The Parallel Java 2 Library

Parallel Programming in 100% Java

Alan Kaminsky
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
ark@cs.rit.edu

***Abstract*—The Parallel Java 2 Library (PJ2) is an API and middleware for parallel programming in 100% Java on multicore parallel computers, cluster parallel computers, and hybrid multicore cluster parallel computers. In addition, PJ2 supports programming GPU accelerated parallel computers, with main programs written in Java and GPU kernels written using Nvidia's CUDA. PJ2 also includes a lightweight map-reduce framework for big data parallel programming. PJ2 has its own job scheduling middleware that coordinates usage of a node's or cluster's computational resources by multiple users. Encompassing four major categories of parallel programming—multicore, cluster, GPU, and map-reduce—in a single unified Java-based API, PJ2 is suitable for teaching and learning parallel programming and for real world parallel program development.**

Keywords—parallel programming libraries; parallel programming middleware; Java parallel programming; parallel programming education

I. Introduction

When teaching a parallel programming course, the instructor must choose a programming language (or languages) and a parallel programming library (or libraries). For the students to have the best educational experience, the language and library must meet two requirements. First, the language must be one with which the students are already familiar. Second, the library should support the chosen language and should support all the paradigms—multicore, cluster, GPU, big data, and so on—to be covered in the course. If the students need to learn a new language, or if they need to learn how to use multiple libraries, then too much time will be spent getting up to speed on the language or libraries, leaving inadequate time to study actual parallel programming.

Existing parallel programming libraries fall short on both requirements. Each of the popular libraries supports only one paradigm: OpenMP for multicore parallel programming, MPI for message passing parallel programming on clusters, OpenCL or Nvidia's CUDA for GPU parallel programming, Apache's Hadoop for map-reduce big data programming. None of these libraries supports all the aforementioned paradigms with a single, integrated API. As for languages, many computer science students are familiar with Java. The Advanced Placement computer science courses are taught in Java [1]. Java is taught as the introductory language in 22 of the top 39 U.S. computer science departments as of 2014 [2], surpassed only by Python. However, none of the popular parallel programming libraries, except Hadoop, support Java (or Python).

To address these deficiencies, I developed the Parallel Java Library (PJ) [3,4,5] and its successor, the Parallel Java 2 Library (PJ2) [6,7]. I have taught parallel computing courses in Java using PJ since 2005 and using PJ2 since 2013. I and others have also used PJ and PJ2 for real-world parallel program development in Java [6]. This poster gives an overview of PJ2.

II. PJ2 Example

Below is an example of a PJ2 multicore parallel program that uses a Monte Carlo technique to calculate an estimate for pi. The program generates a large number N of random (x,y) points in the unit square, counts the number of points C that fall within a distance of 1 from the origin, and estimates pi as $4C/N$. The points are calculated in a parallel loop, introduced by `parallelFor`. Under the hood, PJ2 automatically creates a team of threads, one thread for each core of the machine, and partitions the loop iterations among the threads. Each thread has its own local counter and pseudorandom number generator. The thread-local counters are automatically sum-reduced together into the global counter. For a detailed explanation of how the program works, see [6].

```
public class PiSmp extends Task
   {
   // Command line arguments.
   long seed;
   long N;

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

   // Main program.
   public void main
      (String[] args)
      throws Exception
      {
      // Validate command line arguments.
      if (args.length != 2) usage();
      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);
      parallelFor (0, N - 1) .exec (new LongLoop()
         {
         Random prng;
         LongVbl thrCount;
         public void start()
            {
            prng = new Random (seed + 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;
      }
   });

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

III. PJ2 Capabilities

PJ2 is a Java class library that supports shared memory parallel programming on multicore computers; message passing parallel programming on cluster computers; hybrid shared memory and message passing parallel programming on clusters of multicore nodes; GPU accelerated parallel programming; and big data map-reduce parallel programming.

For multicore parallel programming, PJ2 provides *parallel loops* and *parallel sections*, conceptually similar to OpenMP. A parallel loop's *schedule*—the manner in which loop iterations are partitioned among the threads—can be specified: fixed and leapfrog schedules for inherently balanced computations; dynamic, guided, and proportional schedules to achieve load balancing. *Reduction variables* are supported. PJ2 includes reduction variable classes for primitive types with predefined reduction operations, such as sum, minimum, and maximum. One can also define one's own reduction variable classes doing arbitrary reduction operations on arbitrary data types, including objects.

For cluster parallel programming, one writes a *job* consisting of multiple *tasks*. Each task is its own independent multicore parallel program running on a cluster node. Tasks can be created when the job commences execution; tasks can also be created on the fly as the job executes. Tasks communicate with each other via *tuple space* [8]; tasks put *tuples* (objects) containing arbitrary information into tuple space, and other tasks take tuples out of tuple space that match given *templates*. PJ2 provides *master-worker parallel loops*, in which the loop iterations are automatically partitioned among a number of worker tasks. PJ2 includes its own integrated cluster middleware; a *tracker* daemon maintains a job queue, keeps track of the status of the cluster's computational resources (nodes, cores, accelerators), and schedules each task in each job to run on an appropriate node when resources become available.

For GPU accelerated parallel programming, PJ2 supports invoking GPU kernel functions from PJ2 programs. The PJ2 program is written in Java; the kernel function is written using Nvidia's CUDA. The kernel function manifests itself as a Java method that is called like any other method. PJ2 also provides Java classes for variables, arrays, and matrices of primitive types and `struct` types. Each instance of such a class *mirrors* a corresponding variable on the GPU and provides methods for transferring the data stored in the variable between the CPU and the GPU. PJ2's GPU programming capabilities are fully integrated with the rest of the library. One can easily write a multicore program consisting of multiple threads each interacting with its own GPU accelerator, or a cluster job where each task is itself a GPU parallel program running on a GPU accelerated node. The tracker is aware of CPU cores and GPU accelerators on each cluster node, and automatically schedules tasks to run on nodes possessing the necessary computational resources.

For big data parallel programming, PJ2 includes *Parallel Java Map Reduce (PJMR)*, a lightweight map-reduce framework built on top of PJ2's cluster programming capabilities. One writes Java classes for a source, a mapper, a combiner, and a reducer. The *source* reads raw data records from a file or files stored on disk. The *mapper* extracts relevant information from a record and adds the information to a combiner. The *combiner* is a mapping from keys to values; when a key-value pair is added, the new value is reduced into the value associated with the key using some reduction operation. The *reducer* processes a key-value pair from a combiner to generate the program's results. A map-reduce job consists of multiple mapping tasks as well as a reducing task. Each mapping task consists of a source, one or more mappers, and a combiner. The reducing task consists of a combiner and one or more reducers. The mapping tasks run in parallel on multiple cluster nodes, each processing the portion of the data records stored on that node. When each mapping task finishes, its combiner is reduced into the reducing task's combiner. When all the mapping tasks have finished, the reducing task generates the program's results from the combiner's final contents.

IV. PJ2 Educational Resources

PJ2 is free, GNU GPL licensed software available from the author's web site [7]. The download includes all Java source files, Java class files, and Javadoc documentation. The library also includes numerous parallel program examples illustrating various features of the library. PJ2 runs on any system with Java version 1.7 or higher.

A companion textbook, titled *Big CPU, Big Data: Solving the World's Toughest Computational Problems with Parallel Computing*, is being written. The textbook is free, Creative Commons licensed, and is available from the author's web site [6]. The textbook covers multicore, cluster, GPU, and map-reduce parallel programming using PJ2, as well as related topics like performance modeling and strong and weak scaling.

References

[1] "AP Computer Science A course overview." June 2014. http://media.collegeboard.com/digitalServices/pdf/ap/ap-course-overviews/ap-computer-science-a-course-overview.pdf

[2] P. Guo. "Python is now the most popular introductory teaching language at top U.S. universities." July 7, 2014. http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext

[3] A. Kaminsky. "Parallel Java: a unified API for shared memory and cluster parallel programming in 100% Java." IEEE International Parallel and Distributed Processing Symposium, 2007.

[4] A. Kaminsky. Building Parallel Programs: SMPs, Clusters, and Java. Boston, MA: Course Technology, 2010.

[5] A. Kaminsky. "Parallel Java Library." June 20, 2012. http://www.cs.rit.edu/~ark/pj.shtml

[6] A. Kaminsky. Big CPU, Big Data: Solving the World's Toughest Computational Problems with Parallel Computing. Creative Commons, 2014. http://www.cs.rit.edu/~ark/bcbd/

[7] A. Kaminsky. "Parallel Java 2 Library." September 19, 2014. http://www.cs.rit.edu/~ark/pj2.shtml

[8] D. Gelernter. "Generative communication in Linda." ACM Transactions on Programming Languages and Systems, 7(1):80-112, January 1985.