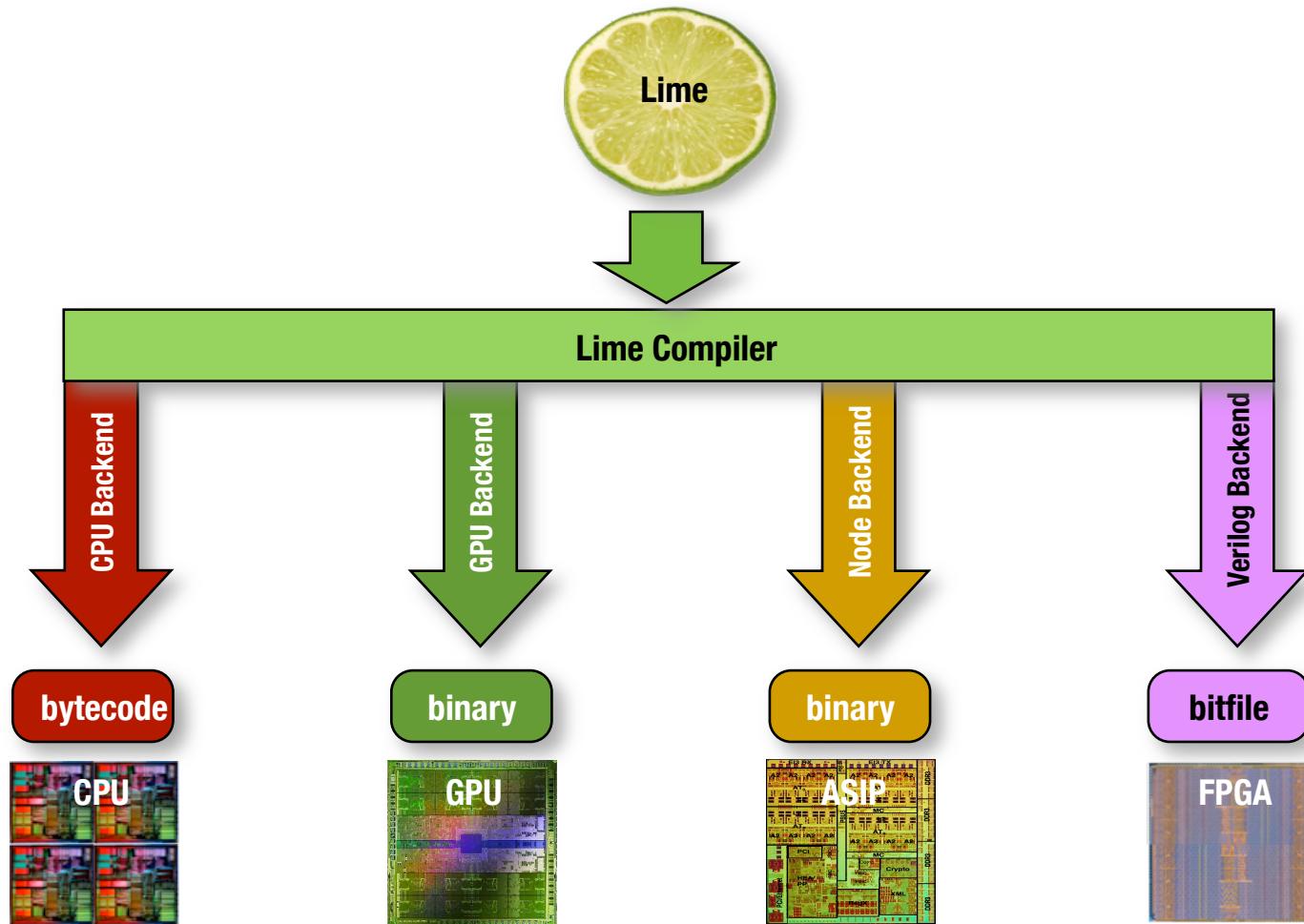
```
always @(posedge clk_22) begin
    if (reset_23) begin
        pc_30      <= 32'sd0;
        outReady_27 <= 1'd0;
        inReq_25   <= 1'd0;
    end else begin
        case (pc_30)
            0 : begin
                outReady_27 <= 1'd0;
                if (inReady_24) begin
                    inReq_25 <= 1'd1;
                    pc_30    <= 3'd1;
                end
            end
            1 : begin
                inReq_25 <= 1'd0;
                pc_30    <= 3'd2;
            end
            2 : begin
                a_31      <= inData_26[31:0];
                inReq_25 <= 1'd0;
                pc_30    <= 3'd3;
            end
            3 : begin
                multTmp_32 <= 32'sd2 * a_31;
                pc_30    <= 3'd4;
            end
        endcase
    end
end
```





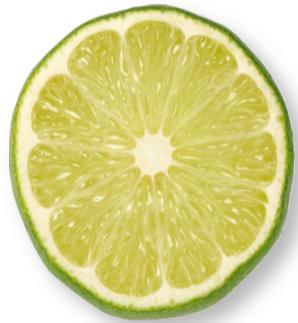
Lime

The Liquid Metal Programming Language



Compiler Philosophy:

- **Bytecode backend** compiles the full language
- Each **device backend** compiles a subset of the language
- Compiler gives specific feedback on “exclusions”



What is Lime?

Lime = Java + Isolation + Abstract Parallelism

Isolation: ability to move computation

- local** keyword

- Immutable types

Abstract Parallelism: freedom to schedule

- Stream programming model (**task, =>**)

- Data parallel constructs (**@ map, ! reduce**)

The Lime Development Experience

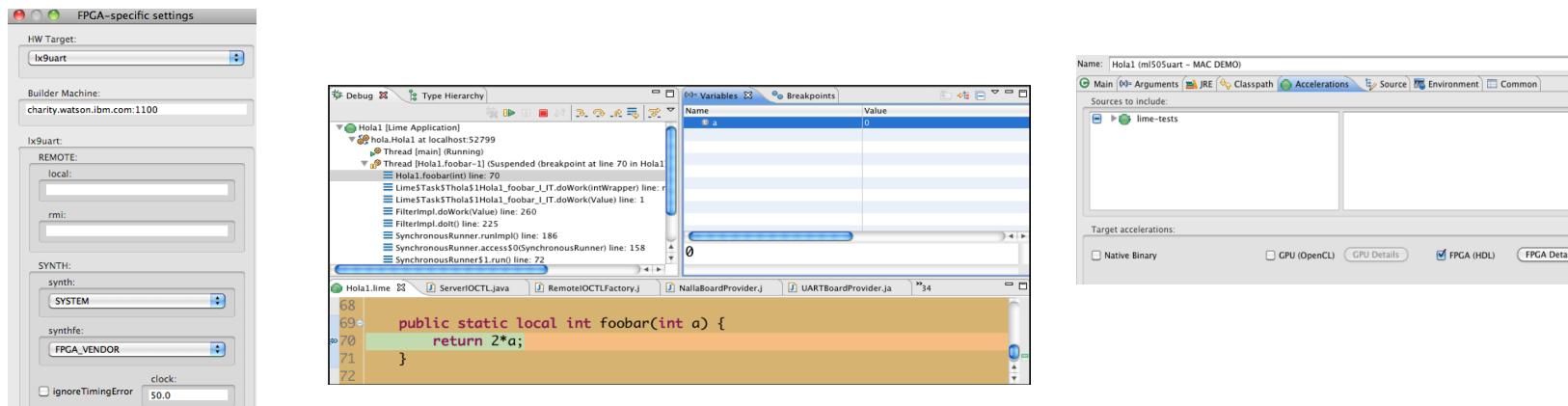
1. Prototype in standard Java

- Lime is a superset of Java
- **Java-like Eclipse IDE (editors, debugger, navigation)**

2. Gentle, incremental migration to parallel Lime code

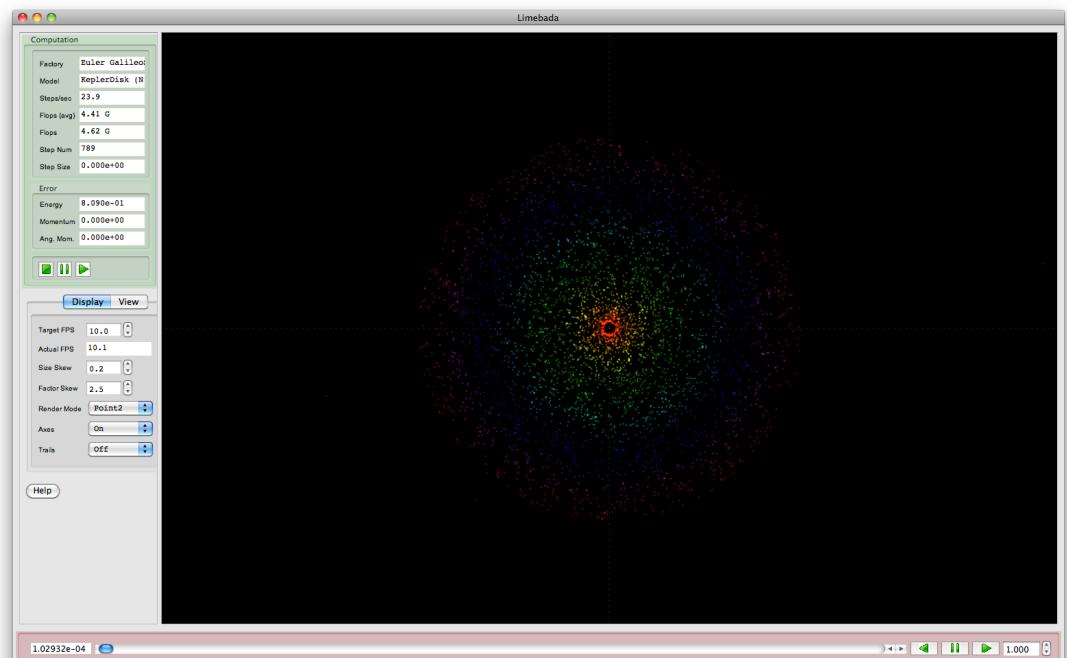
Lime language constructs restrict program to safe parallel structures

- Isolation
- Immutability
- Safe Deterministic Parallelism
- Bounded Space



Example Program: N-body simulation

1. Compute forces using particle states
2. Update particles based on forces
3. Periodic checkpoint to disk or display



Task and Connect



```
class NBody {  
    static void simulate() {  
        float[][][] particles = ...; // initial state  
  
        Task nbody =  task NBody(particles).particleGenerator  
                    => task forceComputation  
                    => task NBody(particles).forceAccumulator;  
        nbody.finish();  
    }  
  
    float[][][] particles;  
  
    NBody(float[][][] particles) { this.particles = particles; }  
    static local float[][][] forceComputation(float[][][] particles) {...}  
    float[][][] particleGenerator() {...}  
    void forceAccumulator(float[][][] particles) {...}  
}
```

Relocation Brackets



```
class NBody {  
    static void simulate() {  
        float[][][] particles = ...; // initial state  
  
        Task nbody =  task NBody(particles).particleGenerator  
                    => ([ task forceComputation ])  
                    => task NBody(particles).forceAccumulator;  
        nbody.finish();  
    }  
  
    float[][][] particles;  
  
    NBody(float[][][] particles) { this.particles = particles; }  
    static local float[][][] forceComputation(float[][][] particles) {...}  
    float[][][] particleGenerator() {...}  
    void forceAccumulator(float[][][]){...}  
}
```

```
// compute the force on each particle
static local float [[[3]]] forceComputation(float [[[4]]] particles) {
    return @ force(particles, #particles); // map operator @
}

// compute the force on particle p
static local float [[3]] force(float [[4]] p, float [[[4]]] P) {
    float fx = 0, fy = 0, fz = 0;
    for (float [[4]] Pj : P) {
        float dx = Pj[0] - p[0];
        float dy = Pj[1] - p[1];
        float dz = ...;
        float F = ...; // scalar code, compute force between p and Pj
        fx += dx + F;
        fy += dy + F;
        fz += dz + F;
    }
    return new float [] { fx, fy, fz };
}
```

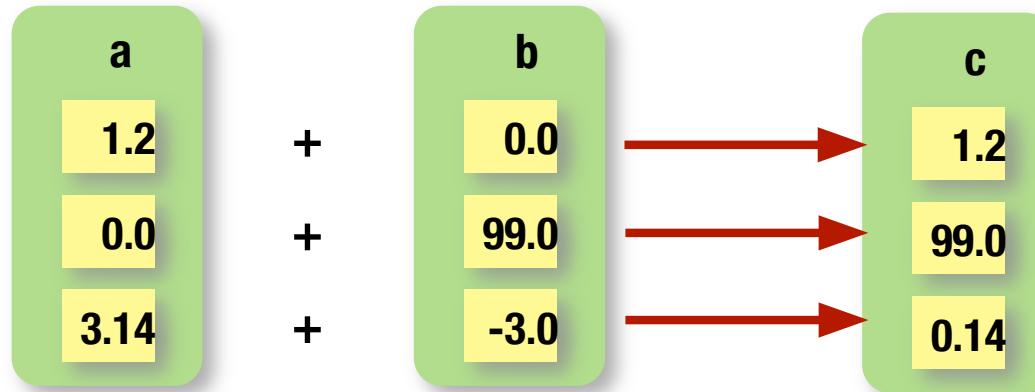


Language constructs guarantee safe, deterministic, parallel execution semantics

- Deeply immutable types (e.g. **float**[])
- Isolation (e.g. **local**)
- Data parallel function application (**map**, **@**)

Data Parallelism via map (@)

```
float[] a = ...;  
float[] b = ...;  
float[] c = a @+ b; // map operator + over 2 arrays
```



```
static local float add(float x, float y) { return x + y; }  
float[] c = @add(a, b); // map function add() over 2 arrays  
float[] d = @add(a, #1); // map function add() over 1 array, 1 scalar
```

What didn't you write? (GPU)

The image shows three side-by-side code editors, each displaying a portion of an OpenCL kernel. The code is written in C and is part of a Java application, as indicated by the file names in the tabs: 'GalileoSingleAccCalc.java', 'EulerIntegrator.java', and 'limebada_calculator_.java'. The code is annotated with several comments and includes error checking.

```

/*
 * Class:      limebada_calculator_OCLAccCalculator
 * Method:    computeAccOCL
 * Signature: (ID[D][D])
 */
JNIEXPORT jlong JNICALL Java_limebada_calculator_OCLAccCalculator_computeAccOCL(JNIEnv *env, jobject obj, jint deviceIndex, jdouble softenLength)
{
    struct timeval start, end;
    initializeFIDs(env, obj);

    // Setup various buffers if this is the first time we enter here
    // Note that we are using our own copy via Java-hosted field.
    int numParticles = env->GetArrayLength(partData) / 4;
    cl_float4 * positions = (cl_float4 *) env->GetLongField(obj, posMemID);
    float* accs = (float *) env->GetLongField(obj, accMemID);
    Compute_t *compute = (Compute_t *) env->GetLongField(obj, oclInfoID);

    if (positions == NULL) {
        // It needs to be big enough to contain doubles as we unpack.
        int positionSpace = 4 * numParticles * sizeof(double);
        int accSpace = 3 * numParticles * sizeof(double); // enough
        positions = (cl_float4 *) malloc(positionSpace);
        accs = (float *) malloc(accSpace);
        compute = setupCompute(numParticles, deviceIndex);
        env->SetLongField(obj, posMemID, (long) positions);
        env->SetLongField(obj, accMemID, (long) accs);
        env->SetLongField(obj, oclInfoID, (long) compute);
    }

    gettimeofday(&start, NULL);
    // Copy data over and then squeeze it down to float
    env->GetDeviceByteArrayRegion(partData, 0, 4 * numParticles, (double*)positions);
    for (int i=0, i<4*numParticles; i++) {
        ((cl_float*)positions)[i] = (cl_float) ((double *)positions[i]);
    }
    gettimeofday(&end, NULL);
    marhsalTime += ((end.tv_sec * 1000000 + end.tv_usec) - (start.tv_sec * 1000000 + start.tv_usec)) / 1000000.0;

    float softenLenSquared = softenLength * softenLength;
    calcAcc(compute, numParticles, positions, accs, softenLenSquared);

    cl_ulong startTime;
    cl_ulong endTime;

    int retCode = clGetEventProfilingInfo(kernelComputationEvent, CL_PROFILING_COMMAND_START);
    checkError(retCode, "clGetEventProfilingInfo");
    retCode = clGetEventProfilingInfo(kernelComputationEvent, CL_PROFILING_COMMAND_END);
    checkError(retCode, "clGetEventProfilingInfo");

    kernelTime += endTime - startTime;

    retCode = clGetEventProfilingInfo(firstTransferEvent, CL_PROFILING_COMMAND_START);
    checkError(retCode, "clGetEventProfilingInfo");
    retCode = clGetEventProfilingInfo(lastTransferEvent, CL_PROFILING_COMMAND_END);
    checkError(retCode, "clGetEventProfilingInfo");

    totalOclTime += endTime - startTime;
    calledNUM++;
}
}

// Performance notes:
// (1) rsqrt seems just as fast as native_sqrt
// (2) Using float4 throughout will cause extra
static const char *Source =
"float4 calcAccHelp(float softLenSq, float4 acc,
{
    // 3 flops
    float dx = pj.x - pi.x;
    float dy = pj.y - pi.y;
    float dz = pj.z - pi.z;
    // 10 flops
    float d2 = dx*dx + dy*dy + dz*dz + softLenSq;
    float id = rsqrt(d2);
    float f = pj.w*id*id*id;
    // 6 flops
    acc.x += f * dx;
    acc.y += f * dy;
    acc.z += f * dz;
    return acc;
}

__kernel void calcAccKernel(
    __global float4* positions,
    __global float4* accs,
    __local float4* localPos,
    float softLenSq) {
    int gl = get_global_id(0);
    int ti = get_local_id(0);
    int gs = get_global_size(0);
    int ls = get_local_size(0);
    int blks = gs / ls;
    float4 pi = positions[gl];
    float4 acc = {0,0,0,0};
    int baseIndex = 0;
    for (int b=0; b<blks; b++) {
        localPos[ti] = positions[baseIndex+ti];
        barrier(CLK_LOCAL_MEM_FENCE);
        for (int j=0; j<ls; j++) {
            float4 pj = localPos[j];
            acc = calcAccHelp(softLenSq, acc, pi, pj);
        }
        baseIndex += ls;
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    accs[gl] = acc;
}
}

static Compute_t* setupCompute(int N, int deviceIndex) {
    Compute_t* compute = (Compute_t*) malloc(sizeof(Compute_t));
    // Setup device, compile kernel, and create communication buffers
    compute->info = setupGPU(0, deviceIndex, Source, "calcAccKernel");
    compute->input = makeBuffer(compute->info, CL_MEM_READ_ONLY, sizeof(cl_float4) * N);
    compute->output = makeBuffer(compute->info, CL_MEM_WRITE_ONLY, sizeof(cl_float4) * N);
    return compute;
}

static void calcAcc(Compute_t *comp, int N, cl_float4 *pos, cl_float4 *accs,
{
    // We can't let softenLen be zero or else we can't let a body self-interact
    // This is not softening in the sense of preventing close-body encounter
    if (softenLenSquared == 0)
        softenLenSquared = 1e-24f; // will compute -1.0 power of this

    int err;
    cl_command_queue queue = comp->info->queue;
    cl_mem input = comp->input;
    cl_mem output = comp->output;
    cl_kernel kernel = comp->info->kernel;

    err = clEnqueueWriteBuffer(queue, input, CL_TRUE, 0, sizeof(cl_float4) * N, checkError(err, "clEnqueueWriteBuffer"));

    err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
    err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
    err |= clSetKernelArg(kernel, 2, comp->info->maxWorkGroupSize * sizeof(cl_float4));
    err |= clSetKernelArg(kernel, 3, sizeof(float), &softenLenSquared);
    checkError(err, "clSetKernelArg");

    // one-dimensional
    size_t global = N;
    size_t local = 1;
    err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global, NULL, 0, N, checkError(err, "clEnqueueNDRangeKernel"));

    err = clEnqueueReadBuffer(queue, output, CL_TRUE, 0, sizeof(cl_float4) * N, checkError(err, "clEnqueueReadBuffer"));

    err = clFinish(queue);
    checkError(err, "clFinish");
}
}

```

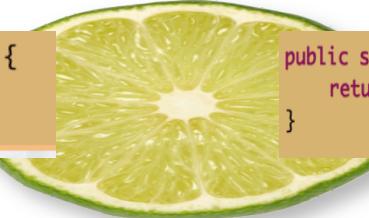
What didn't you write? (FPGA)

User Logic

```
public static local int foobar(int a) {
    return 2*a;
}
```

Platform runtime and communication

```
public static Task makeTask1() {
    return ( task Hola1().source => task Hola1.foobar => task Hola1.printer );
}
```



```
always @(posedge clk_22) begin
    if (reset_23) begin
        pc_30  <=  32'sd0;
        outReady_27  <=  1'd0;
        inReq_25  <=  1'd0;
    end else begin
        case (pc_30)
            0 : begin
                outReady_27  <=  1'd0;
                if (inReady_24) begin
                    inReq_25  <=  1'd1;
                    pc_30  <=  3'd1;
                end
            end
            1 : begin
                inReq_25  <=  1'd0;
                pc_30  <=  3'd2;
            end
            2 : begin
                a_31  <=  inData_26[31:0];
                inReq_25  <=  1'd0;
                pc_30  <=  3'd3;
            end
            3 : begin
                multTmp_32  <=  32'sd2 * a_31;
                pc_30  <=  3'd4;
            end
        end
    end
}
```

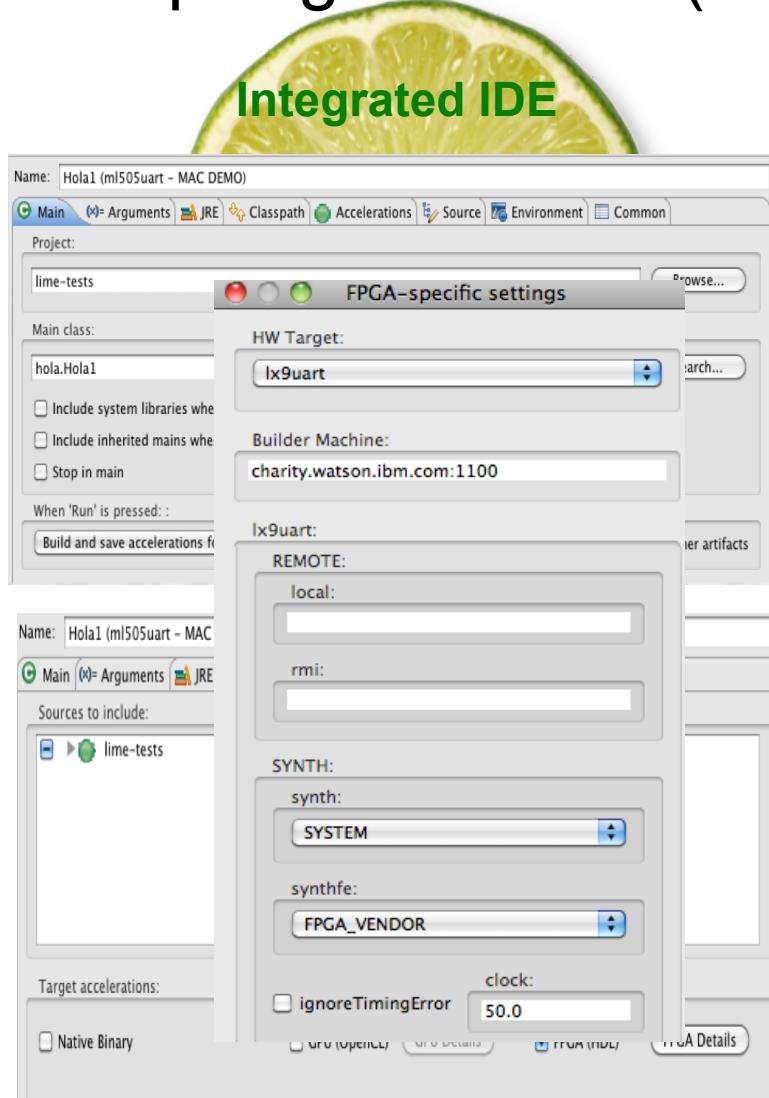
HDL

3000 lines of platform independent HDL
 500 lines of platform dependent HDL
 3 months of development time
 ??? months of debugging time



Lime Developer Tools

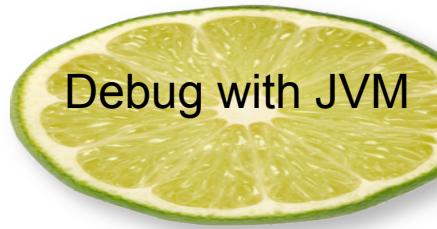
Compiling the Code (FPGA)



Multiple 3rd party tools with long workflow

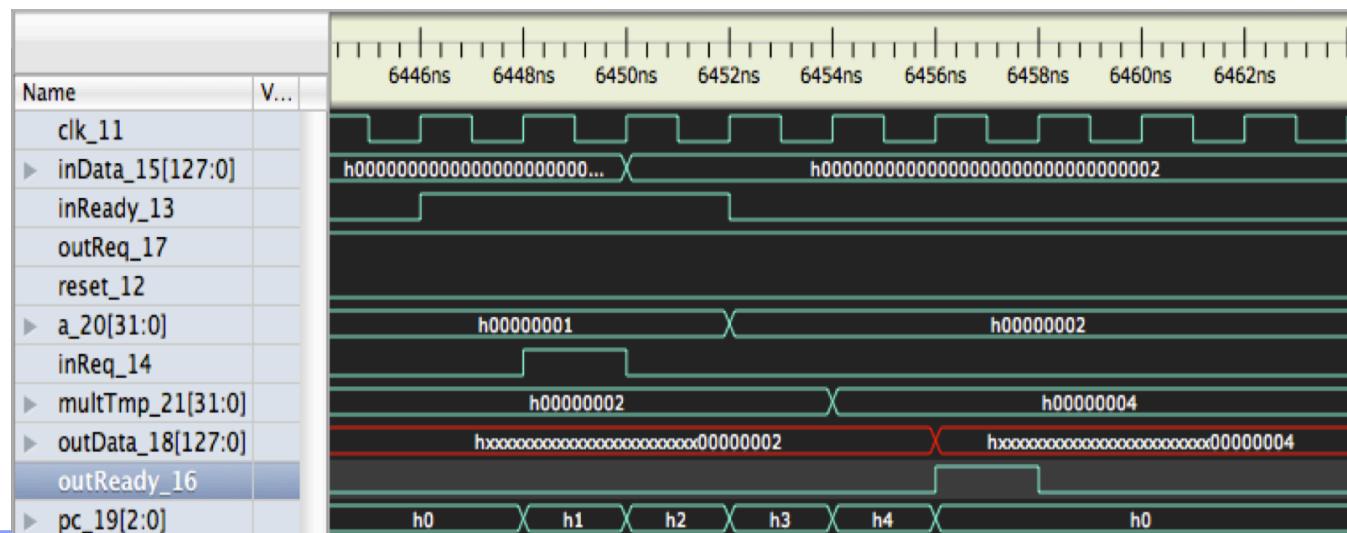
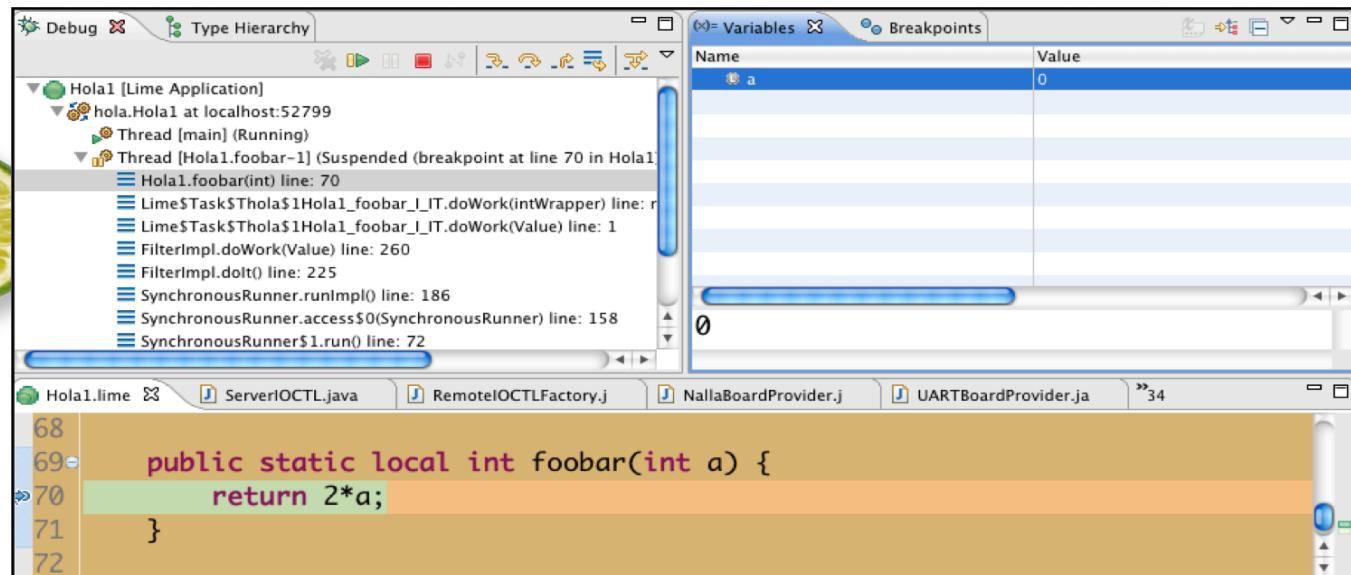
1. Create an ISE® or Quartus® project
2. Write HDL and testbench
3. Simulate design
4. Repeat 2 & 3 until functionality is achieved
5. Implement your top module
6. Follow advanced design guides
7. Modify the design as necessary, simulate, synthesize, and implement until design requirements are met.
8. Run timing simulations, verify timing closures
9. Program your Xilinx® or Altera® device

Debugging

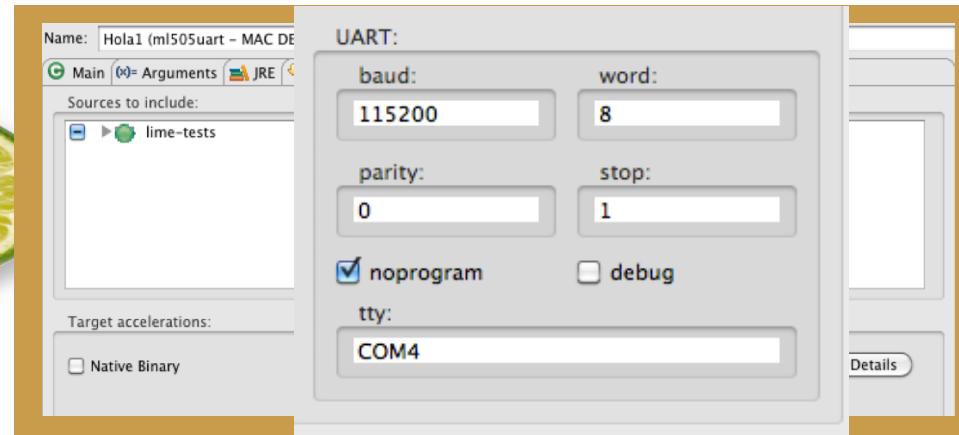
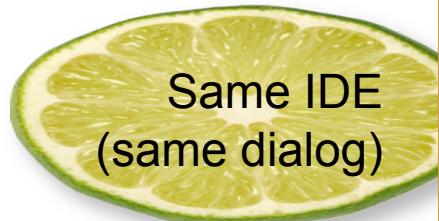


vs.

Low-level
Waveforms

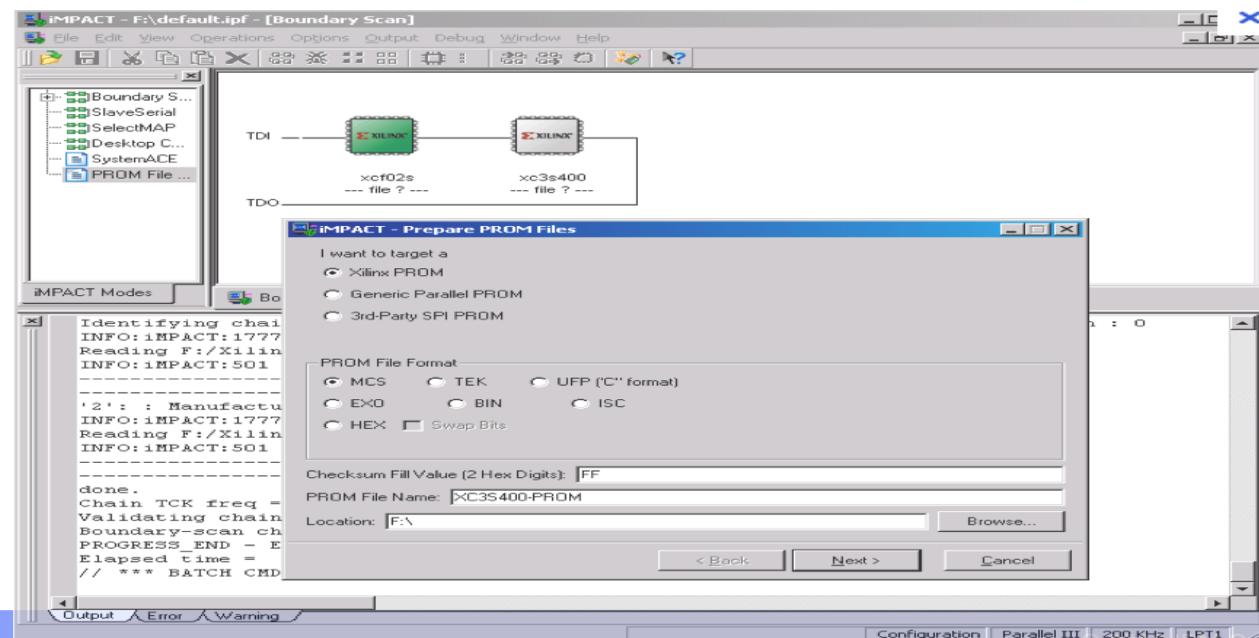


Running the Code

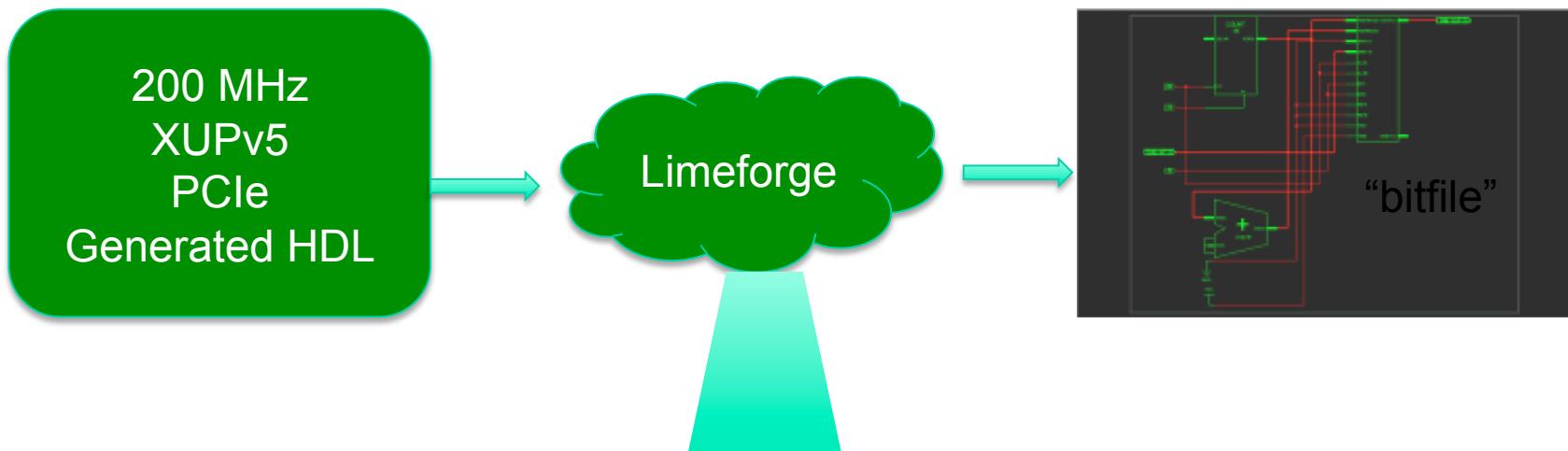


VS.

Yet
Another
Tool



Limeforge (Synthesis in the Cloud)



Lime Builder Service – Mozilla Firefox: IBM Edition
http://lyric.watson.ibm.com:8080/

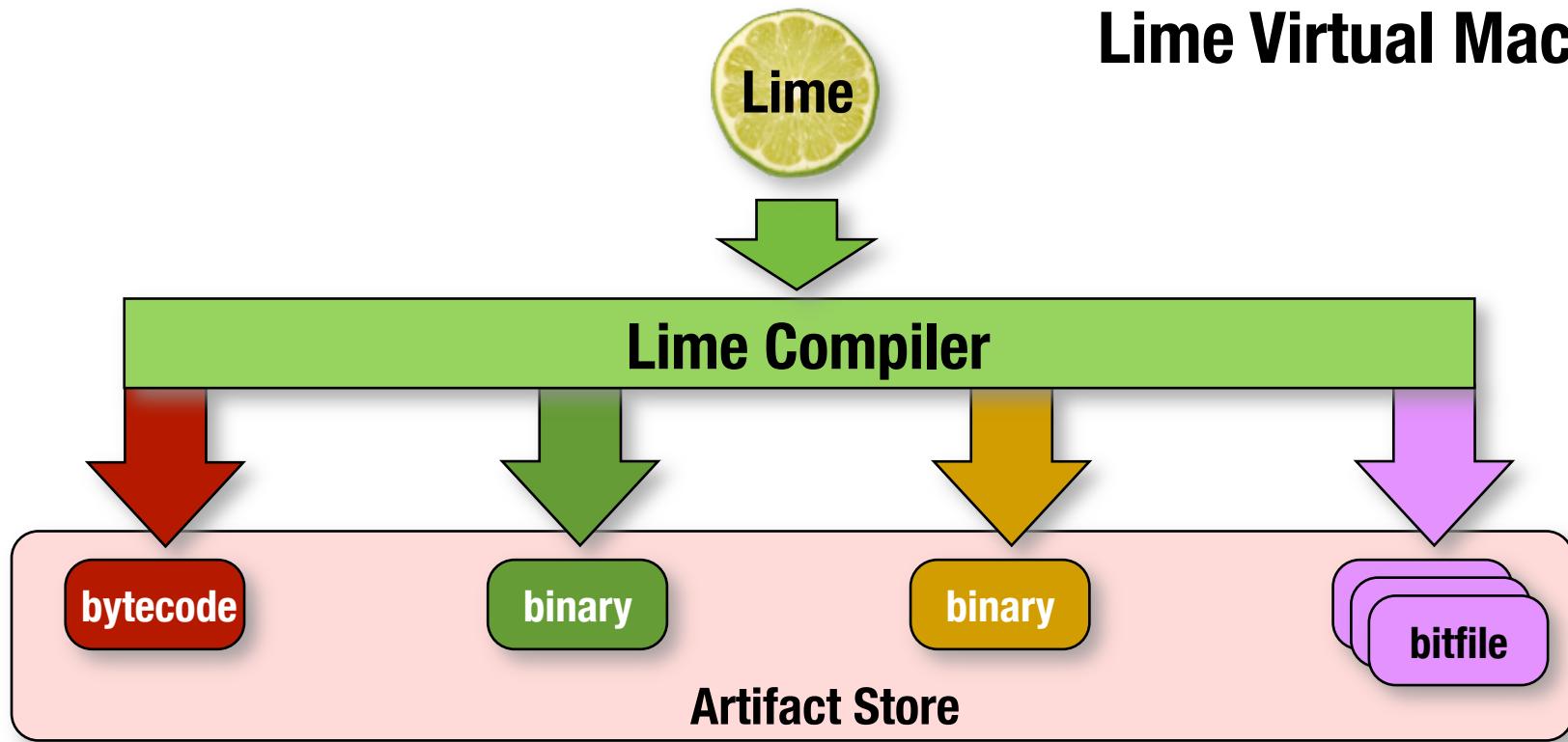
Overall Job Status

Job ID	Job Name	Start	Submitter	Task ID	Board	Target	Action	Status	Phase	Job Time	Progress	Result
1004	Hola5.makeTask5	07/12 11:04	perry@dyn9002034247	3777144747	nallapcie280	225.0 Mhz	<input type="button" value="KILL"/>	Ongoing	Map	7.0 min	25%	null
1003	Hola4.makeTask4	07/12 10:59	perry@dyn9002034247	1097401331	nallapcie280	150.0 Mhz	<input type="button" value="KILL"/>	Ongoing	Par	12.7 min	60%	null
1002	Hola1.makeTask1	07/12 10:59	perry@dyn9002034247	1057238476	nallapcie280	200.0 Mhz	<input type="button" value="KILL"/>	Ongoing	Par	12.7 min	60%	null
1001	Hola1.makeTask1	07/11 17:49	perry@dyn9002034247	1057238476	nallapcie280	200.0 Mhz	<input type="button" value="Completed"/>	Completed	unknown	4.7 min	100%	Error



Lime Implementation

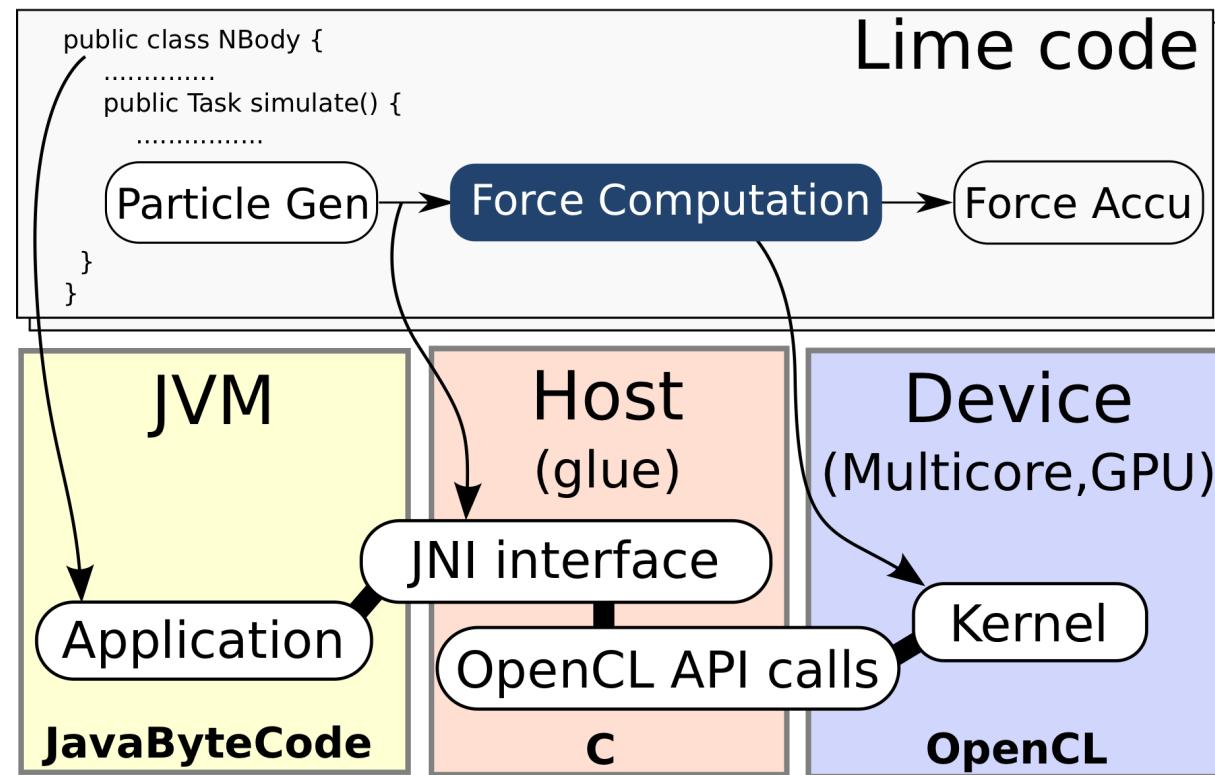
Lime Virtual Machine



Lime Virtual Machine (LVM)



GPU Backend Structure



How do we evaluate performance?

- **Speedup for Naïve Users**

How much faster than Java?

- **Slowdown for Expert Users**

How much slower than hand-tuned low-level code?

Current status (bottom line)

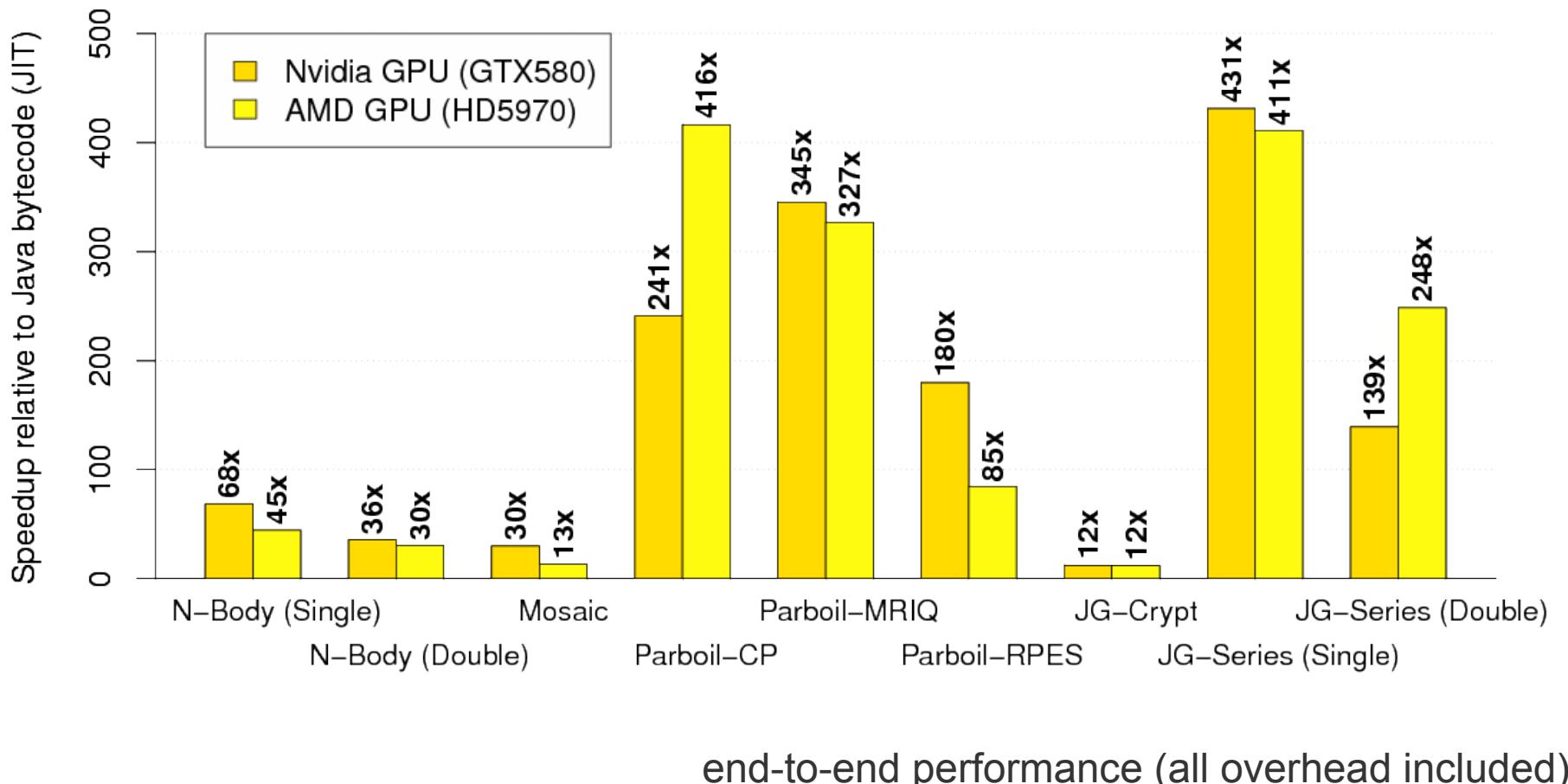
No magical super-compiler

- Some benchmarks Lime performs as well as hand-coded alternative
- Some, not.
- Communication/computation ratio dominates

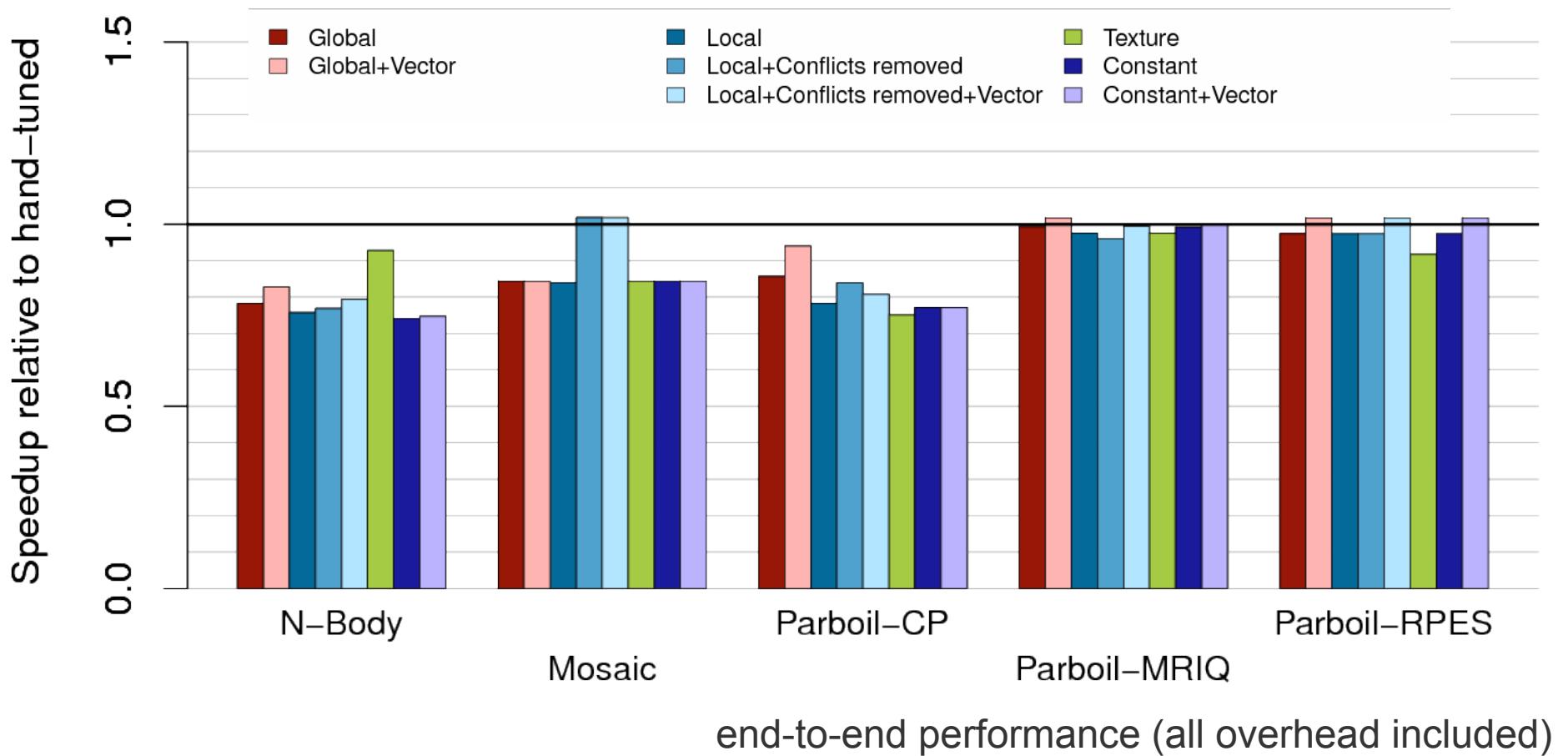
Ongoing research into better implementation technology

- Big performance improvements in reach
- Good enough for prime time? We're optimistic.

Speedup vs. Java [PLDI '12]



Speedup vs. Hand-Tuned OpenCL [PLDI '12] (GTX 580)



Lime FPGA Results

- **Combinatorial circuits: match hand-coded**
 - pipelining
- **Complex circuits: goal is 2-4x worse, now $\geq 10x$**
 - I/O pipelining and classic compiler optimizations
- **Increasing synthesizable language feature set**
 - First of its kind FPGA garbage collector

Liquid Metal Summary

- ✓ High-level, concise abstractions for parallelism
- ✓ Gentle, incremental migration from Java
- ✓ Rich Eclipse-based tool support
- ✓ General solution for programming heterogeneous systems



No magic

- parallel algorithm design is not easier
- one version of a program may not run well on all devices
- vanilla Java programs will not run on accelerator

<http://www.research.ibm.com/liquidmetal/>

Backup

Export Model

Development for customer-owned platform



```
always @(posedge clk_22) begin
    if (reset_23) begin
        pc_30      <= 32'sd0;
        outReady_27 <= 1'd0;
        inReq_25   <= 1'd0;
    end else begin
        case (pc_30)
            0 : begin
                outReady_27 <= 1'd0;
                if (inReady_24) begin
                    inReq_25 <= 1'd1;
                    pc_30    <= 3'd1;
                end
            end
            1 : begin
                inReq_25 <= 1'd0;
                pc_30    <= 3'd2;
            end
            2 : begin
                a_31      <= inData_26[31:0];
                inReq_25 <= 1'd0;
                pc_30    <= 3'd3;
            end
            3 : begin
                multTmp_32 <= 32'sd2 * a_31;
                pc_30    <= 3'd4;
            end
        endcase
    end
end
```

