THE PARALLEL JAVA 2 LIBRARY

Multicore Parallel Programming in 100% Java

```
package edu.rit.pj2.example;

import edu.rit.image.Color;
import edu.rit.image.ColorArray;
import edu.rit.image.ColorImageQueue;
import edu.rit.image.ColorPngWriter;
import edu.rit.pj2.Loop;
import edu.rit.pj2.Schedule;
import edu.rit.pj2.Section;
import edu.rit.pj2.Task;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;

public class MandelbrotSmp2
   extends Task
   {

   // Command line arguments.
   int width;
   int height;
   double xcenter;
   double ycenter;
   double resolution;
   int maxiter;
   double gamma;
   File filename;

   // Initial pixel offsets from center.
   int xoffset;
   int yoffset;

   // Table of hues.
   Color[] huetable;

   // For writing PNG image file.
   ColorPngWriter writer;
   ColorImageQueue imageQueue;

   // Mandelbrot Set main program.
   public void main
      (String[] args)
      throws Exception
      {
      // Validate command line arguments.
      if (args.length != 8) usage();
      width = Integer.parseInt (args[0]);
      height = Integer.parseInt (args[1]);
      xcenter = Double.parseDouble (args[2]);
      ycenter = Double.parseDouble (args[3]);
      resolution = Double.parseDouble (args[4]);
      maxiter = Integer.parseInt (args[5]);
      gamma = Double.parseDouble (args[6]);
      filename = new File (args[7]);

      // Initial pixel offsets from center.
      xoffset = -(width - 1) / 2;
      yoffset = (height - 1) / 2;

      // Create table of hues for different iteration counts.
      huetable = new Color [maxiter + 2];
      for (int i = 1; i <= maxiter; ++ i)
         huetable[i] = new Color().hsb
            (/*hue*/ (float) Math.pow ((double)(i - 1)/maxiter, gamma),
             /*sat*/ 1.0f,
             /*bri*/ 1.0f);
      huetable[maxiter + 1] = new Color().hsb (1.0f, 1.0f, 0.0f);

      // For writing PNG image file.
      writer = new ColorPngWriter (height, width,
         new BufferedOutputStream (new FileOutputStream (filename)));
      imageQueue = writer.getImageQueue();

      // Overlapped computation and I/O.
      parallelDo (new Section()
         {
         public void run()
            {
            // Compute all rows and columns.
            parallelFor (0, height - 1)
               .schedule (dynamic) .exec (new Loop()
               {
               ColorArray pixelData;

               public void start()
                  {
                  pixelData = new ColorArray (width);
                  }

               public void run (int r) throws Exception
                  {
                  double y = ycenter + (yoffset - r) / resolution;
                  for (int c = 0; c < width; ++ c)
                     {
                     double x = xcenter + (xoffset + c) / resolution;

                     // Iterate until convergence.
                     int i = 0;
                     double aold = 0.0;
                     double bold = 0.0;
                     double a = 0.0;
                     double b = 0.0;
                     double zmagsqr = 0.0;
                     while (i <= maxiter && zmagsqr <= 4.0)
                        {
                        ++ i;
                        a = aold*aold - bold*bold + x;
                        b = 2.0*aold*bold + y;
                        zmagsqr = a*a + b*b;
                        aold = a;
                        bold = b;
                        }

                     // Record number of iterations for pixel.
                     pixelData.color (c, huetable[i]);
                     }

                  imageQueue.put (r, pixelData);
                  }
               });
            }
         },

      new Section()
         {
         public void run() throws Exception
            {
            // Write image to PNG file.
            writer.write();
            }
         });
      }

   }
```

Shared global variables

Sequential code

Parallel sections

Computation section

Work sharing parallel loop

Load balancing schedule

Per-thread variables

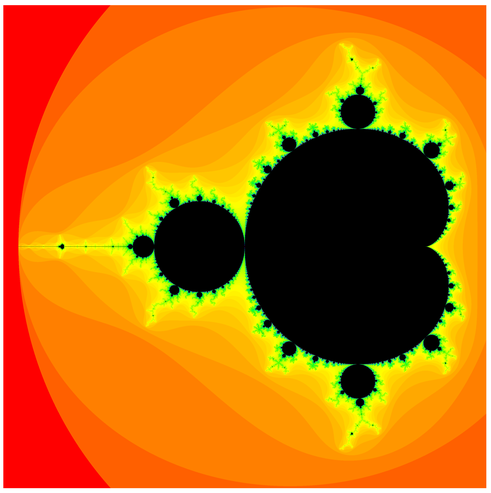
Per-thread initialization

Parallel loop body

Implicit barrier at end of parallel loop

Output section

Implicit barrier at end of parallel sections

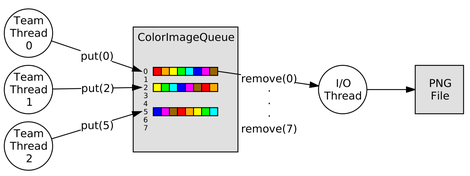


Computation of the Mandelbrot Set

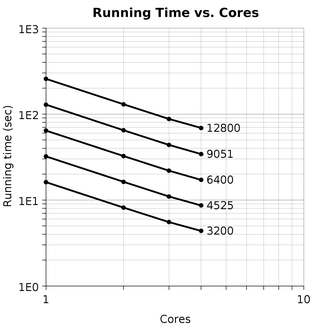
Features:

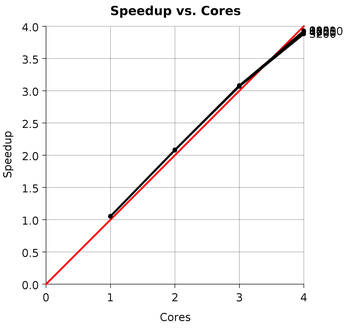
- **Overlapped computation and I/O**
- **Work sharing parallel loop**
- **Automatic load balancing**

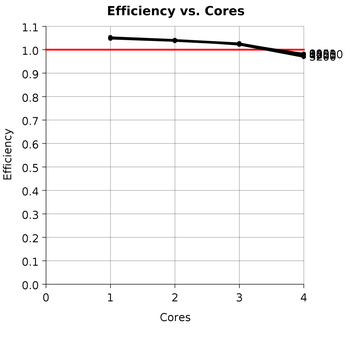
Design



Strong scaling performance (4 cores, 2.6 GHz)







Presented at the *International Conference for High Performance Computing, Networking, Storage and Analysis (SC14)*
New Orleans, Louisiana, USA
November 16–21, 2014

THE PARALLEL JAVA 2 LIBRARY

Cluster Parallel Programming in 100% Java

```
package edu.rit.pj2.example;

import edu.rit.pj2.Job;
import edu.rit.pj2.LongVbl;
import edu.rit.pj2.LongLoop;
import edu.rit.pj2.Task;
import edu.rit.util.Random;

public class PiClu
   extends Job
   {

   // Job main program.
   public void main
      (String[] args)
      {
      // Parse command line arguments.
      if (args.length != 2) usage();
      long seed = Long.parseLong (args[0]);
      long N = Long.parseLong (args[1]);

      // Set up a task group of K worker tasks.
      masterFor (0, N - 1, WorkerTask.class) .args (""+seed, ""+N);

      // Set up reduction task.
      rule() .atFinish() .task (ReduceTask.class) .args (""+N)
         .runInJobProcess();
      }

   // Print a usage message and exit.
   private static void usage()
      {
      System.err.println ("Usage: java pj2 [workers=<K>] " +
         "edu.rit.pj2.example.PiClu <seed> <N>");
      System.err.println ("<K> = Number of worker tasks (default: 1)");
      System.err.println ("<seed> = Random seed");
      System.err.println ("<N> = Number of random points");
      throw new IllegalArgumentException();
      }

   // Class PiClu.WorkerTask performs part of the computation for the
   // PiClu program.
   private static class WorkerTask
      extends Task
      {
      // Command line arguments.
      long seed;
      long N;

      // Number of points within the unit circle.
      LongVbl count;

      // Worker task main program.
      public void main
         (String[] args)
         throws Exception
         {
         // Parse command line arguments.
         seed = Long.parseLong (args[0]);
         N = Long.parseLong (args[1]);

         // Generate n random points in the unit square, count how many are
         // in the unit circle.
         count = new LongVbl.Sum (0);
         workerFor() .exec (new LongLoop()
            {
            Random prng;
            LongVbl thrCount;

            public void start()
               {
               prng = new Random (seed + 1000*taskRank() + rank());
               thrCount = threadLocal (count);
               }

            public void run (long i)
               {
               double x = prng.nextDouble();
               double y = prng.nextDouble();
               if (x*x + y*y <= 1.0) ++ thrCount.item;
               }
            });

         // Report results.
         putTuple (count);
         }
      }

   // Class PiClu.ReduceTask combines the worker tasks' results and prints
   // the overall result for the PiClu program.
   private static class ReduceTask
      extends Task
      {
      // Reduce task main program.
      public void main
         (String[] args)
         throws Exception
         {
         long N = Long.parseLong (args[0]);
         LongVbl count = new LongVbl.Sum (0L);
         LongVbl template = new LongVbl();
         LongVbl taskCount;
         while ((taskCount = tryToTakeTuple (template)) != null)
            count.reduce (taskCount);
         System.out.printf ("pi = 4*%d/%d = %.9f%n",
            count.item, N, 4.0*count.item/N);
         }
      }

   }
```

Parallel job with multiple parallel tasks

Master-worker parallel loop, master portion

Multicore worker task

Shared global variables

Global reduction variable

Master-worker parallel loop, worker portion

Per-thread PRNG

Per-thread counter

Parallel loop body

Update per-thread counter

Automatically reduce all per-thread counters Into global counter

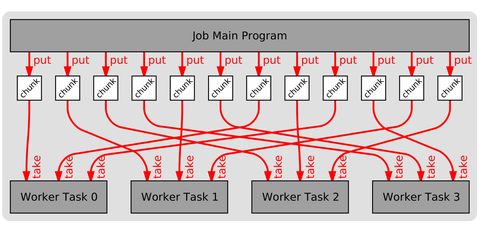
Send worker task's result to reduction task

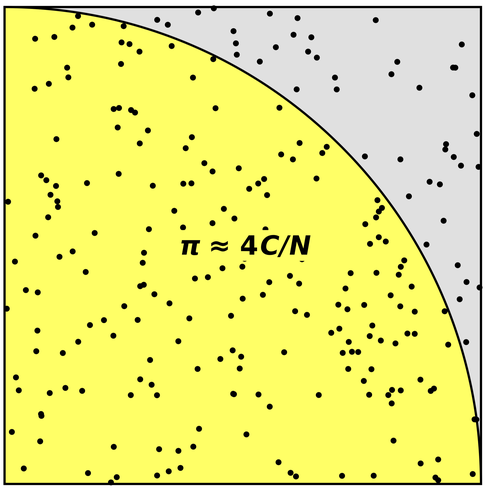
Single-core reduction task

Receive result from a worker task

Accumulate result

Master-worker cluster parallel loop



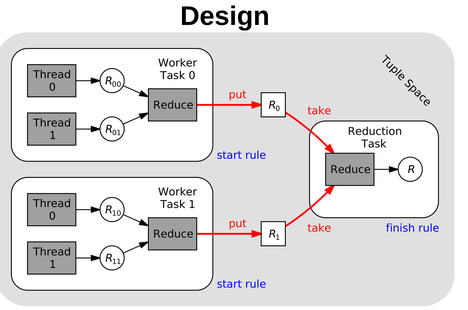


Monte Carlo estimation of π

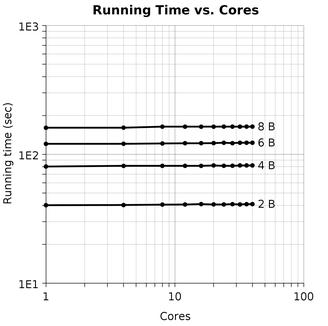
Features:

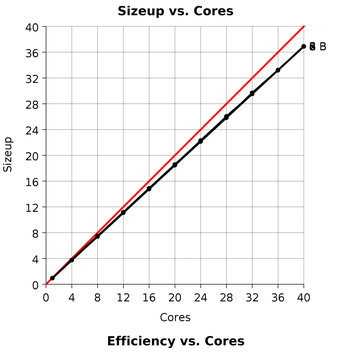
- One master process
- Multiple multicore worker processes, one per cluster node
- Interprocess communication via tuple space

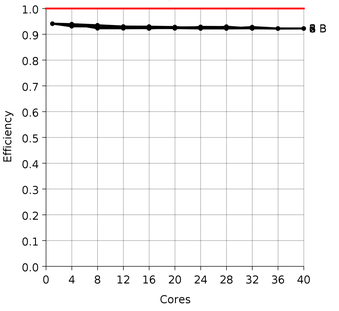
Design



Weak scaling performance (10 nodes, 40 cores, 2.6 GHz)







THE PARALLEL JAVA 2 LIBRARY

Map-Reduce Parallel Programming in 100% Java

```
package edu.rit.pjmr.example;

import edu.rit.pj2.LongVbl;
import edu.rit.pjmr.Combiner;
import edu.rit.pjmr.Customizer;
import edu.rit.pjmr.Mapper;
import edu.rit.pjmr.PjmrJob;
import edu.rit.pjmr.Reducer;
import edu.rit.pjmr.TextFileSource;
import java.util.Date;
import java.util.Scanner;

public class WebLog03
   extends PjmrJob<Long,String,IPAddress,LongVbl>
   {

   // PJMR job main program.
   public void main
      (String[] args)
      {
      if (args.length < 3) usage();
      int NT = threads() == DEFAULT_THREADS ? 1 : threads();
      System.out.printf
         ("$ java pj2 threads=%d edu.rit.pjmr.example.WebLog03", NT);
      for (String arg : args)
         System.out.printf (" %s", arg);
      System.out.println();
      System.out.printf ("%s%n", new Date());

      for (int i = 2; i < args.length; ++ i)
         mapperTask (args[i])
            .source (new TextFileSource (args[1]))
            .mapper (NT, MyMapper.class, args[0]);

      reducerTask()
         .customizer (MyCustomizer.class)
         .reducer (MyReducer.class);

      startJob();
      }

   // Print a usage message and exit.
   private static void usage()
      {
      System.err.println ("Usage: java pj2 [threads=<NT>] " +
         "edu.rit.pjmr.example.WebLog03 <file> <node> [<node> ...]");
      throw new IllegalArgumentException();
      }

   // Mapper class.
   private static class MyMapper
      extends Mapper<Long,String,IPAddress,LongVbl>
      {
      private static final LongVbl ONE = new LongVbl.Sum (1L);
      private String account;

      public void start
         (String[] args,
          Combiner<IPAddress,LongVbl> combiner)
         {
         account = args[0];
         }

      public void map
         (Long inKey,     // Line number
          String inValue, // Line from file
          Combiner<IPAddress,LongVbl> combiner)
         {
         int n = inValue.length();
         int i = inValue.indexOf ("/~");
         if (i != -1)
            {
            int j = i + 2;
            char c;
            while (j < n &&
               (c = inValue.charAt (j)) != '/' &&
               ! Character.isWhitespace (c))
                  ++ j;
            if (inValue.substring (i + 2, j) .equals (account))
               {
               Scanner scanner = new Scanner (inValue);
               if (scanner.hasNext())
                  {
                  try
                     {
                     IPAddress ipaddr = new IPAddress (scanner.next());
                     combiner.add (ipaddr, ONE);
                     }
                  catch (Throwable exc)
                     {
                     }
                  }
               }
            }
         }
      }

   // Reducer task customizer class.
   private static class MyCustomizer
      extends Customizer<IPAddress,LongVbl>
      {
      public boolean comesBefore
         (IPAddress key_1, LongVbl value_1, // IP address -> requests
          IPAddress key_2, LongVbl value_2)
         {
         if (value_1.item > value_2.item)
            return true;
         else if (value_1.item < value_2.item)
            return false;
         else
            return key_1.compareTo (key_2) < 0;
         }
      }

   // Reducer class.
   private static class MyReducer
      extends Reducer<IPAddress,LongVbl>
      {
      long total;
      public void start (String[] args)
         {
         total = 0;
         }
      public void reduce
         (IPAddress key, // IP address
          LongVbl value) // Number of requests
         {
         System.out.printf ("%s\t%s%n", value, key);
         total += value.item;
         }
      public void finish()
         {
         System.out.printf ("%d\tTotal%n", total);
         }
      }

   }
```

Map-reduce job configuration

Configure multiple mapper tasks

Configure reducer task

Fire up the job

Mapper class

Process one line of an input file

Extract relevant information

Record and reduce output (key, value)

Customizer class

Specify order of reducer inputs

Reducer class

Reduce mapper outputs for a given key

Print results

```
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
192.248.8.65 - - [10/Oct/2013:23:59:56 -0400] "GET /~ark
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
74.74.166.42 - - [11/Oct/2013:00:00:00 -0400] "GET /~hpb
74.74.166.42 - - [11/Oct/2013:00:00:00 -0400] "GET /~hpb
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
74.74.166.42 - - [11/Oct/2013:00:00:00 -0400] "GET /~hpb
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
67.167.235.253 - - [11/Oct/2013:00:00:00 -0400] "GET /~a
74.74.166.42 - - [11/Oct/2013:00:00:00 -0400] "GET /~hpb
. . .
```

Web log files

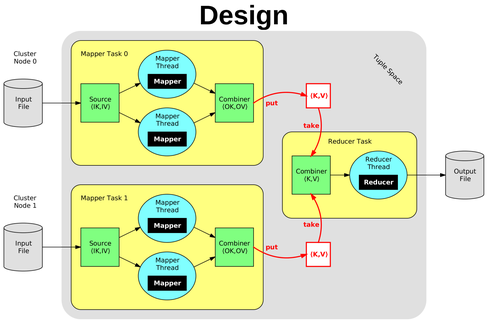
Number of requests from each unique IP address to user ark's web pages, in descending order, plus total requests

```
993     66.249.75.56
255     100.43.83.140
86      67.240.174.196
85      129.21.90.221
83      157.56.229.138
81      65.55.52.117
76      129.21.141.79
59      129.21.154.211
56      129.21.36.199
53      151.48.1.83
51      129.21.74.122
51      129.21.107.29
51      129.21.121.182
49      129.21.133.239
47      129.21.75.146
43      50.30.232.235
40      129.21.62.112
36      71.186.248.249
35      24.93.20.221
35      129.21.134.32
. . .
4589    Total
```

Features:

- **User writes—**
  - **Reduction variable class**
  - **Mapper class**
  - **Reducer class**
  - **Customizer class (optional)**
  - **Map-reduce job main program**
- **The framework does the rest**

Design



Performance:

- **10 nodes, 40 cores, 2.6 GHz**
- **1.4 GB total web log files (0.14 GB file on each node)**
- **10 mapper tasks**
- **2 threads per mapper task**
- **22.9 seconds running time**

THE PARALLEL JAVA 2 LIBRARY

GPU Parallel Programming in Java + CUDA

```
package edu.rit.gpu.example;

import edu.rit.gpu.Kernel;
import edu.rit.gpu.Gpu;
import edu.rit.gpu.GpuLongVbl;
import edu.rit.gpu.Module;
import edu.rit.pj2.Task;

public class PiGpu
   extends Task
   {
   // Kernel function interface.
   private static interface PiKernel
      extends Kernel
      {
      public void computeRandomPoints
         (long seed,
          long N);
      }

   // Task main program.
   public void main
      (String[] args)
      throws Exception
      {
      // Validate command line arguments.
      if (args.length != 2) usage();
      long seed = Long.parseLong (args[0]);
      long N = Long.parseLong (args[1]);

      // Initialize GPU.
      Gpu gpu = Gpu.gpu();
      gpu.ensureComputeCapability (2, 0);

      // Set up GPU counter variable.
      Module module = gpu.getModule
         ("edu/rit/gpu/example/PiGpu.cubin");
      GpuLongVbl count = module.getLongVbl ("devCount");

      // Generate n random points in the unit square, count how
      // many are in the unit circle.
      count.item = 0;
      count.hostToDev();
      PiKernel kernel = module.getKernel (PiKernel.class);
      kernel.setBlockDim (1024);
      kernel.setGridDim (gpu.getMultiprocessorCount());
      kernel.computeRandomPoints (seed, N);

      // Print results.
      count.devToHost();
      System.out.printf ("pi = 4*%d/%d = %.9f%n", count.item, N,
         4.0*count.item/N);
      }

   // Print a usage message and exit.
   private static void usage()
      {
      System.err.println ("Usage: java pj2 " +
         "edu.rit.gpu.example.PiGpu <seed> <N>");
      System.err.println ("<seed> = Random seed");
      System.err.println ("<N> = Number of random points");
      throw new IllegalArgumentException();
      }
   }
```

Java main program

Java interface to GPU kernel function

Get access to GPU

Get access to result variable on the GPU

Initialize GPU variable

Execute kernel on GPU

Read GPU variable and print result

```
#include "Random.cu"

// Number of threads per block.
#define NT 1024

// Overall counter variable in global memory.
__device__ unsigned long long int devCount;

// Per-thread counter variables in shared memory.
__shared__ unsigned long long int shrCount [NT];

/**
 * Device kernel to compute random points.
 *
 * Called with a one-dimensional grid of one-dimensional blocks,
 * NB blocks, NT threads per block. NT must be a power of 2.
 *
 * @param  seed  Pseudorandom number generator seed.
 * @param  N     Number of points.
 */
extern "C" __global__ void computeRandomPoints
   (unsigned long long int seed,
    unsigned long long int N)
   {
   int thr, size, rank;
   unsigned long long int count;
   prng_t prng;

   // Determine number of threads and this thread's rank.
   thr = threadIdx.x;
   size = gridDim.x*NT;
   rank = blockIdx.x*NT + thr;

   // Initialize per-thread prng and count.
   prngSetSeed (&prng, seed + rank);
   count = 0;

   // Compute random points.
   for (unsigned long long int i = rank; i < N; i += size)
      {
      double x = prngNextDouble (&prng);
      double y = prngNextDouble (&prng);
      if (x*x + y*y <= 1.0) ++ count;
      }

   // Shared memory parallel reduction within thread block.
   shrCount[thr] = count;
   __syncthreads();
   for (int i = NT/2; i > 0; i >>= 1)
      {
      if (thr < i)
         shrCount[thr] += shrCount[thr+i];
      __syncthreads();
      }

   // Atomic reduction into overall counter.
   if (thr == 0)
      atomicAdd (&devCount, shrCount[0]);
   }
```

CUDA kernel

GPU result variable

Partition computation among kernel threads

Perform computation in parallel

Reduce per-thread results

Reduce per-block results into GPU result variable

Monte Carlo estimation of π on the GPU

Running time comparison (msec)

- **CPU: Intel Pentium E5400, 2.7 GHz**
- **GPU: Nvidia Tesla C2070, 448 cores, 1.1 GHz**

N	*CPU*	*GPU*	*Ratio*
1e6	74	64	1.2
1e7	561	67	8.4
1e8	5450	83	65.7
1e9	54310	253	214.7
1e10	543521	1963	276.9

THE PARALLEL JAVA 2 LIBRARY

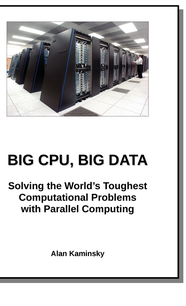
Resources for Teaching Parallel Programming

The Parallel Java 2 Library

- **Download—http://www.cs.rit.edu/~ark/pj2.shtml**
- **Free software, GNU GPL licensed**
- **Supports multicore and cluster parallel programming in 100% Java (JDK 1.7)**
- **Supports GPU parallel programming in Java with GPU kernels in C/CUDA**
- **Includes a lightweight map-reduce framework for big data programming in 100% Java**
- **Includes its own easy-to-install cluster middleware**

***BIG CPU, BIG DATA* textbook**

- **Download—http://www.cs.rit.edu/~ark/bcbd/**
- **Free classroom-tested textbook, Creative Commons licensed**
- **Covers multicore, cluster, GPU, and map-reduce parallel programming with the Parallel Java 2 Library**
- **Includes numerous complete parallel program examples**
- **Emphasizes performance measurement, performance modeling, strong and weak scaling**



Alan Kaminsky, Professor
Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
http://www.cs.rit.edu/~ark/
ark@cs.rit.edu