Biblioteka PCJ do naukowych obliczeń równoległych

Marek Nowicki

Zakład Obliczeń Równoległych i Rozproszonych
Wydział Matematyki i Informatyki
Uniwersytet Mikołaja Kopernika
Chopina 12/18, 87-100 Toruń

faramir@mat.umk.pl

29 kwietnia 2015

12. spotkanie Toruń JUG

Plan prezentacji

MPI

MPI – Message-Passing Interface

- Historia: MPI-1.0 (1992), MPI-2.0 (1997), MPI-3.0 (2012)
- C, C++, Fortran, Java (wrappers, JNI), Python, ...
- zwykle dwa procesy – nadawca i odbiorca
- wiele parametrów

```
MPI_Comm_rank(MPI_Comm comm, int *rank)
MPI_Comm_size(MPI_Comm comm, int *nrank)
MPI_Barrier(MPI_Comm comm)
MPI_Send(void *buf, int count, MPI_Datatype datatype,
         int dest, int tag, MPI_Comm comm)
MPI_Recv(void *buf, int count, MPI_Datatype datatype,
         int source, int tag, MPI_Comm comm,
         MPI_Status *status)
```

OpenMP

OpenMP – Open Multi-Processing

- Historia: 1.0 (1997), 2.5 (2005), 4.0 (2013)
- rozszerzenia do: C, C++, Fortran
- wykorzystywanie dyrektyw kompilatora

  `#pragma omp <directive> [clause]`

```
int omp_get_num_threads();
int omp_get_thread_num();

#pragma omp parallel for private(i) \
                         shared(a, b, c) \
                         reduction(+:sum)

#pragma omp critical
#pragma omp single nowait
#pragma omp master
#pragma omp barrier
```

PGAS

PGAS – Partitioned Global Address Space

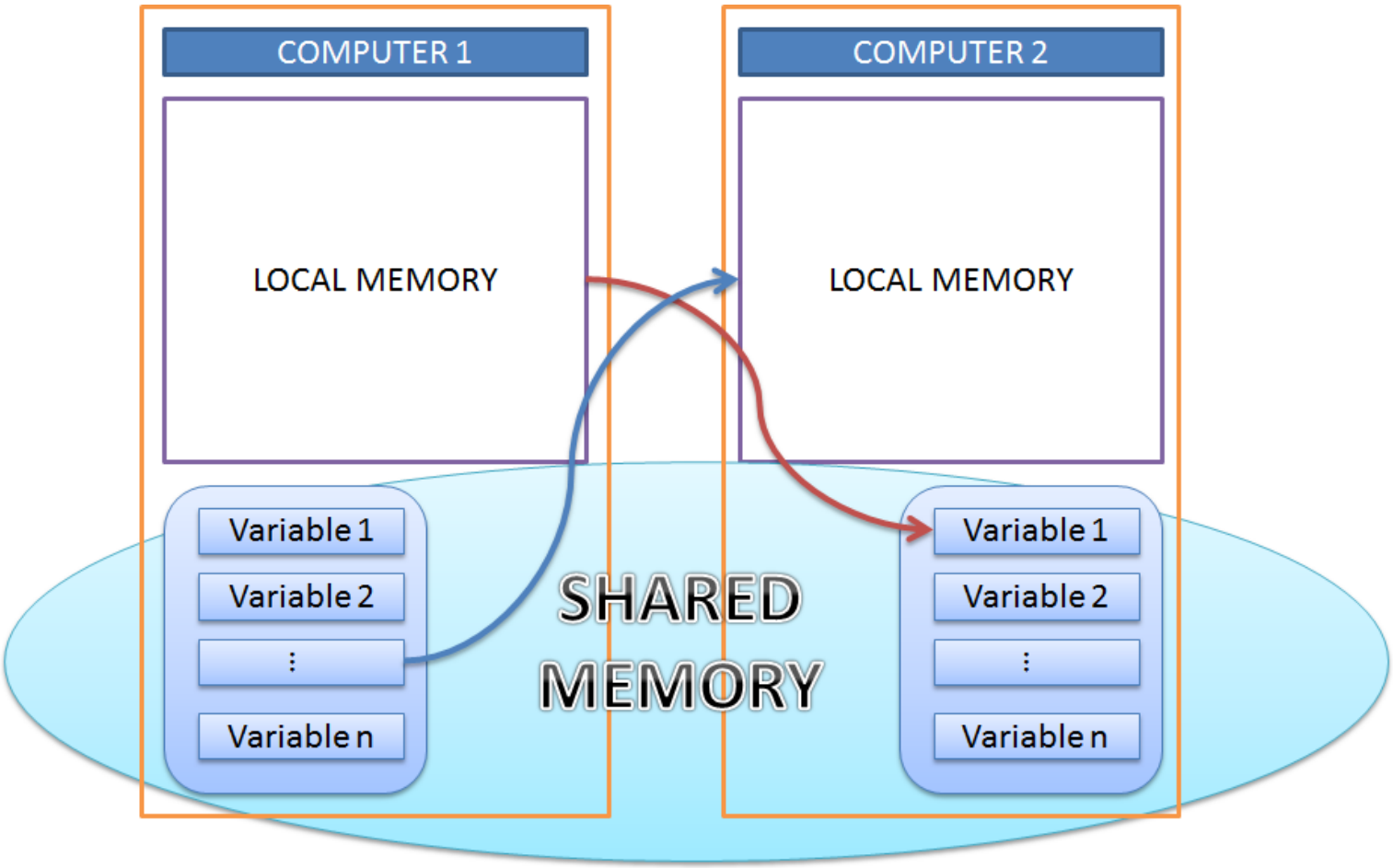
(Podzielona globalna przestrzeń adresowa)

Założenia:

- rozróżnialne dane lokalne i zdalne
- globalna przestrzeń adresowa – dostępna z wszystkich węzłów
- ukrywanie komunikacji przed użytkownikiem
- komunikacja jednostronna

`http://www.pgas.org`

PGAS


COMPUTER 1
COMPUTER 2
LOCAL MEMORY
LOCAL MEMORY
Variable 1
Variable 2
⋮
Variable n
SHARED MEMORY
Variable 1
Variable 2
⋮
Variable n

Java

Programowanie równoległe na jednym węźle

- `volatile`, `synchronized`
- `java.lang.Thread`, `java.lang.Runnable`
- `java.util.concurrent.*` (concurrent *Collections*, *Executors*, `ThreadPool*`, `ForkJoin*`, ...)
- `java.util.concurrent.atomic.*` (`AtomicCounter`, `AtomicInteger`, `AtomicReference`, ...)
- `java.util.concurrent.locks.*`
- `Collection.parallelStream()`
- ...

Java

Programowanie równoległe i rozproszone na wielu węzłach

- dJVM, JESSICA2, Terracotta
- RMI
- ProActive, Akka, ...
- CORBA, SOAP, ...
- MPJ

Java
PCJ – Parallel Computations in Java

Paradygmat programowania:

- PGAS

Podstawowe funkcje:

- synchronizowanie wątków
- pobieranie wartości zmiennych
- wysyłanie wartości

Zaawansowane funkcje:

- rozgłaszanie wartości
- monitorowanie zmiennych
- tworzenie grup wątków
- ...

Java
PCJ – Parallel Computations in Java

- Nie wymaga modyfikacji JVM
- Działa na systemach operacyjnych zawierających JVM
  (np. IBM Java 1.7 na klastrach o architekturze Power7, systemy AIX)
- Wykorzystuje funkcje wprowadzone w standardzie Java SE 7
  (NIO, SDP, . . .)
- Działa z wykorzystaniem Java SE 8
  (prace nad wykorzystaniem wyrażeń lambda)
- Nie wymaga dodatkowych bibliotek
  (brak problemu z zależnościami)

Dostępna na stronie: `http://pcj.icm.edu.pl`

Java
PCJ – Prosta aplikacja

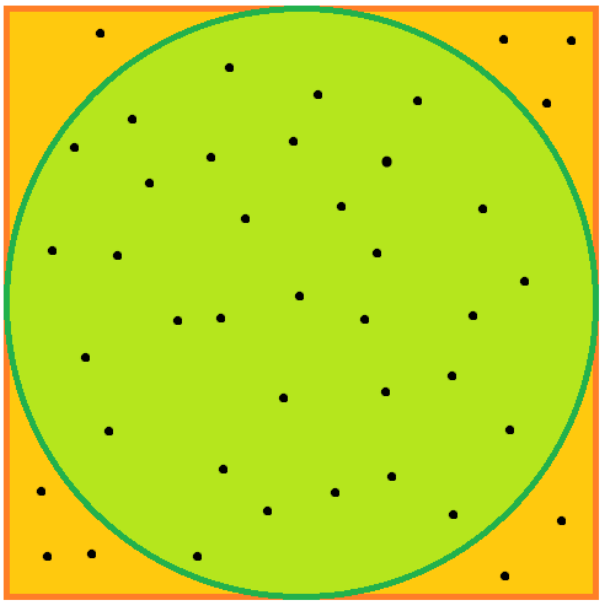
```
1  import org.pcj.*;
2  public class MyApp extends Storage implements StartPoint {
3      @Shared("array")
4      double[] tablica;
5
6      @Override
7      public void main() {
8          PCJ.log("Hello from " + PCJ.myId()
9                  + " of " + PCJ.threadCount());
10     }
11     public static void main(String[] args) {
12         String[] nodes = new String[]{"host0", "host0",
13                 "host1", "host1", "host2", "host2"
14         };
15         PCJ.deploy(MyApp.class, // StartPoint
16                    MyApp.class, // Storage
17                    nodes);
18     }
19 }
```

Przybliżanie wartości liczby π

Metoda Monte Carlo

Rzucamy punkty na kwadrat i zliczamy liczbę punktów w kole wpisanym w kwadrat.

$$\pi \approx \frac{4 \times \text{inCirclePoints}}{\text{totalPoints}}$$



Przybliżanie wartości liczby π (cont.)

Metoda Monte Carlo

```
1     Random random = new Random();
2     long nAll = 1_280_000_000;
3     long n = nAll / PCJ.threadCount();
4
5     long myCircleCount = 0;
6
7     for (long i = 0; i < n; ++i) {
8         double x = 2.0 * random.nextDouble() - 1.0;
9         double y = 2.0 * random.nextDouble() - 1.0;
10        if ((x * x + y * y) <= 1.0) {
11            myCircleCount++;
12        }
13    }
14    PCJ.putLocal("count", myCircleCount);
15    PCJ.barrier();
16
```

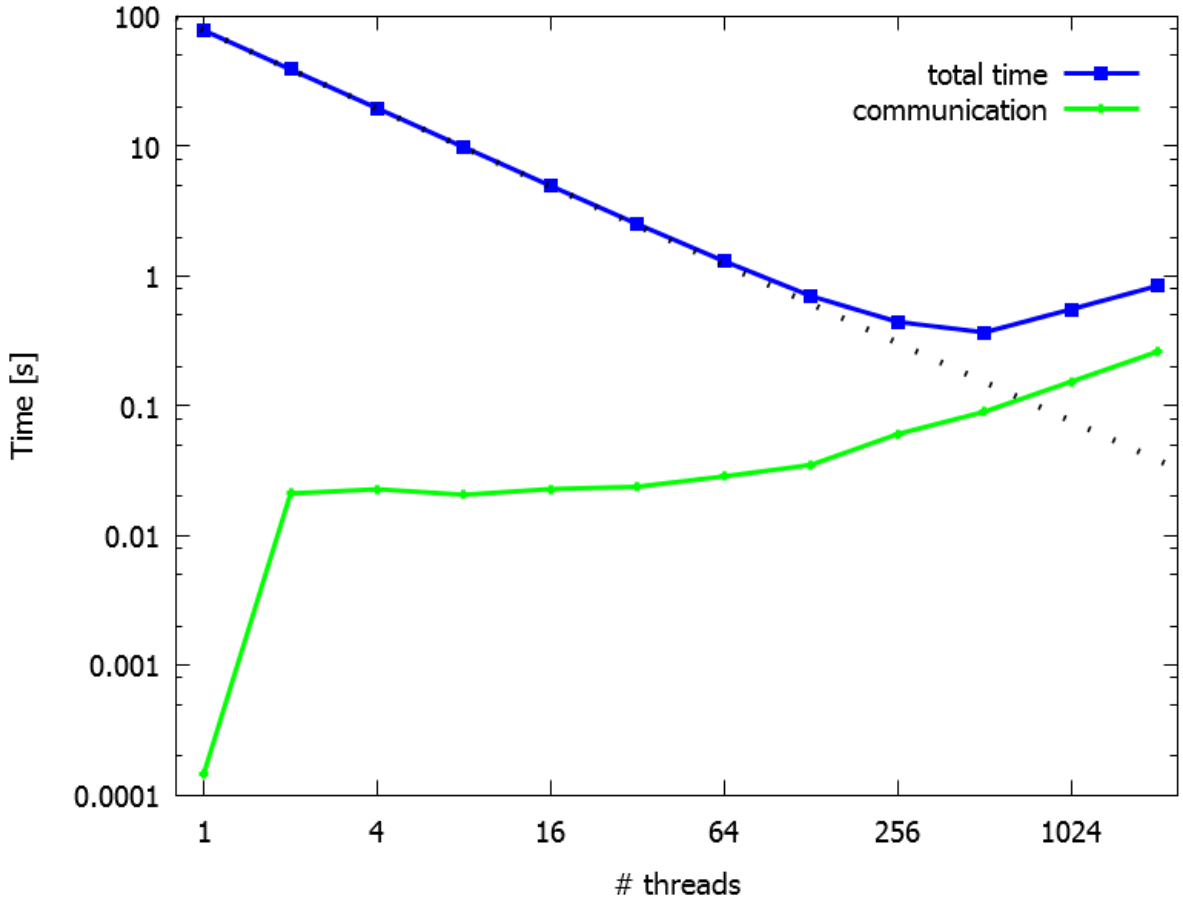
Przybliżanie wartości liczby π (cont.)

Metoda Monte Carlo

```
17      /* reduce -- get calculated data */
18      if (PCJ.myId() == 0) {
19          FutureObject cL[] =
20                  new FutureObject[PCJ.threadCount()];
21          for (int p = 0; p < PCJ.threadCount(); p++) {
22              cL[p] = PCJ.getFutureObject(p, "count");
23          }
24          long globalCircleCount = 0;
25          for (FutureObject fo : cL) {
26              globalCircleCount = globalCircleCount
27                      + (long) fo.get();
28          }
29          return 4.0 * (double) globalCircleCount
30                  / (double) (n * PCJ.threadCount());
31      }
32
33      return Double.NaN;
```

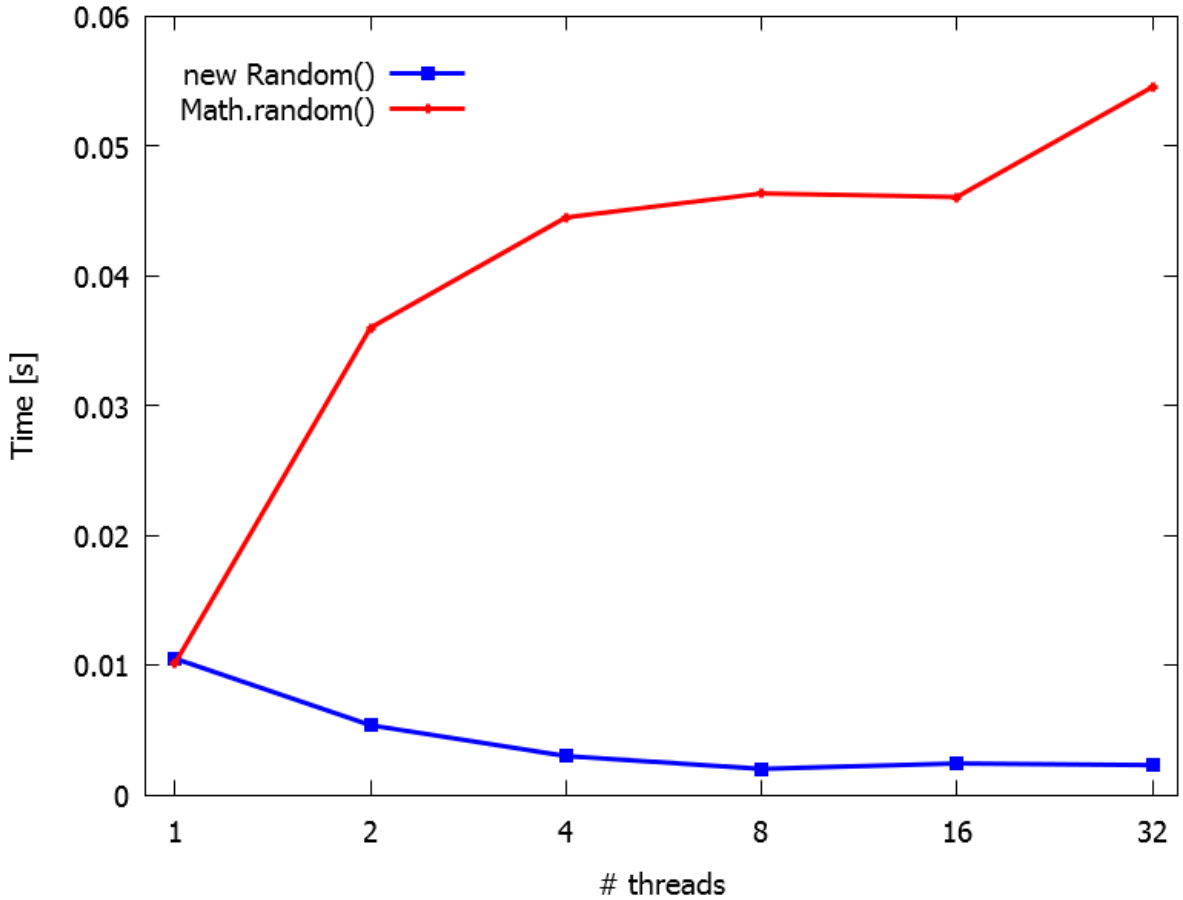
Przybliżanie wartości liczby π

Metoda Monte Carlo – halo2 (ICM): 1 280 000 000 punktów



Przybliżanie wartości liczby π

Metoda Monte Carlo – hydra (1 węzeł, PL-GRID, ICM): 200 000 puntków



Przybliżanie wartości liczby π

Metoda prostokątów

Przybliżanie wartości liczby π za pomocą metody prostokątów, czyli przybliżenie wartość poniższej całki:

$$\pi = \int_0^1 \frac{4}{1+x^2}\,\mathrm{d}x \approx \sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-\frac{1}{2}}{N}\right)^2}$$

Przybliżanie wartości liczby π (cont.)

Metoda prostokątów

```
1  private double f(double x) {
2      return (4.0 / (1.0 + x * x));
3  }
4
5  private double calculateIntegralPi() {
6      long n = 10_000;
7      double w = 1.0 / (double) n;
8
9      double mySum = 0.0;
10     for (int i = PCJ.myId() + 1; i <= n;
11               i += PCJ.threadCount()) {
12         mySum = mySum + f(((double) i - 0.5) * w);
13     }
14     mySum = mySum * w;
15
16     PCJ.putLocal("sum", mySum);
17     PCJ.barrier();
18
```

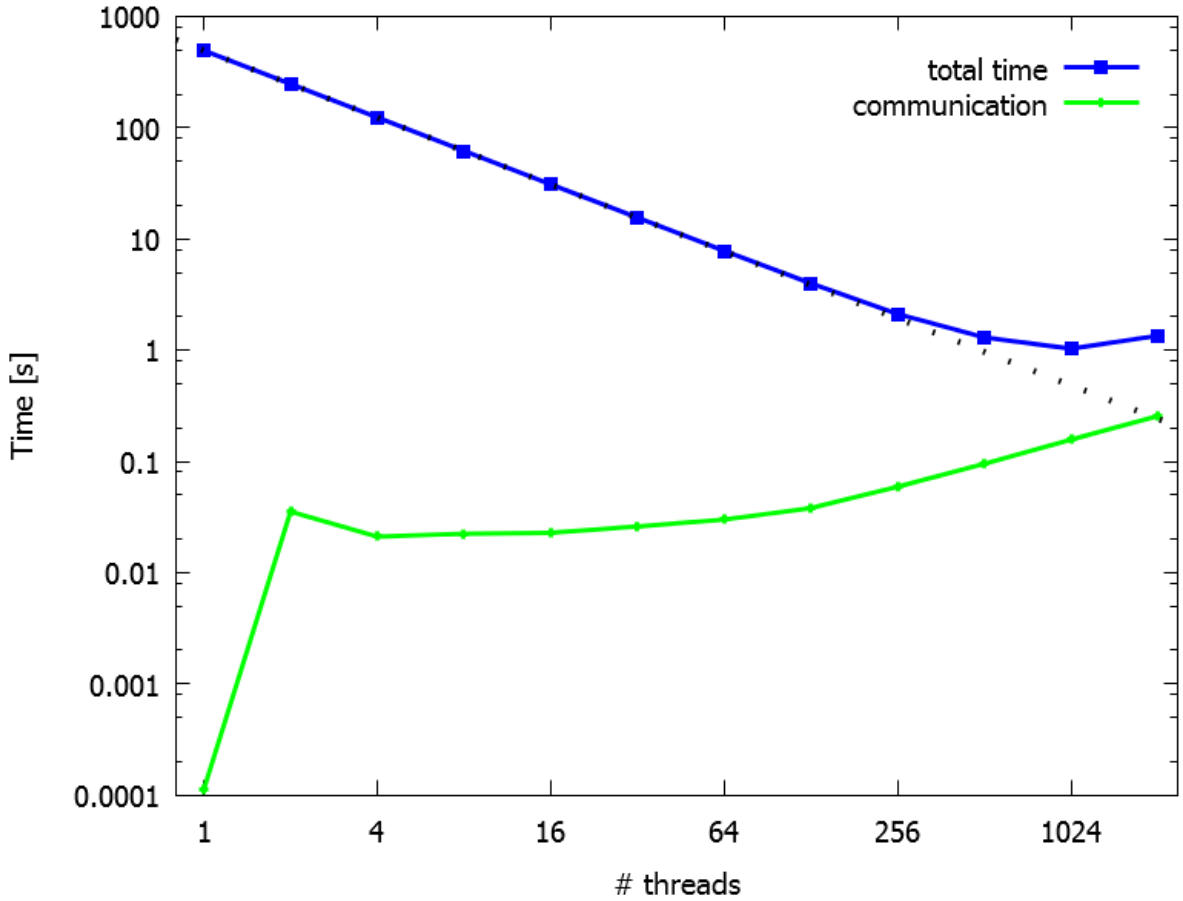
Przybliżanie wartości liczby π (cont.)

Metoda prostokątów

```
19      /* reduce -- get calculated data */
20      double gSum = 0.0;
21      if (PCJ.myId() == 0) {
22          FutureObject[] sL =
23                  new FutureObject[PCJ.threadCount()];
24          for (int i = 0; i < PCJ.threadCount(); ++i) {
25              sL[i] = PCJ.getFutureObject(i, "sum");
26          }
27          for (FutureObject fo : sL) {
28              gSum = gSum + (double) fo.get();
29          }
30      }
31
32      return gSum;
33  }
```

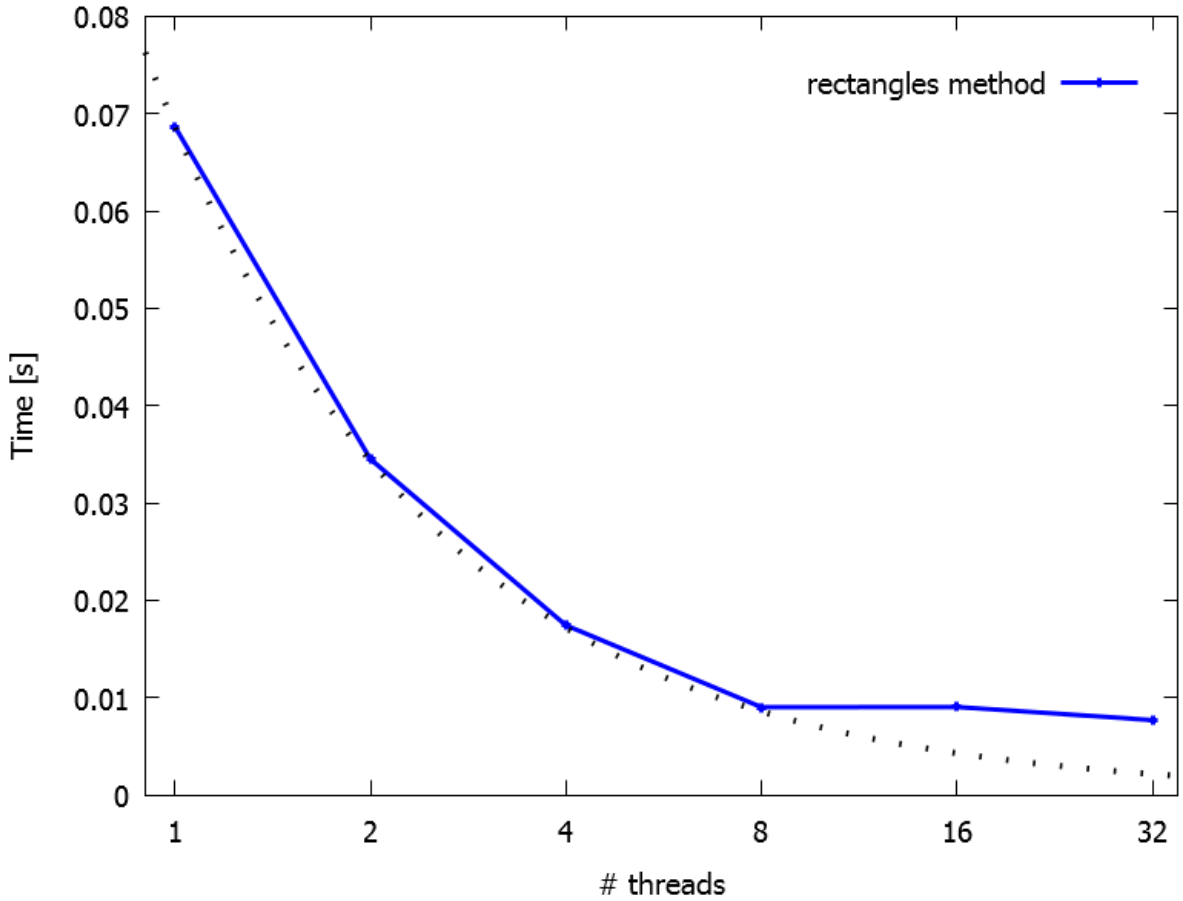
Przybliżanie wartości liczby π

Metoda prostokątów – halo2: 1 280 000 000 prostokątów



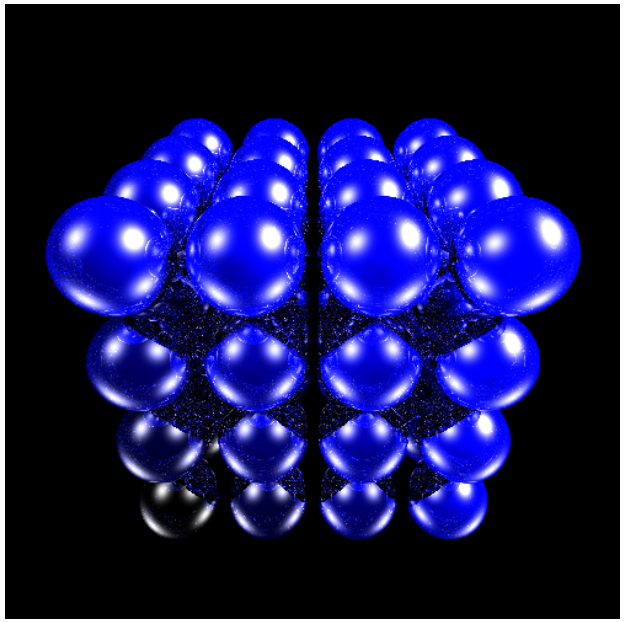
Przybliżanie wartości liczby π

Metoda Monte Carlo – hydra (1 węzeł, PL-GRID, ICM): 10 000 000 prostokątów



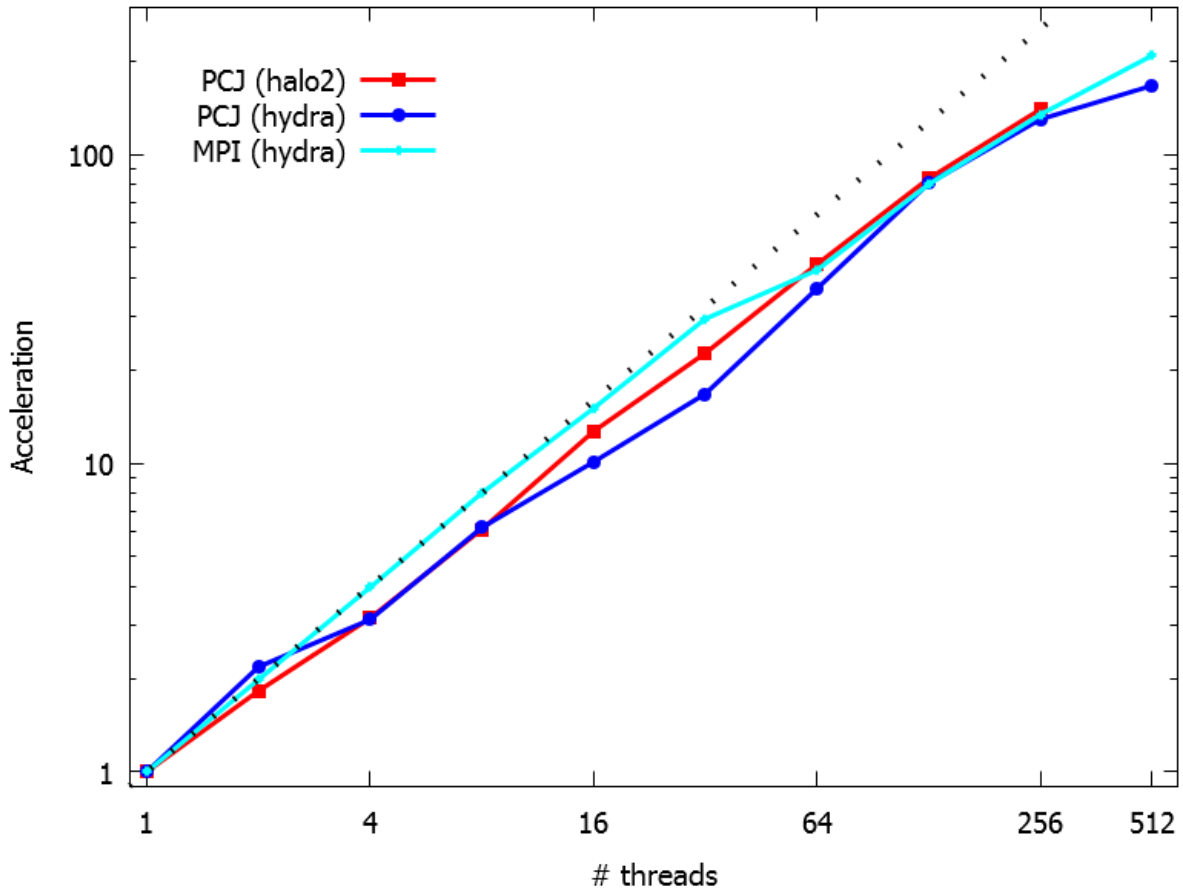
RayTracer

- część zbioru testów: Java Grande Forum Benchmark
- renderowanie sceny 3D używając mechanizmu *ray tracing*
- scena zawiera 64 kule (krata: $4 \times 4 \times 4$) i 5 źródeł światła
- generowany jest kwadratowy obraz rozmiaru NxN pikseli



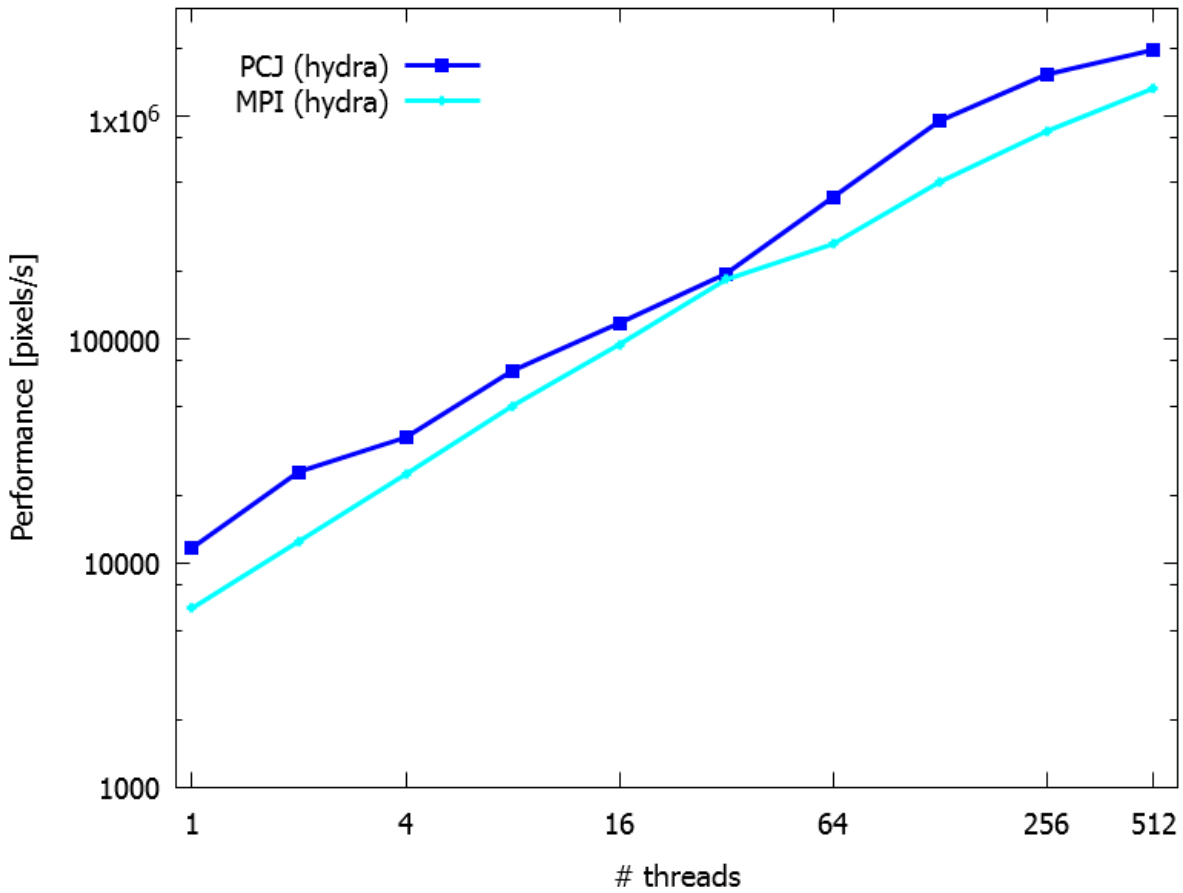
RayTracer

Przyspieszenie (obraz rozmiaru 2500 × 2500 pikseli)



RayTracer

Wydajność (obraz rozmiaru 2500 × 2500 pikseli)



MapReduce

for-each (Java 5 style):

```
1     long sum = 0;
2     for (User user : users) {
3         sum += user.getAge();
4     }
5     double average = (double) sum / users.size();
```

map.reduce.get (Java 8 style):

```
1     long sum = users.parallelStream()
2                     .map(u -> (long) u.getAge())
3                     .reduce(Long::sum)
4                     .get();
5     double average = (double) sum / users.size();
```

MapReduce

fork/join (Java 7 style):

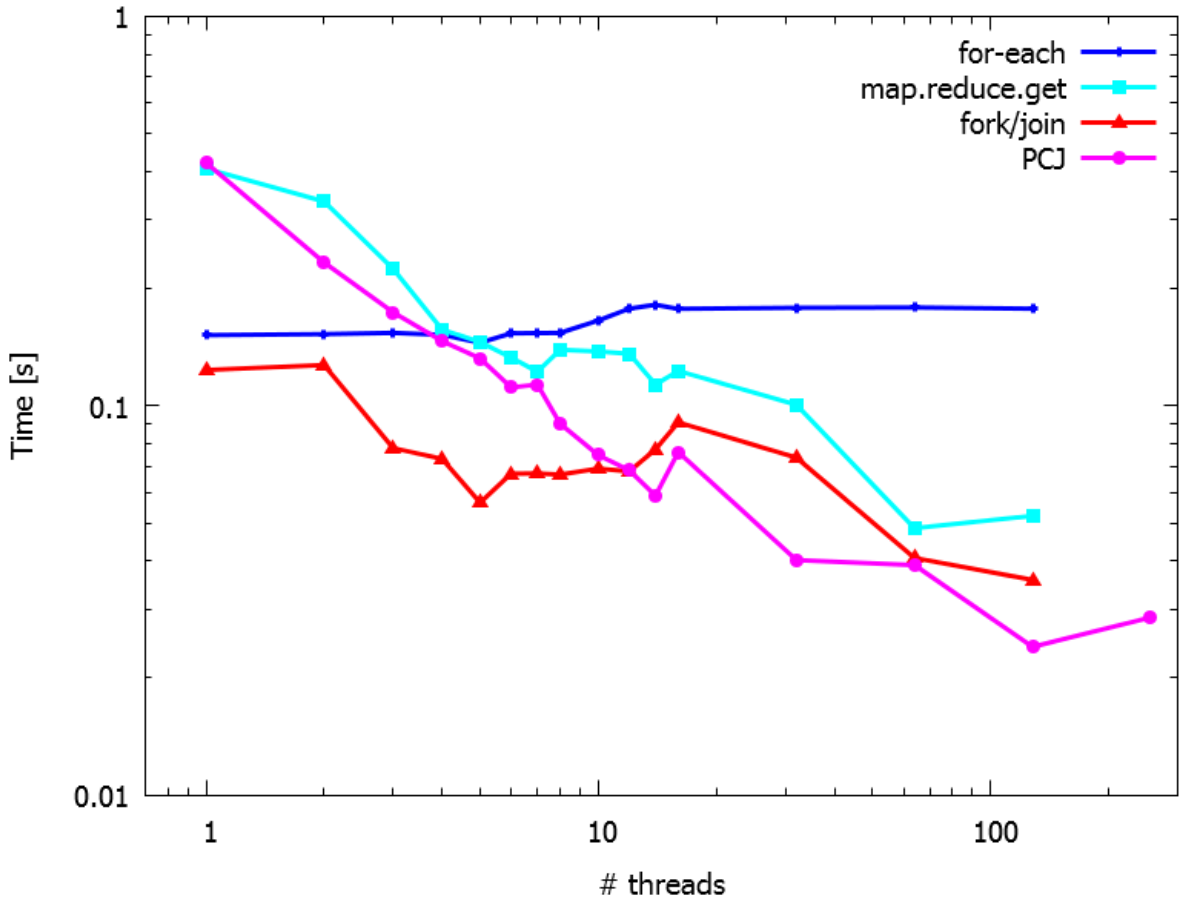
```
1  class ForkMapReduce extends RecursiveTask<Long> {
2      final private static int threshold = 16_384;
3      final private List<User> list;
4      protected ForkMapReduce(List<User> list) { this.list = list; }
5
6      @Override
7      protected Long compute() {
8          int length = list.size();
9          if (length < threshold) {
10             long sum = 0; for (User u : list) { sum += u.getAge(); } return sum;
11         }
12
13         int split = length / 2;
14         ForkMapReduce left = new ForkMapReduce(list.subList(0, split));
15         ForkMapReduce right = new ForkMapReduce(list.subList(split, length));
16
17         invokeAll(left, right);
18         return left.getRawResult() + right.getRawResult();
19     }
20 }
21 ...
22         ForkMapReduce fmr = new ForkMapReduce(users);
23         ForkJoinPool.commonPool().invoke(fmr);
24         long result = fmr.getRawResult();
25         double average = (double) result / users.size();
```

MapReduce
PCJ style

```
1       @Shared  long sum;
2       @Shared  int usersCount;
3   ...
4       myUsers = loadUsers(PCJ.myId());
5       // kazdy styl dla Java...
6       long s = 0;
7       for (User u : myUsers) {
8           s += u.getAge();
9       }
10
11      PCJ.putLocal("sum", s);
12      PCJ.putLocal("usersCount", myUsers.size());
13      PCJ.barrier();
14
15      s = pcj_reduce("sum");
16      int count = pcj_reduce("usersCount");
17      if (PCJ.myId() == 0) {
18          double average = (double) s / count;
19      }
```

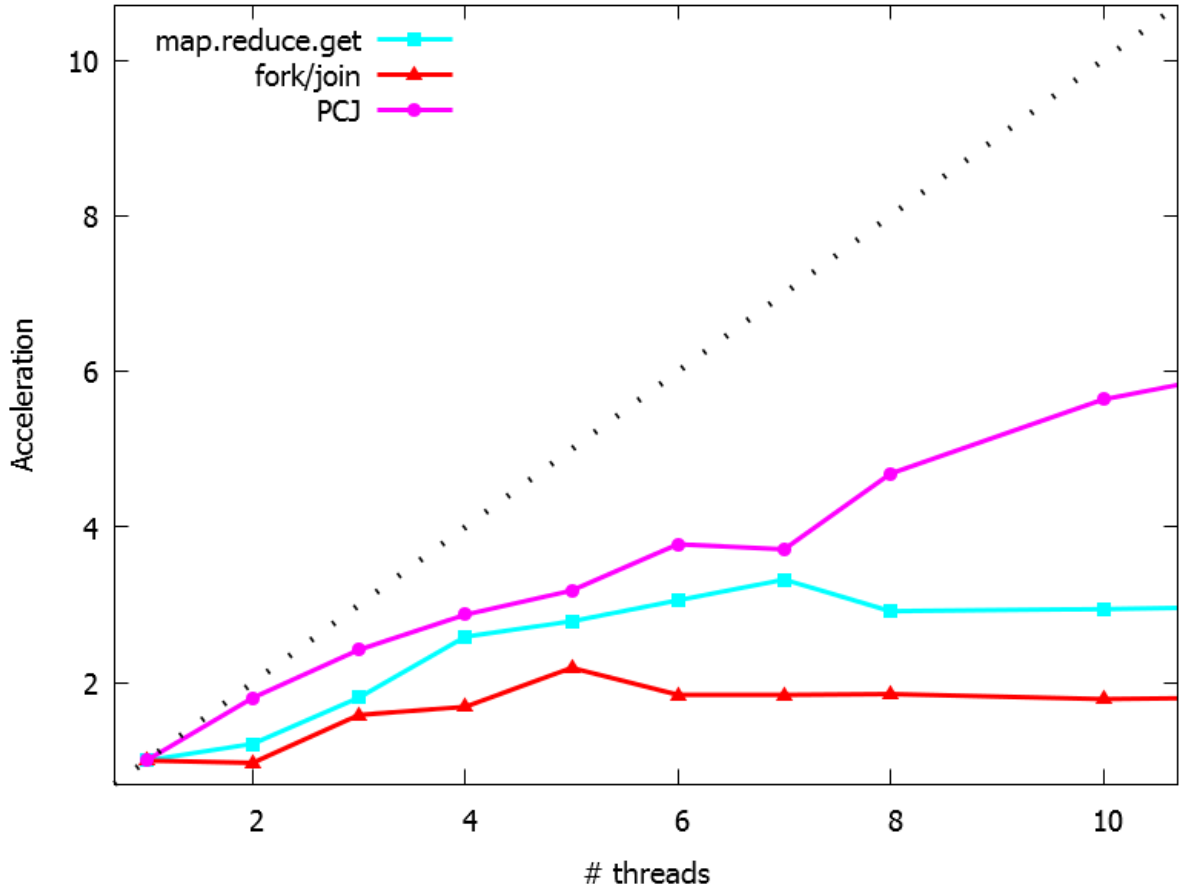
MapReduce

Time: 12 000 000 users



MapReduce

Acceleration: 12 000 000 users



Podsumowanie

- biblioteka ma potencjał, by być wykorzystywana w naukowych obliczeniach równoległych
- dobre rezultaty i bardzo dobra skalowalność uzyskiwane są szczególnie dla dużych danych, co otwiera możliwość wykorzystania biblioteki w analizie tzw. *Dużych Danych*.
- rozwiązania oparte o język Java mogą być tak samo szybkie, a nawet szybsze, od rozwiązań bazujących na językach niższego poziomu jak C czy C++

Podsumowanie

Biblioteka PCJ zdobyła nagrodę[1]:

HPC Challenge Class 2 Best Productivity Award

na międzynarodowej konferencji:

SC14
the International Conference for High Performance Computing, Networking, Storage and Analysis

za zaprezentowanie bardzo efektywnego sposobu programowania aplikacji równoległych.

[1] Nagroda przyznana na podstawie decyzji ekspertów oceniających implementację podzbioru testów HPC Challenge z wykorzystaniem PCJ

Plany na przyszłość
Aplikacje

- Przeportowanie aplikacji **SinusMed** na PCJ
- Zmierzenie się z pozostałymi testami na **HPC Challenge**
- Zbudowanie biblioteki opartej o PCJ do **operowania na grafach**
- Analiza danych genetycznych
- Zrównoleglanie **algorytmów genetycznych** w PCJ
- Analiza dużych danych – *BigData*

HPDCJ Project (CHIST-ERA)

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

Dziękuję za uwagę

http://pcj.icm.edu.pl

e-mail: faramir@mat.umk.pl