HPC Challenge 2014
PCJ Benchmarks
(Parallel Computing in Java)

Marek Nowicki, Lukasz Górska, Piotr Bala
bala@icm.edu.pl

N. Copernicus University, Torun, Poland
ICM - University of Warsaw, Warsaw, Poland
Parallel computing in Java – challenges

- Parallel programming is still difficult especially while traditional programming paradigms are used.
- There is need for new programming paradigms such as Partitioned Global Address Space (PGAS).
- HPC market has to open for new languages widely used for data analysis such as Java.
- Parallel programming in Java is either threads or fork/join and is limited to a single JVM.
- There has been a number of parallel extensions to Java however none of them become popular.
Java library developed at ICM

- pcj.icm.edu.pl

Programming paradigm:

- partitioned global address space (PGAS)
- all variables are local by default
- variables can be global (@Shared)
- one sided communication (put, get)

Features

- does not require modification of JVM
- does not require other libraries!
- works on almost all operating system that have JVM
- uses newest Java SE 7 (NIO, SDP, . . . )
Basic functionality of PCJ:
- tasks numbering
- synchronization of tasks
- getting values
- putting values

Advanced functionality:
- broadcasting values
- monitoring variables
- parallel I/O
- creating groups of nodes
- working with groups.
import org.pcj.*

public class PcjHelloWorld extends Storage implements StartPoint {

    @Override
    public void main() {
        System.out.println("Hello!");
    }

    public static void main(String[] args) {
        String[] nodes = new String[]{"localhost", "localhost"];
        PCJ.deploy(PcjHelloWorld.class, PcjHelloWorld.class, nodes);
    }
}
@Shared double a;

double c;

if (PCJ.myId()==0) c = (double) PCJ.get(3, "a");

FutureObject aL[] = new FutureObject[PCJ.threadCount()];
if (PCJ.myId()==0) aL[p] = PCJ.getFutureObject(p, "a");
c = (double) aL[p].get();

if (PCJ.myId()==0) PCJ.put(3, "a", 5.0);

public static void PCJ.barrier();
public static int PCJ.threadCount()
@Shared double a

FutureObject aL[] = new FutureObject[PCJ.threadCount()];
double a0 = 0.0;

if (PCJ.myId() == 0) {
    for (int p = 0; p < PCJ.threadCount(); p++) {
        aL[p] = PCJ.getFutureObject(p, "a");
    }

    for (int p = 0; p < PCJ.threadCount(); p++) {
        a0 = a0 + (double) aL[p].get();
    }
}

## HPC Challenge PCJ benchmarks

### HPC Benchmarks
- **STREAM**: 180 LOC
- **Random Access**: 146 LOC
- **GlobalFFT 1D**: 498 LOC

### Our benchmarks
- **MapReduce**: 126 LOC
- **RayTracing**: 1627 LOC (incl. 100 comment lines)

52 PCJ calls, incl. 35 log statements

---

```java
long sum = 0;
for ( User user : users ) { um += user.getAge (); }
```

```java
double average = (double) sum / users.size ();
```
MapReduce - Java

- **Java**
  
  ```java
  long sum = 0;
  for ( User user : users ) {
      um += user.getAge ();
  }
  double average = (double) sum / users.size ();
  ```

- **Java 8 parallel streams**
  
  ```java
  long sum = users.parallelStream ()
      .map (u -> ( long ) u. getAge ()
      .reduce ( Long :: sum )
      .get ();
  double average = (double) sum / users.size ();
  ```
@Shared long sum ;
@Shared int usersCount ;
...

myUsers = loadUsers( PCJ.myId ());
long s = 0;
for ( User u : myUsers ) {
    s += u. getAge ();
}
PCJ.putLocal ("sum", s);
PCJ.barrier ();
s = pcj_reduce ("sum");
double average = (double) s / count ;
PCJ performance – STREAM

![Graph](image_url)
PCJ performance – Random Access

![Graph showing the performance of halo2, hydra, and boreasz in random access across different numbers of cores.](image-url)
PCJ performance – Global FFT

![Graph showing performance of Global FFT with different configurations: halo2, hydra, boreasz. The x-axis represents the number of cores, ranging from 1 to 1024, and the y-axis represents GlobalFFT in Gflops/s. The graph compares the performance across different core counts.](image)
PCJ performance – Raytracer

The image shows a graph plotting the efficiency of different raytracer implementations (halo2, hydra, boreasz) against the number of cores. The x-axis represents the number of cores, ranging from 1 to 1024, and the y-axis represents efficiency in pixels per second (px/s), ranging from 10000 to 1e+006 (1 million). The graph illustrates how the efficiency scales with the number of cores for each implementation.
PCJ performance – MapReduce

MapReduce [s] vs Number of cores

- halo2
- hydra
- boreasz

UIKit logo
PCJ for HPC and BigData

- For single node PCJ performance is competitive compare to Java 8 parallel streams
- PCJ performance is competitive compare to standard solutions based on MPI
- PCJ runs on multiple nodes (multiple JVM)
- PCJ has very good scalability and has been run on 10k cores
- PCJ can be used to parallelize data analysis codes written in Java
Heterogenous parallel and distributed computing with Java

- **Partners**
  - ICM University of Warsaw (Warsaw, Poland)
  - IBM Research Lab (Zurich, Switzerland)
  - Queen's University of Belfast (Belfast, UK)
  - Bilkent Üniversitesi (Ankara, Turkey)

- **Focus**
  - ease of use and programmability of Java for distributed heterogeneous computing
  - heterogeneous systems including GPU and mobile devices
  - dependability and resilience by adding fault tolerance mechanisms
  - key applications including data-intensive Big Data applications

- **1st October 2014 – 31st September 2017**

- [hpdcj.icm.edu.pl](http://hpdcj.icm.edu.pl)
pcj.icm.edu.pl

Piotr Bała (ICM University of Warsaw)
Marek Nowicki (WMiI UMK)
Łukasz Górski (WMiI UMK)

bala@icm.edu.pl