Performance evaluation of parallel computing and Big Data processing with Java and PCJ

dr Marek Nowicki[2], dr Łukasz Górski[1], prof. Piotr Bała[1]
bala@icm.edu.pl

[1] ICM University of Warsaw, Warsaw, Poland
www.icm.edu.pl
[2] Nicolaus Copernicus University, Toruń, Poland

Motivation

- Current computer architecture consists of many multicore processors
- Parallel processing is key functionality to program multinode multicore computers
- Current parallel programming models and tools (MPI, OpenMP, OpenACC) are not feasible enough
  - 70% of jobs running on HPC clusters is still single node job
- There is growing interest in new paradigms:
  - Map-Reduce model
  - PGAS (Partitioned Global Address Space) programming model
  - APGAS (Asynchronous Partitioned Global Address Space)
- There is growing interest in new languages:
  - Chapel, X10, XscalableMP, UPC, Click,…
  - MPI for Java, Java bindings in OpenMPI
  - Java parallel streams

Motivation - Java

- Java is (the most) popular programming language
- For many students Java is first programming language
- Java is very popular in Big Data processing

- The parallelism in Java is limited to single JVM (single computer node)
- Recently OpenMPI introduced Java binding
  - It is still MPI style rather than Java style
  - Maintenance problem

- The Apache Spark / Apache Hadoop are designed for single parallelization schema: map-reduce

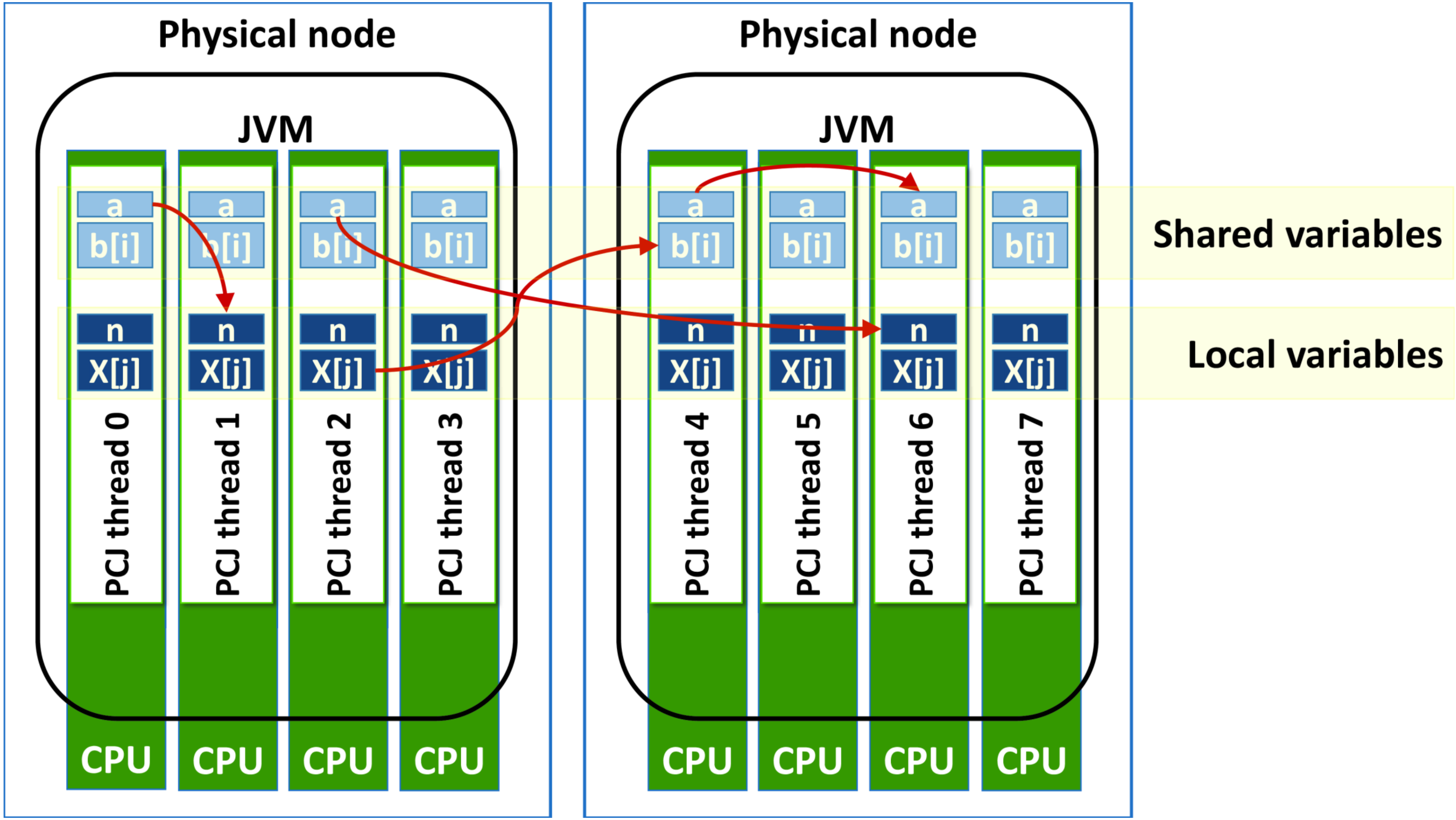
PCJ Library http://pcj.icm.edu.pl

Java library

- Designed based on PGAS
- Simple and easy to use
- Single jar file

- Does not introduce extensions to the Java language
  - no new compilers nor pre-processor
- Does not require additional libraries
- Works with Java 1.8, 1.9
  - Version for Java 1.7 available
- Good performance
- Good scalability (up to 4k nodes/ 200k cores of XC40)

PCJ – memory layout and communication



 M. Nowicki, Ł. Górski, P. Bała

PCJ - Parallel Computing in Java

Package: **`org.pcj`**

- StartPoint ← interface; indicate start class/method
- NodesDescription ← description of nodes
- PCJ ← contains base static methods
  - start / deploy ← starts application
  - myId / threadCount ← get thread identifier / thread count
  - registerStorage ← registers shared space
  - get / asyncGet ← get data from shared space
  - put / asyncPut ← put data to shared space
  - broadcast / asyncBroadcast ← broadcast data to shared space
  - waitFor ← wait for modification of data
  - barrier / asyncBarrier ← execution barrier
  - join ← create/join to subgroup
- PcjFuture<T> ← for notification of async method
- @Storage ← shared space annotation
- @RegisterStorage ← registering shared space

HelloWorld.java

```
import org.pcj.NodesDescription;
import org.pcj.PCJ;
import org.pcj.StartPoint;
public class HelloWorld implements StartPoint {
public static void main(String[] args) {
  PCJ.deploy(HelloWorld.class, new NodesDescription("nodes.txt"));
}
public void main() throws Throwable {
  System.out.println("I am " + PCJ.myId()
                     + " of " + PCJ.threadCount() );
}
} //end of class
```

Example.java Basic communication

```
@RegisterStorage(Example.Shared.class)
public class Example implements StartPoint {
@Storage(Example.class)
enum Shared { a, table }
public int a;
public double[] table;

public void main() throws Throwable {
   …
   if (PCJ.myId() == 0) PCJ.put(42, 4, Shared.table, 10);
   if (PCJ.myId() == 1) { t = PCJ.get(3, Shared.table); }
   if (PCJ.myId() == 0) PCJ.broadcast(a, Shared.a);
   PCJ.waitFor(Shared.a);
```

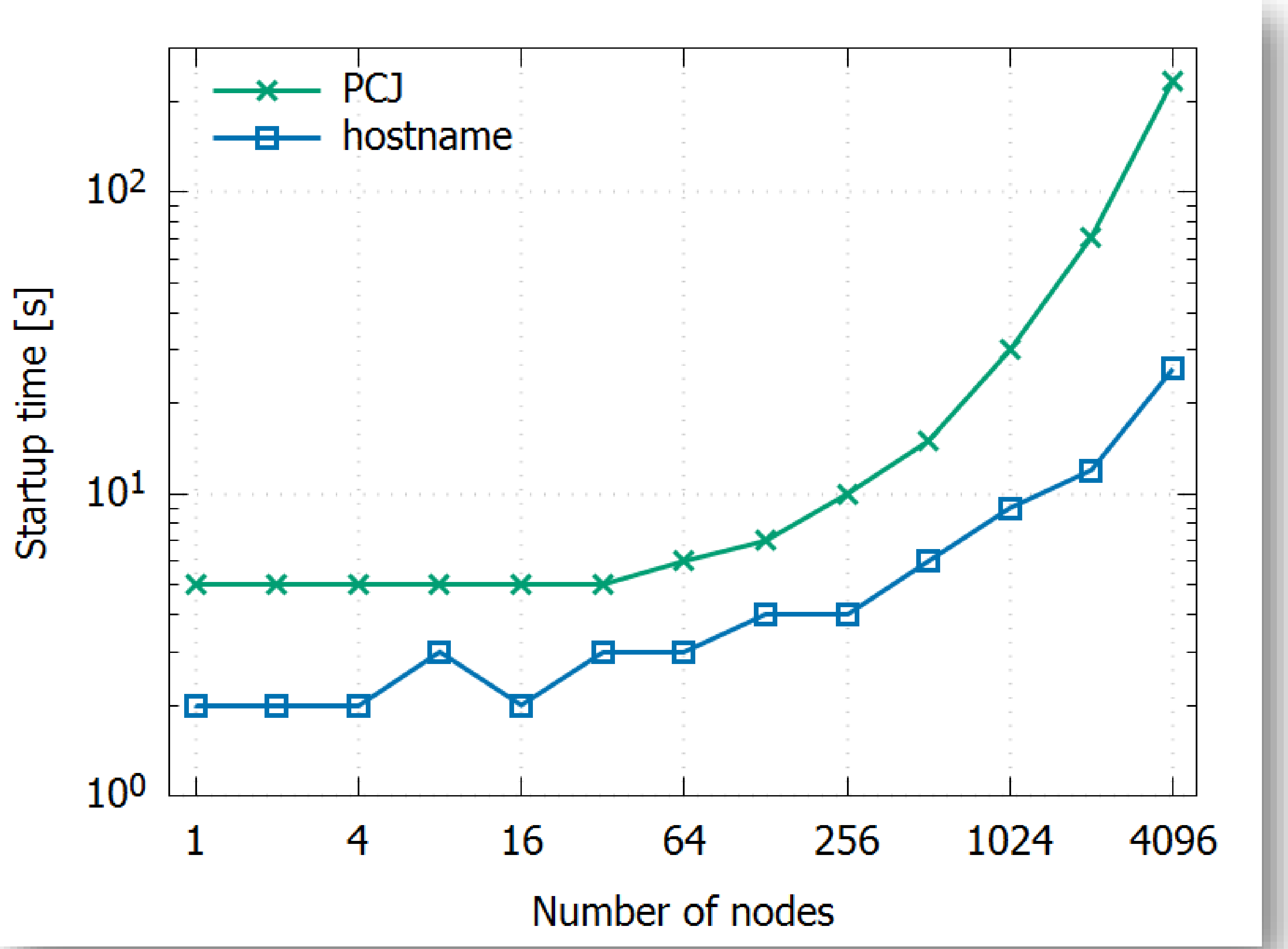
Hardware and tools

- PCJ
  - version 5.0.6
- Oracle Java
  - java version 1.8.0_51
- GraalVM 1.0-RC1
  - openjdk version 1.8.0_161
- Java / OpenMPI
  - version 3.0.0

- **Cray XC40 at ICM**
  - 1084 nodes
  - 2 Intel Xeon E5-2690 v3 @ 2.60GHz
  - 128GB RM
  - Cray CC/8.6.4
  - cray-mpich/7.6.3

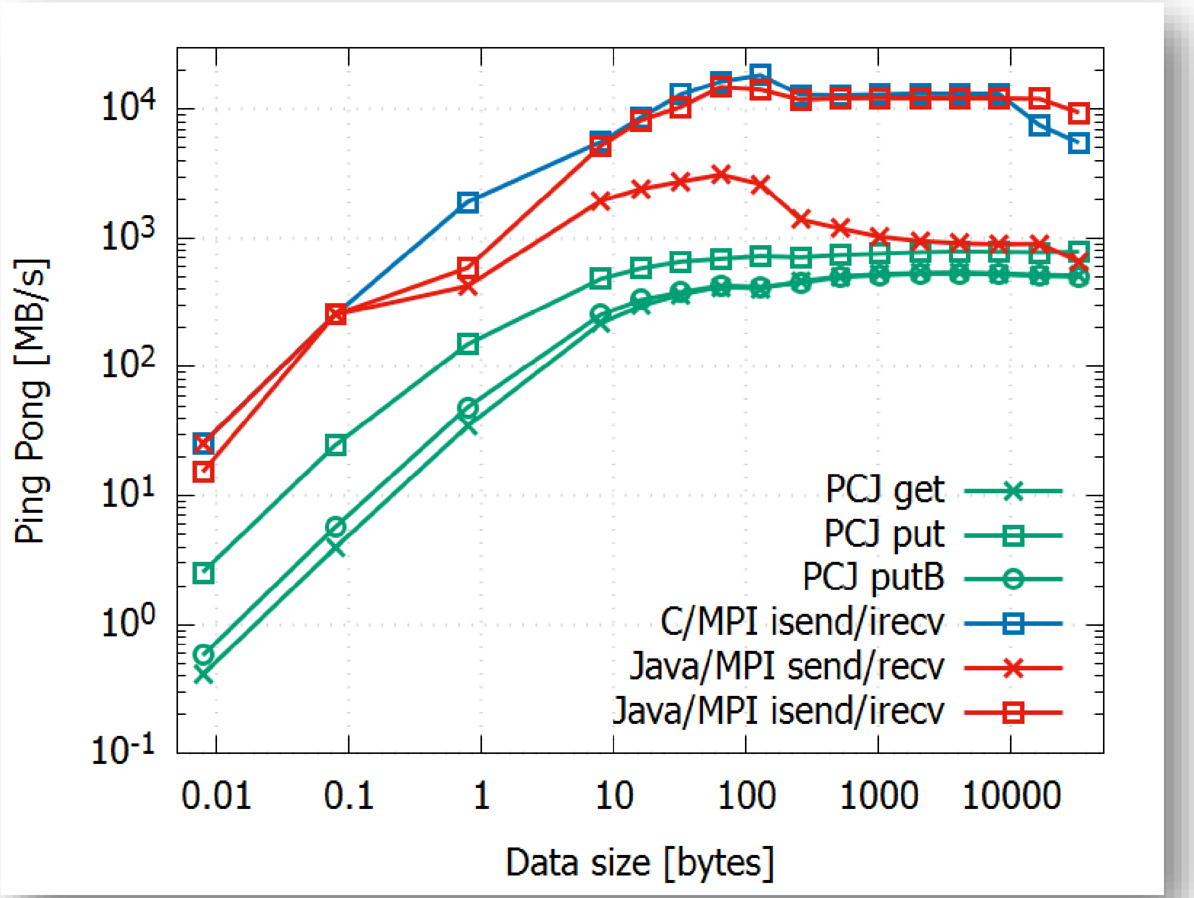
- **Cray XC40 at HLRS**
  - 7712 nodes
  - 2 Intel Xeon E5-2680 v3 @ 2.50GHz
  - 128GB RAM
  - Cray CC/8.6.0
  - cray-mpich/7.6.0

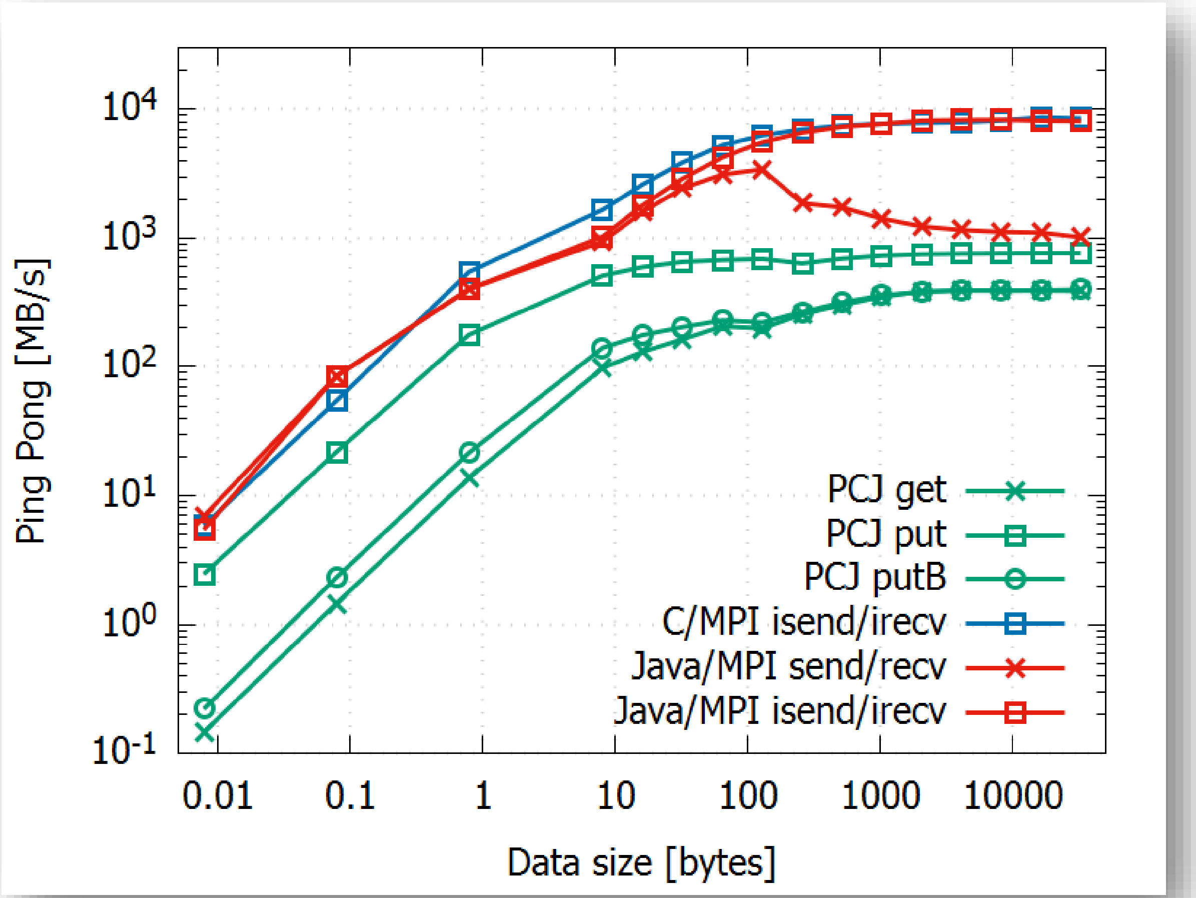
PCJ start-up time



- PCJ works with resource managements systems (slurm, PBS, LL, …)
- List of nodes can be provided on startup (as file or arguments)
- runs on Cray XC40 at HLRS

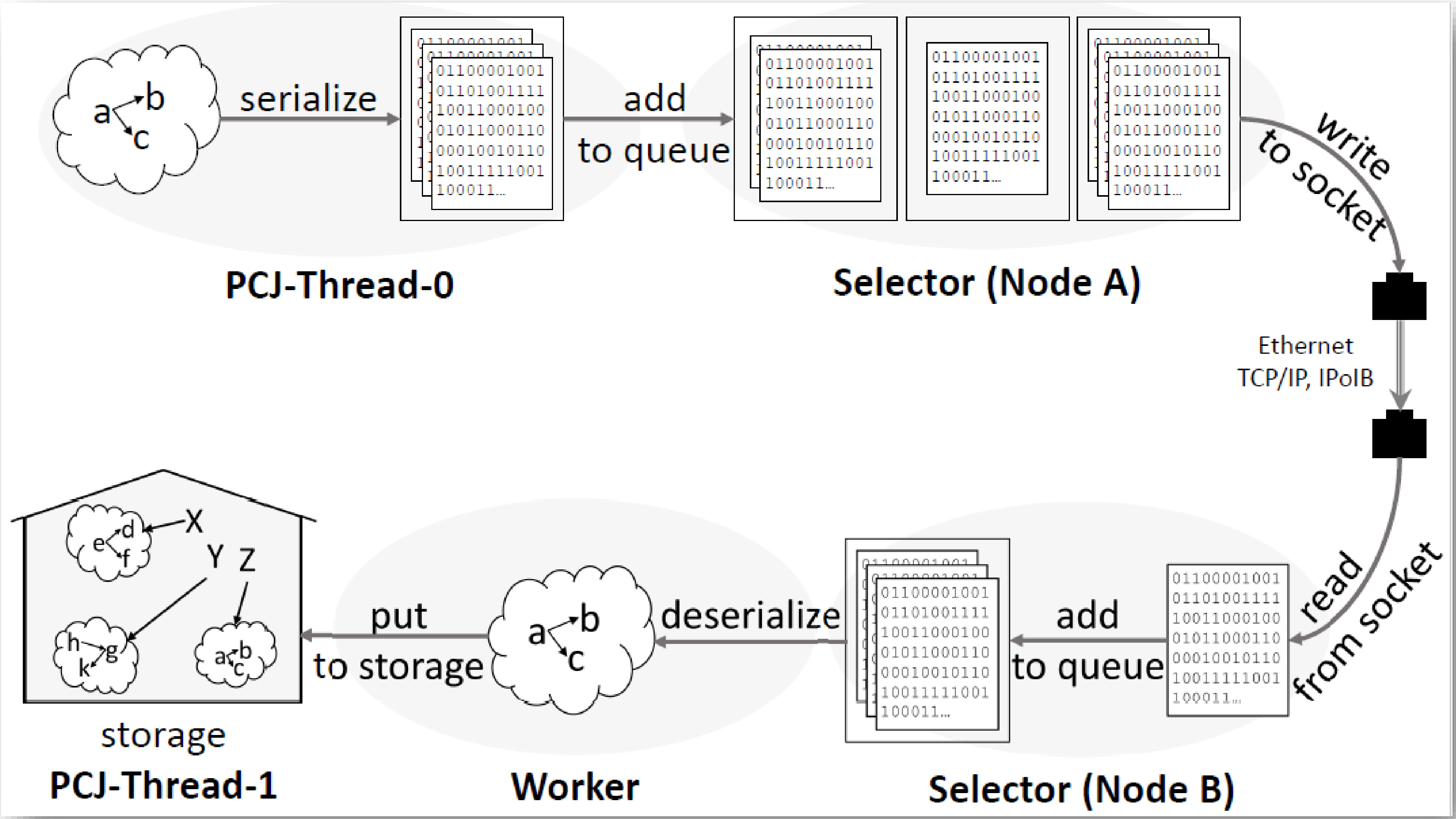
Ping – pong (1 node, 2 nodes)





- PCJ uses Java sockets for communication. MPI uses Cray Aries
- For large data the performance is degradated for PCJ and Java/MPI because of data copying (Java problem)

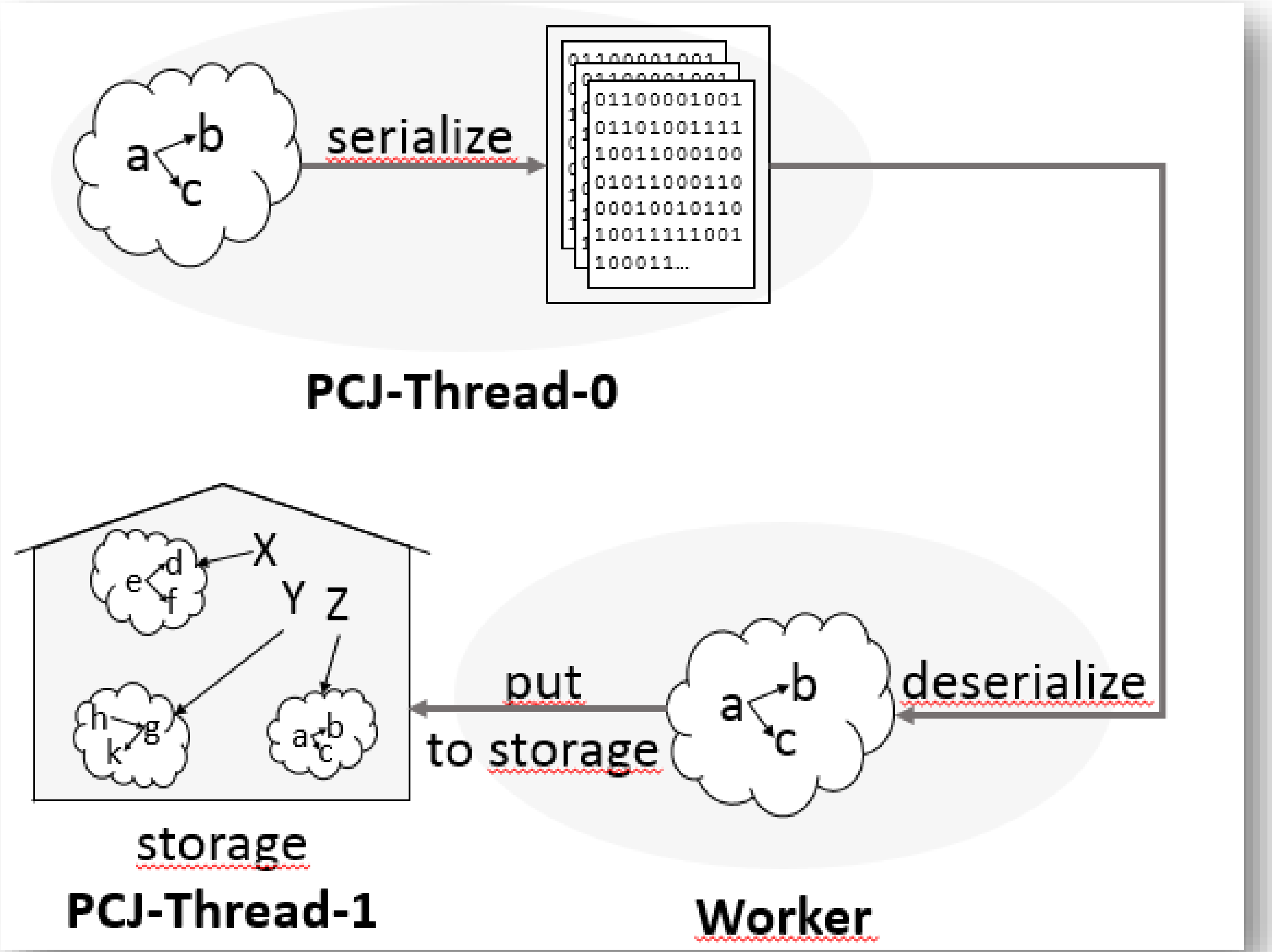
Intranode communication (between nodes)



From: Nowicki, M., Górski, Ł., Bała, P. PCJ – Java library for highly scalable HPC and Big Data processing (in review)

- PCJ threads can run on different JVM's
- PCJ uses Java sockets for communication (java.nio.*)
- Data has to be serialized and deserialized

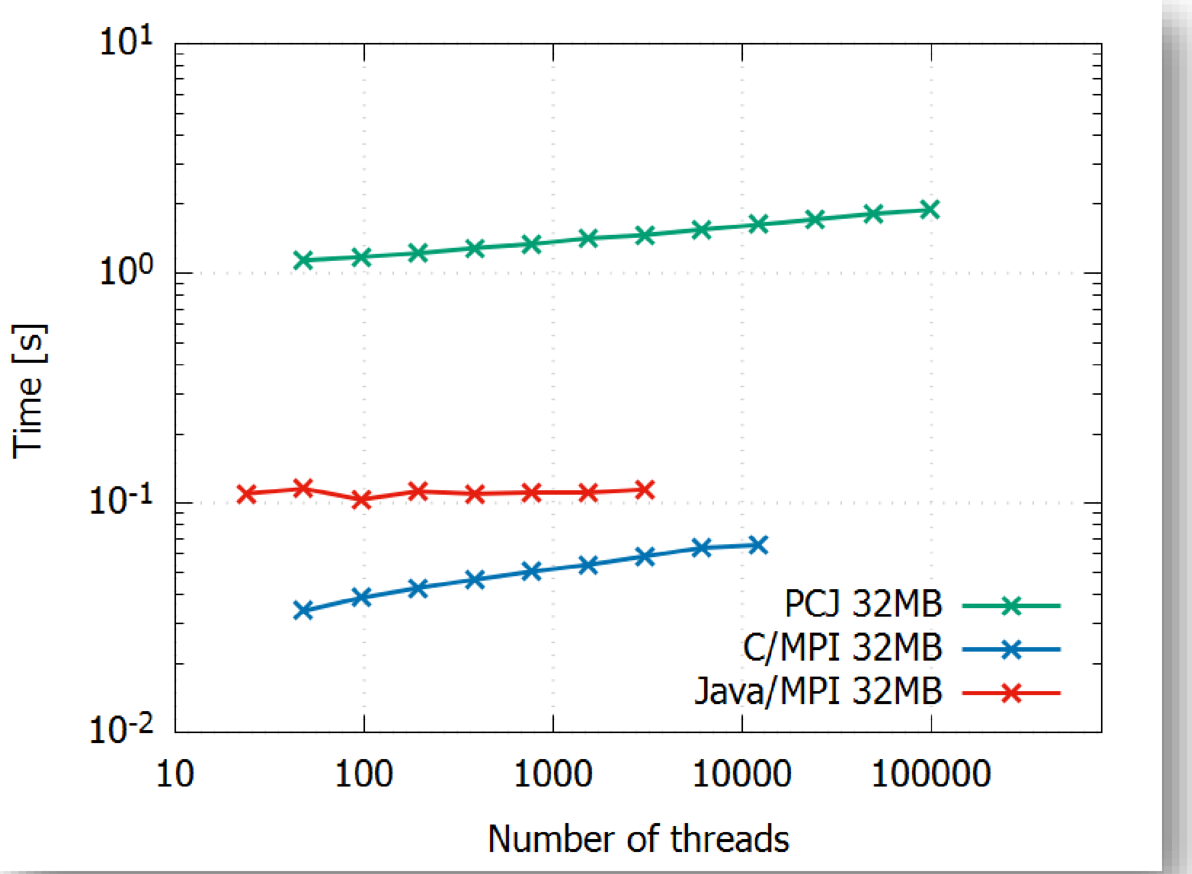
Internode communication (threads within the same JVM)

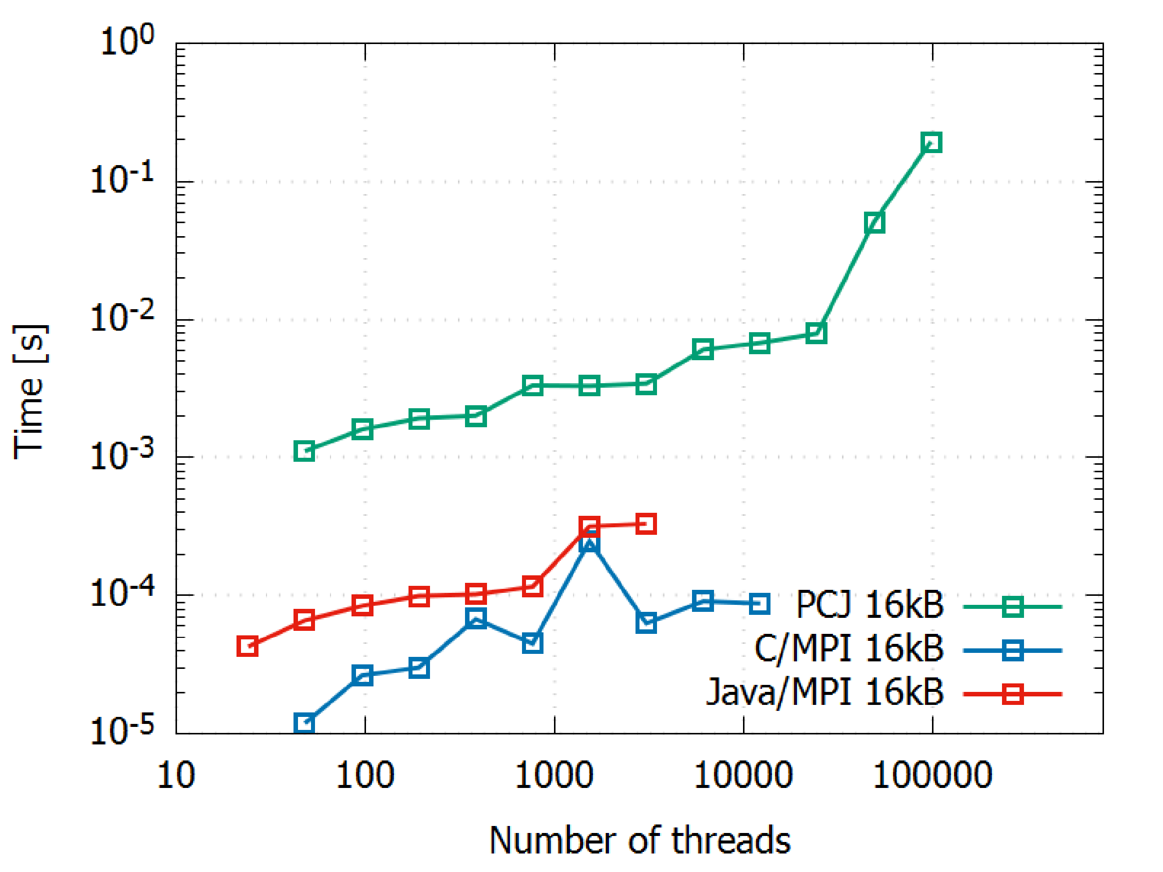


From: Nowicki, M., Górski, Ł., Bała, P. PCJ – Java library for highly scalable HPC and Big Data processing (in review)

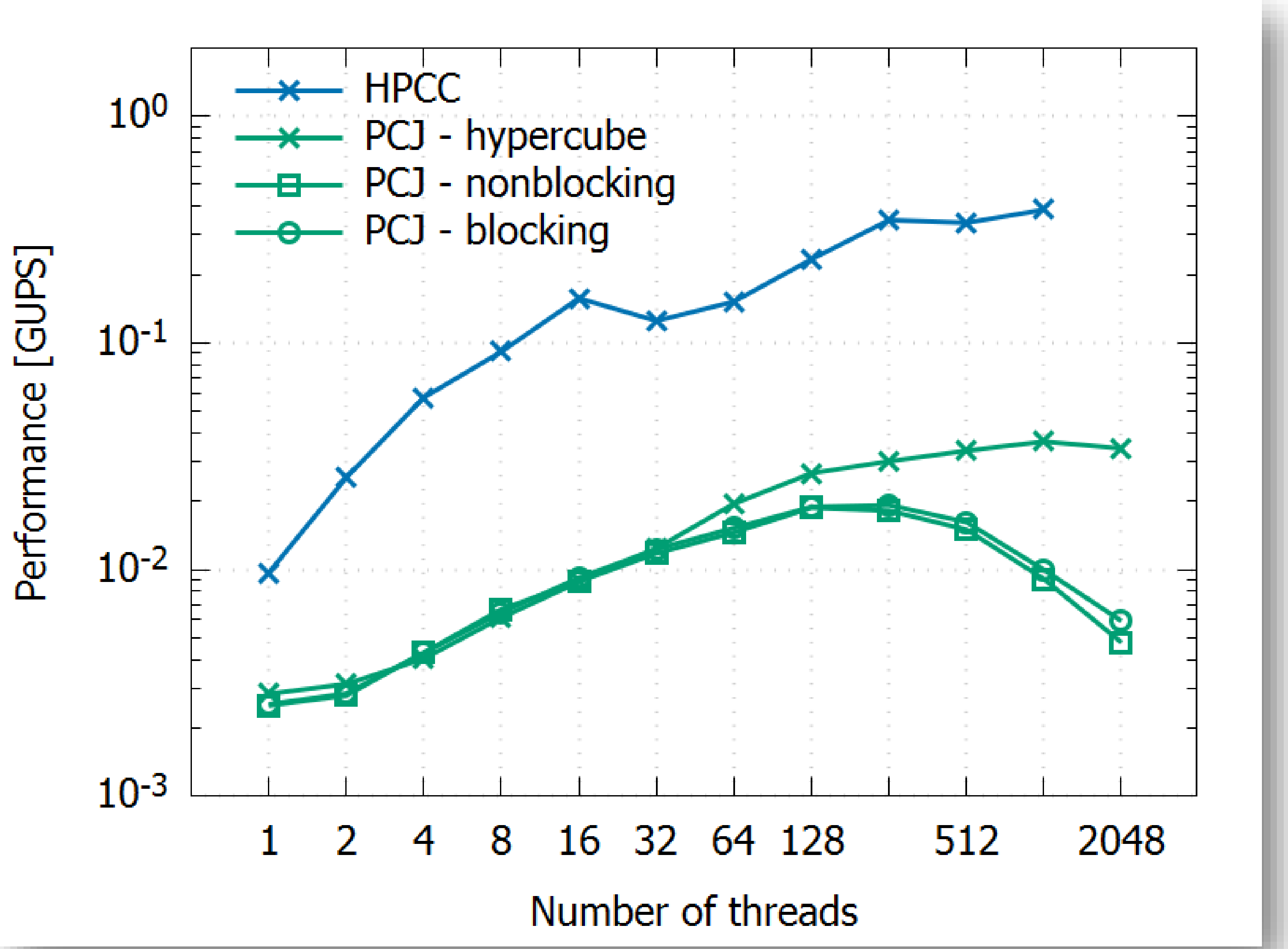
- Java Concurrency for internode communication
- Data has to be serialized and deserialized

Broadcast 32MB (left) 16kB (right)



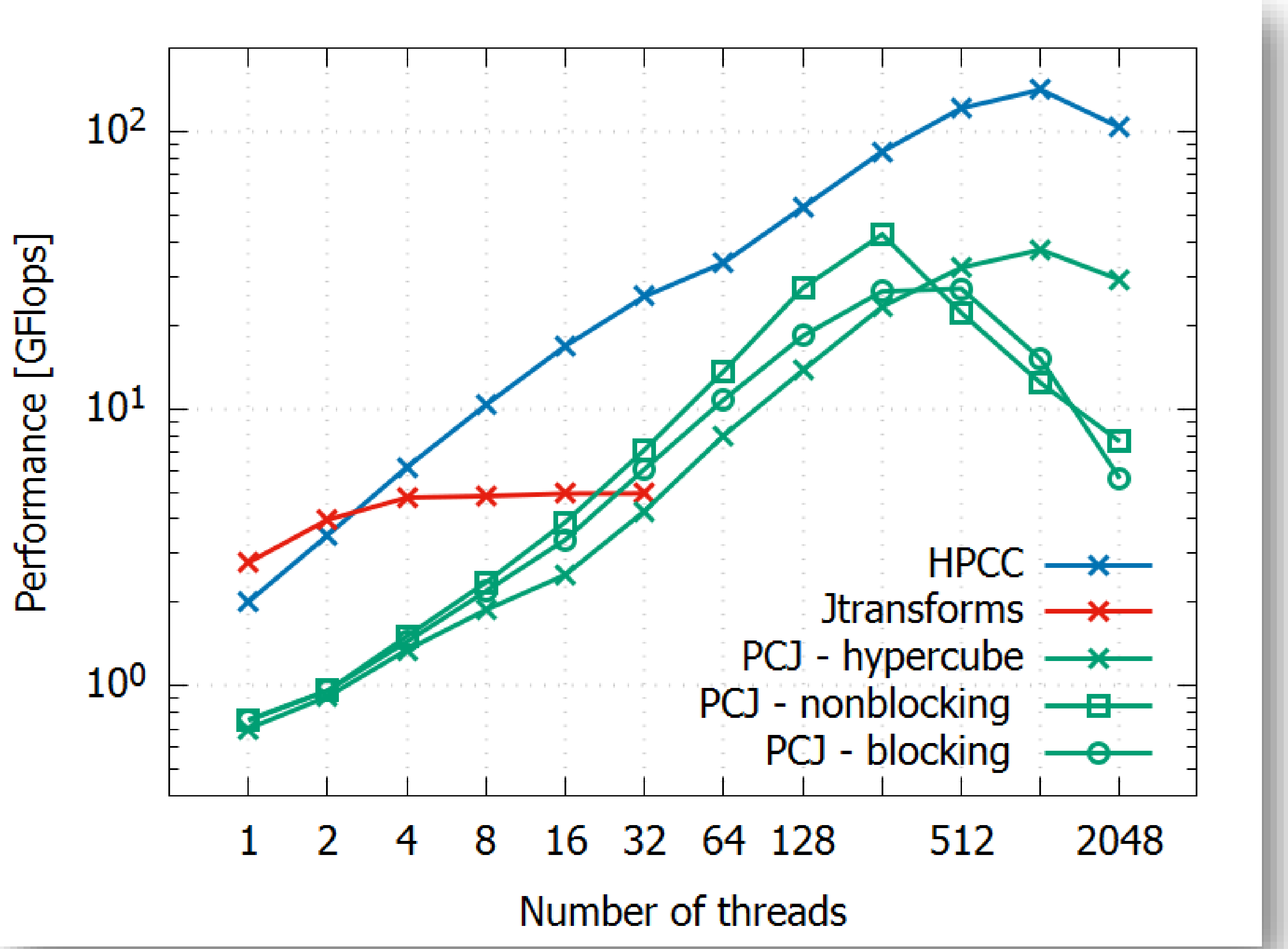


Random Access



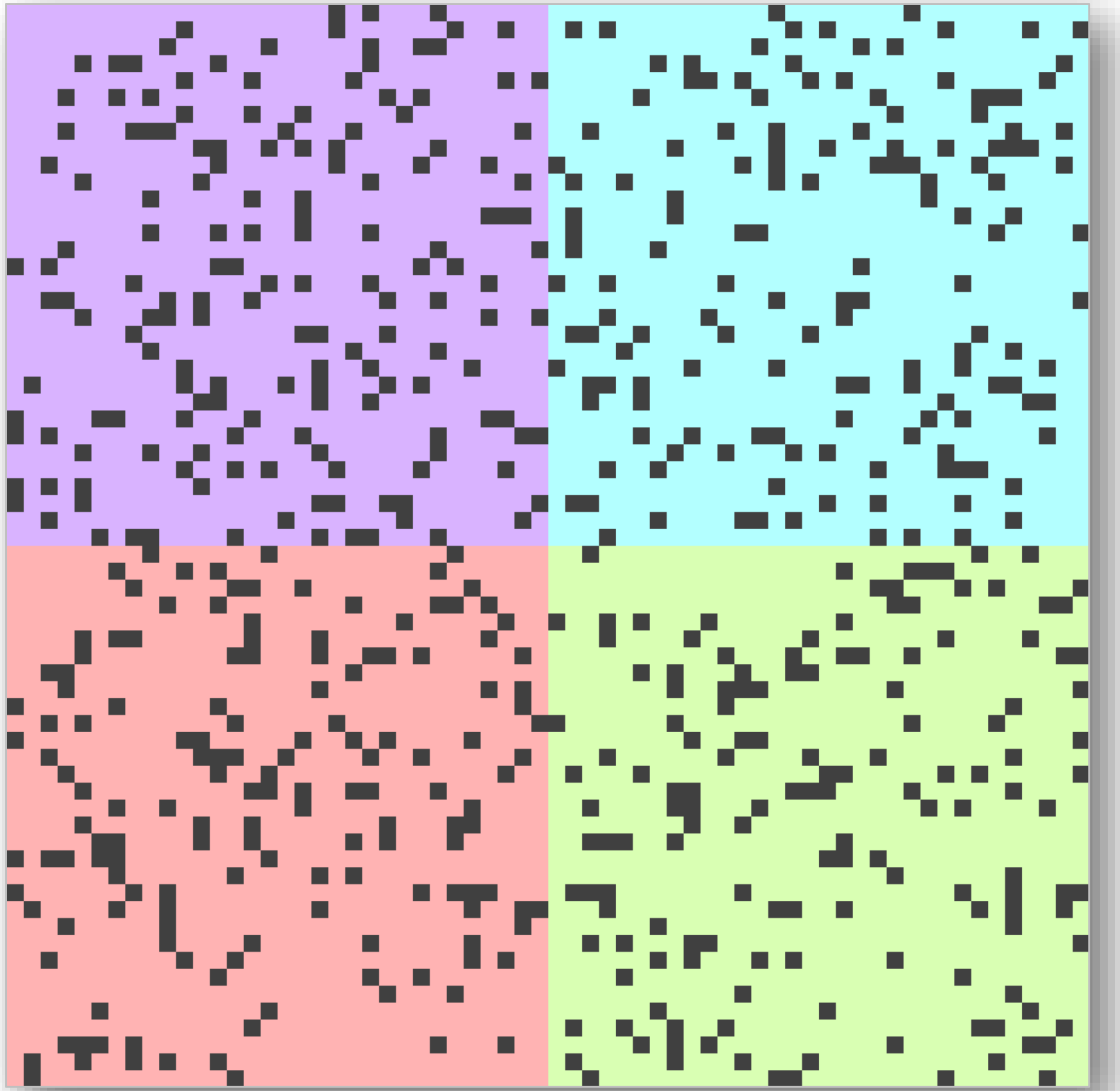
- Implementation according to HPCC rules
- Relatively small buffer (1024 elements)
- Java performance needs improvements

FFT 1D



- All to all communication
- Best performance for hypercube communication algorithm
- Scalability simillar to HPCC C/MPI implementation

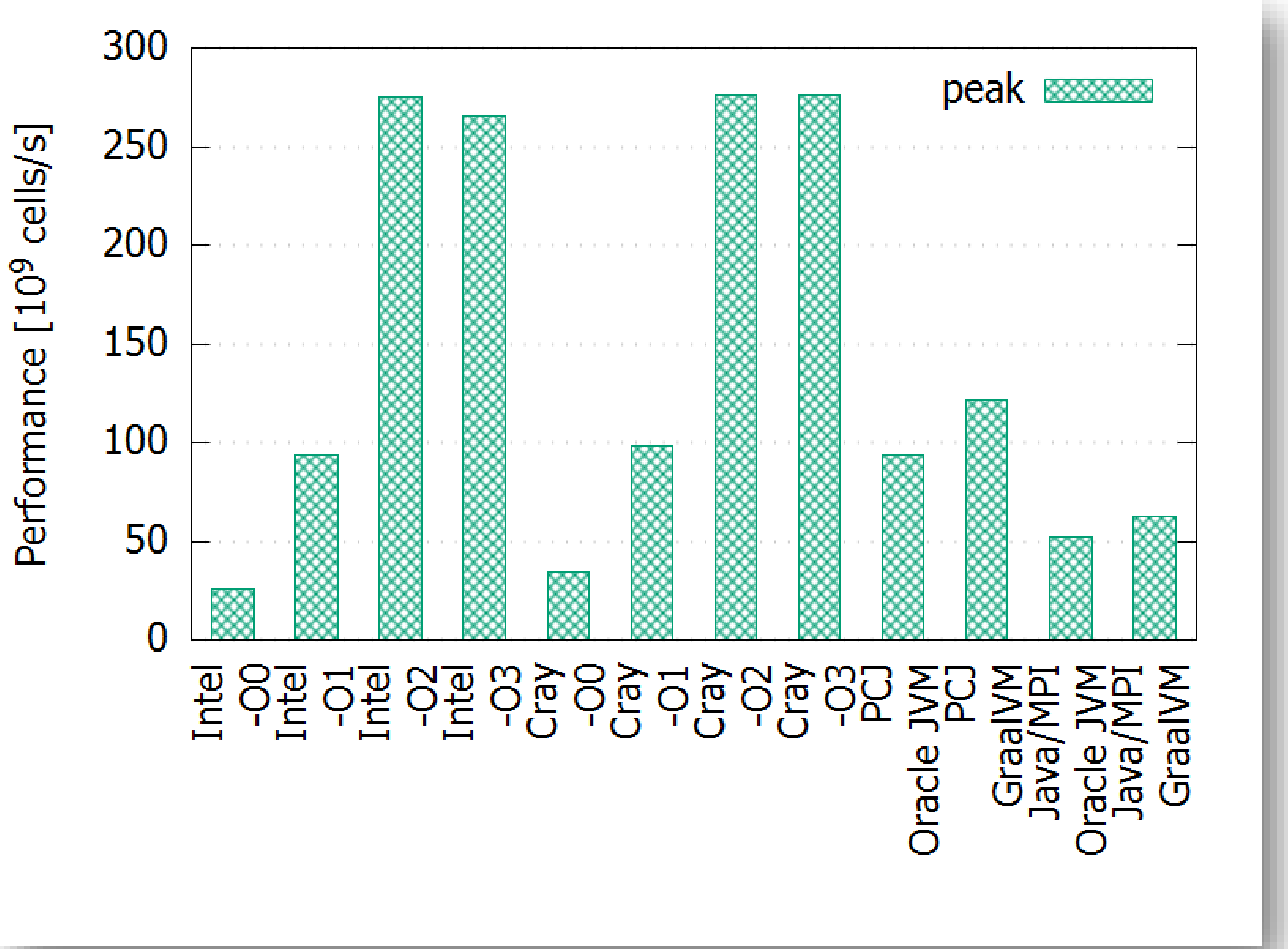
Game of Life (2D stencil)



- Simple code
- Halo exchange
- Asynchronous communication

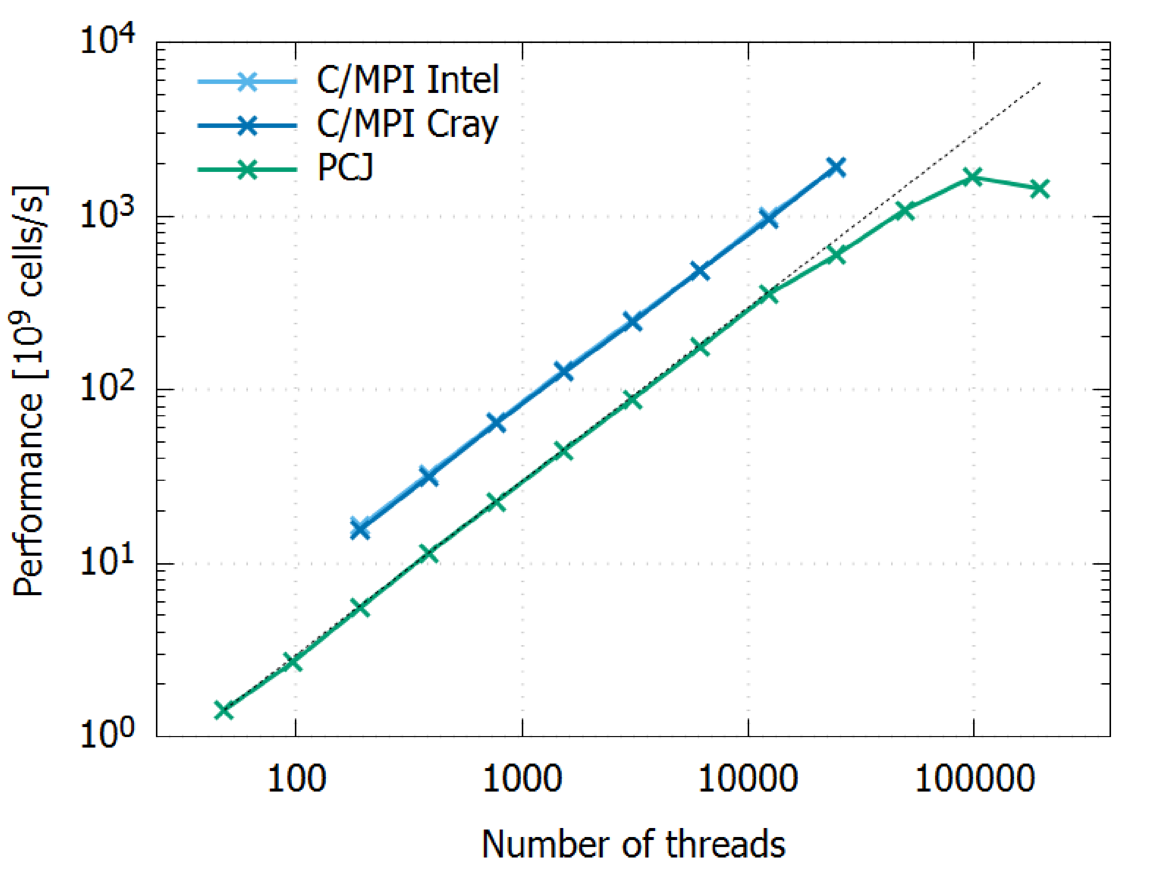
```
PCJ.asyncPut(sendShared.W,
    PCJ.myId() - 1, Shared.E);
// process inside cells

PCJ.waitFor(Shared.E);
board.set(colsPerThread + 1,
  row, recvShared.E[row - 1]);
// process border cells
```

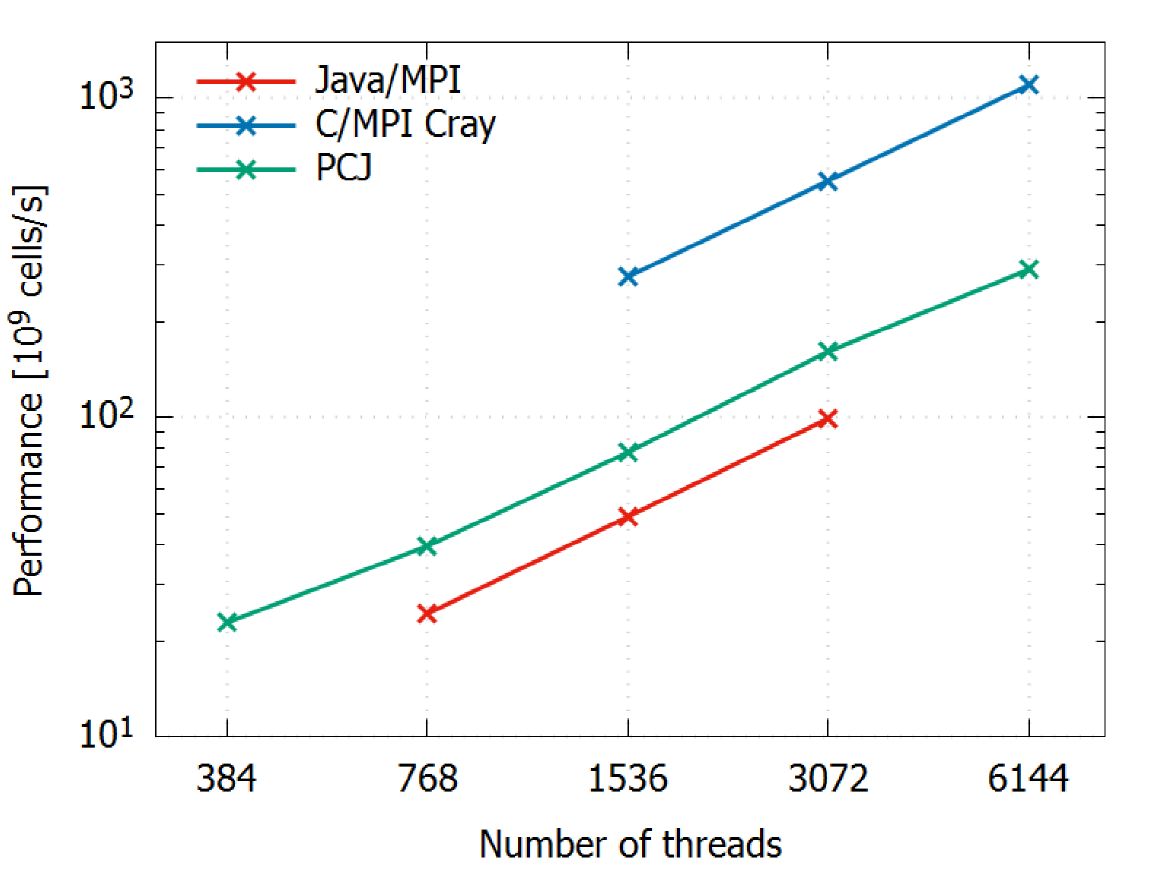
Game of Life (2D stencil) performance comparison



- 2,419,200 x 2,419,200 cells running at 64 nodes of Cray XC40 at ICM.
- GraalVM improves performance by 12% for PCJ and 20% for Java/MPI
- Java/PCJ is almost 2x faster than Java/MPI
- C/MPI is 2x faster than Java/PCJ

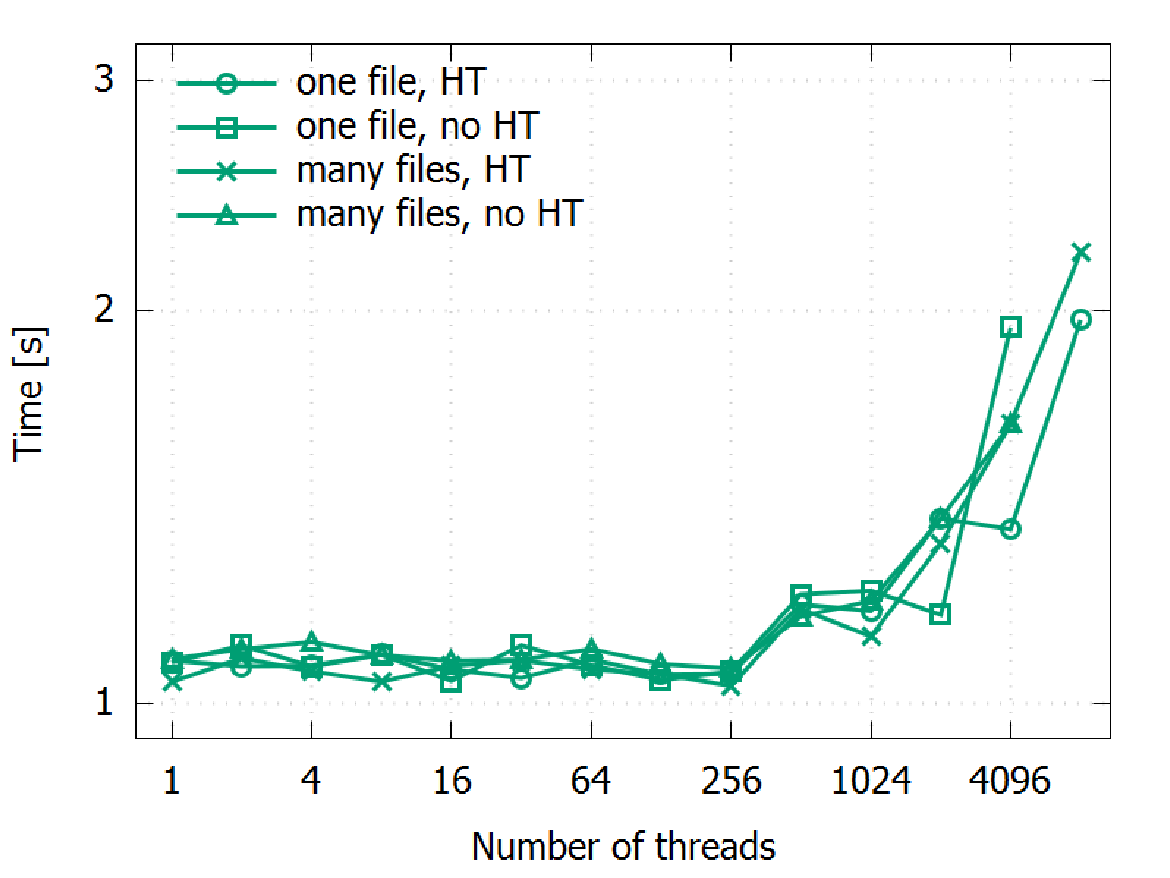
Game of life - scalability

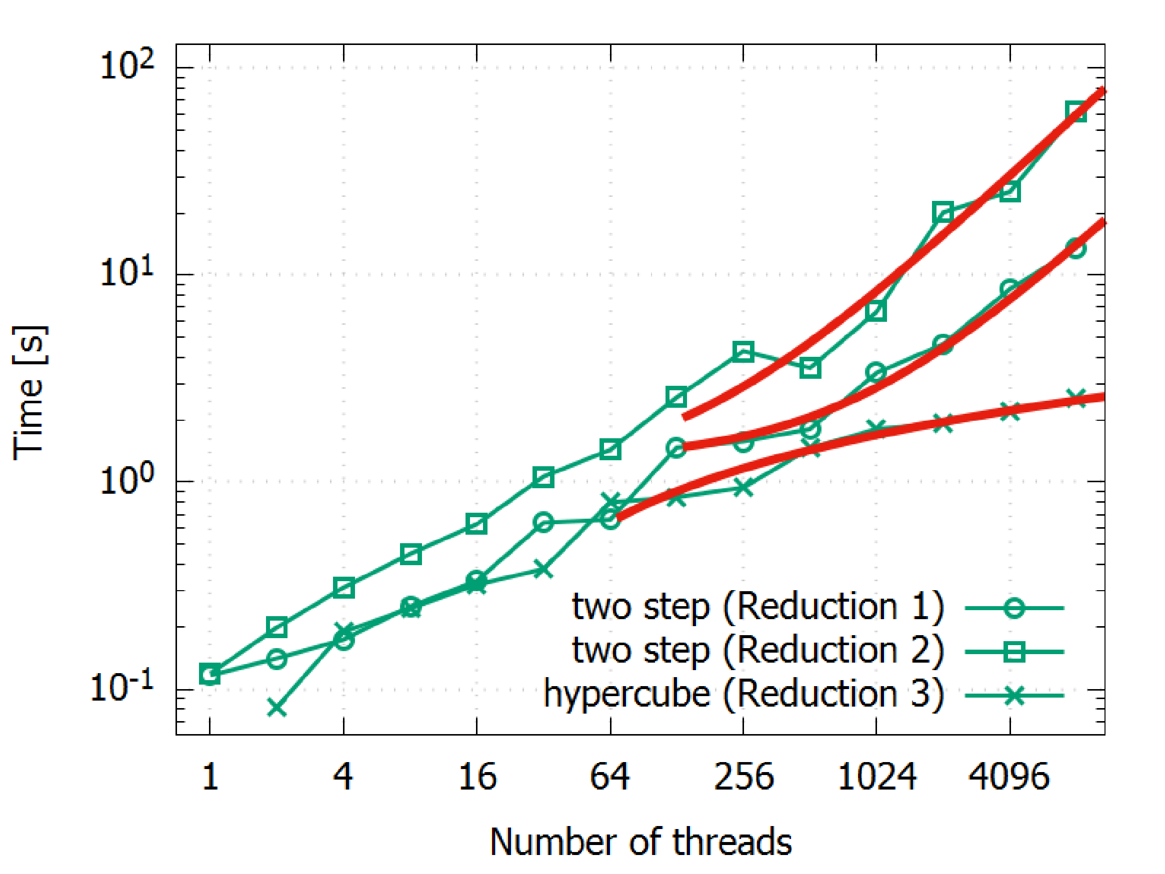




- PCJ application runs on 200k+ cores (4096 nodes of Cray XC40)
- Strong scalability results (for week scalability see paper)

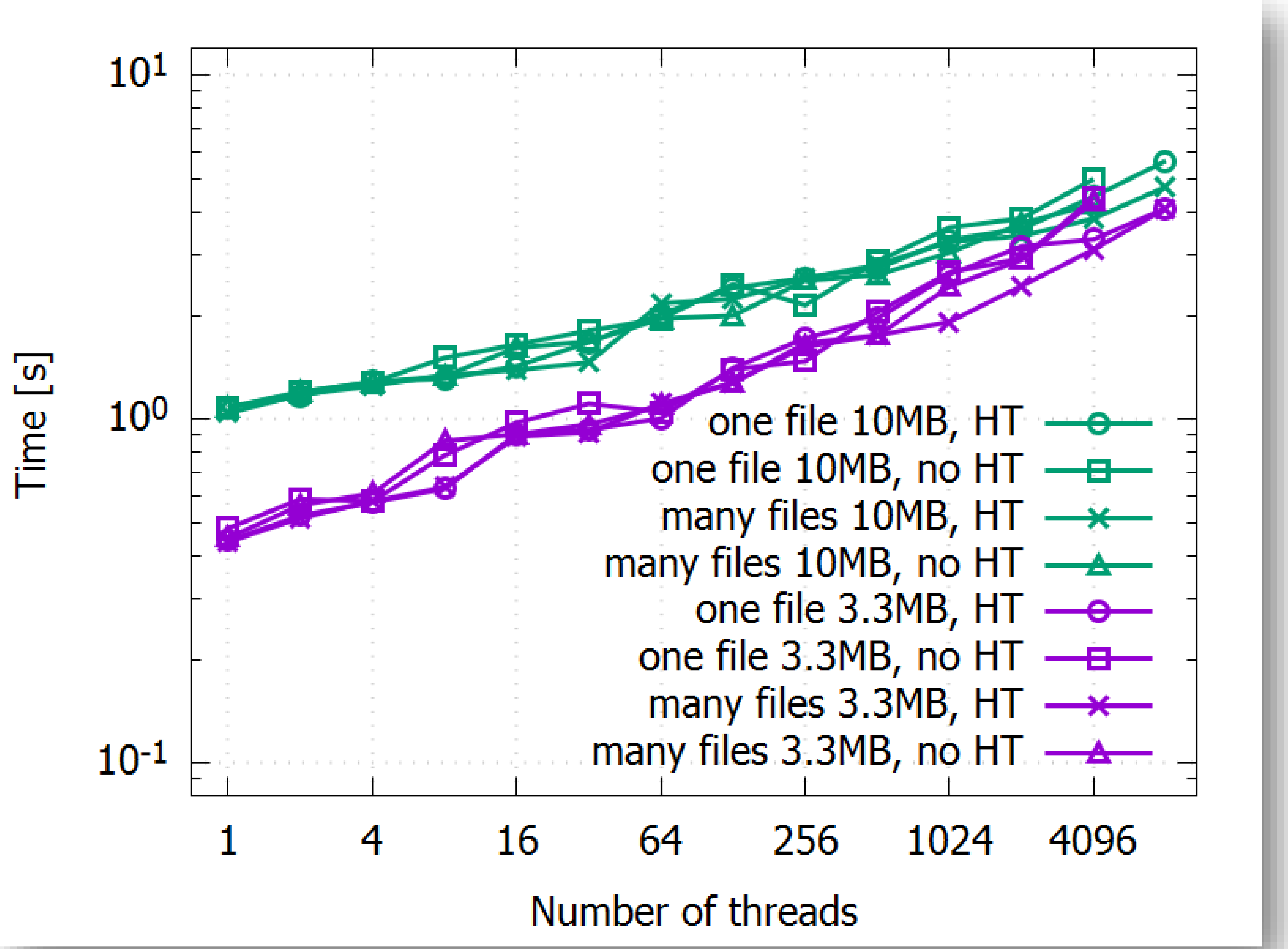
Wordcount: map (left) - reduce (right)





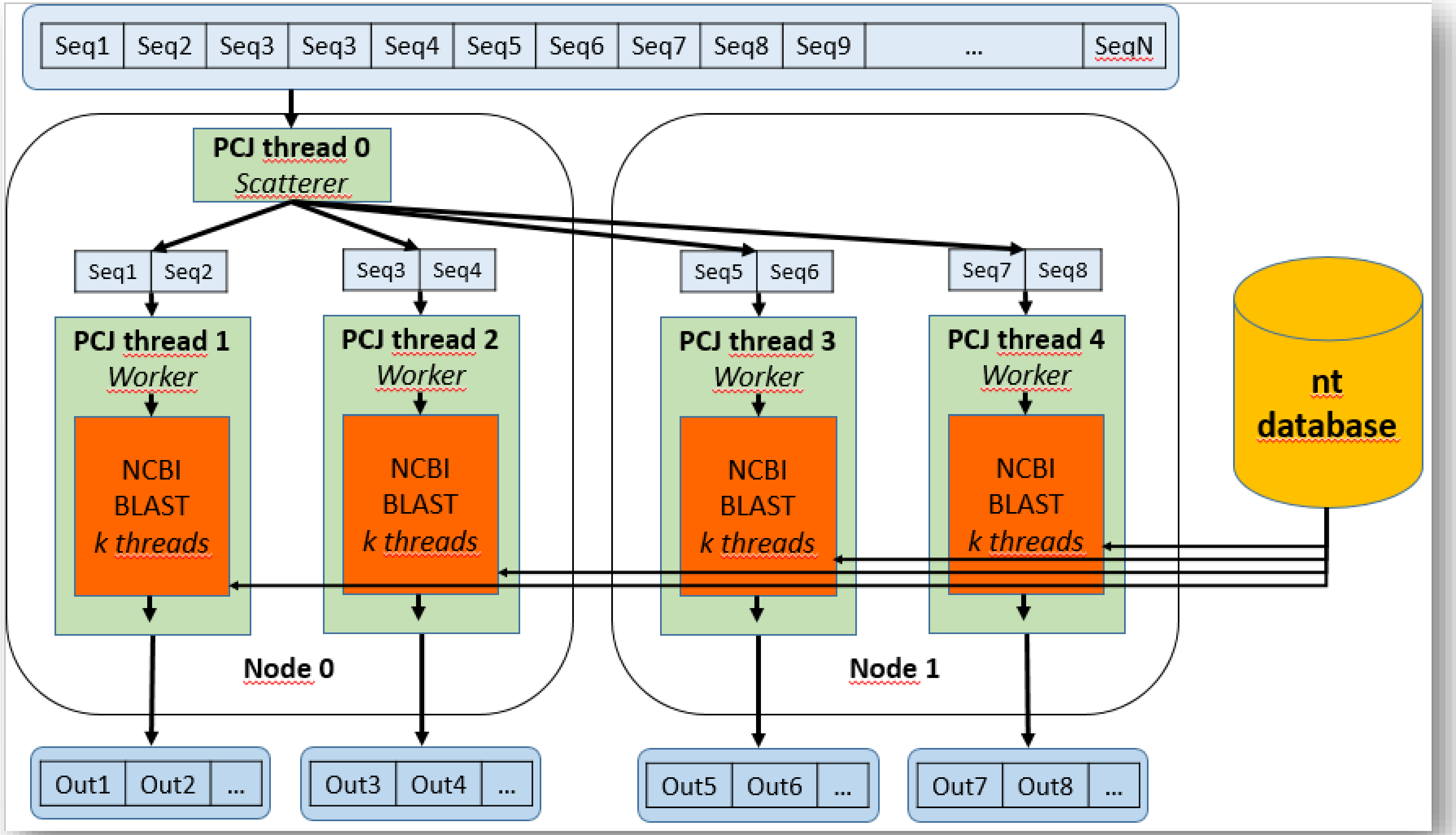
- Weak scalability (the same size of file at each thread)
- Different scalability of reduction (O(n), O(log(n)) for hypercube)

Wordcount (total time)



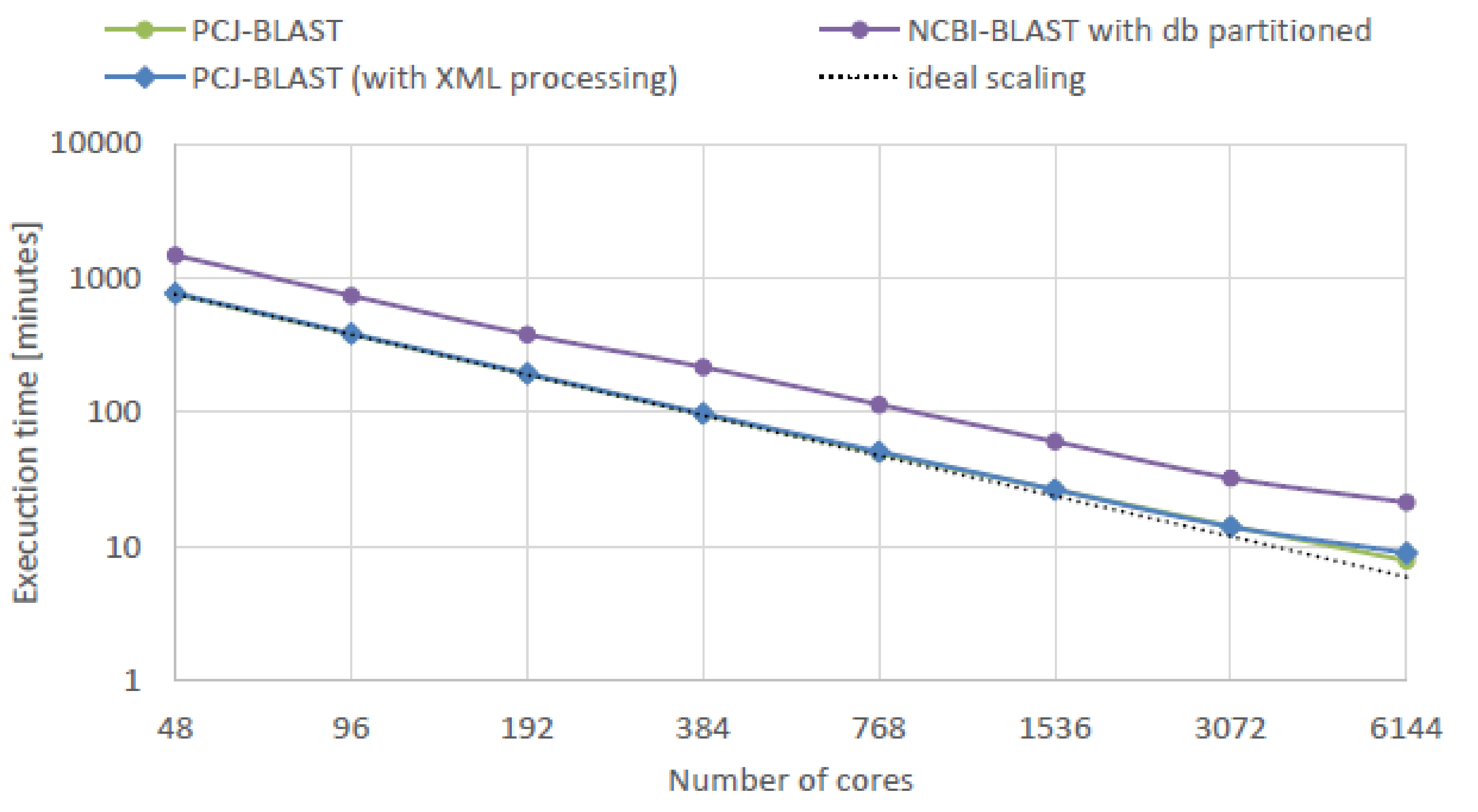
- 10MB and 3MB per thread (weak scalling)
- Performance dominated by the reduction operation

PCJ-BLAST - J. Comput. Biol. 2018 (in press)



- DNA sequenca alignment
- PCJ-Blast – wrapper to NCBI-BLAST
- Performs dynamic load-balancing
- 2x faster than partition of the query and database

PCJ-BLAST - J. Comput. Biol. 2018 (in press)



- DNA sequenca alignment
- PCJ-Blast – wrapper to NCBI-BLAST
- Performs dynamic load-balancing
- 2x faster than partition of the query and database

Piotr Bała bala@icm.edu.pl http://**pcj.icm.edu.pl**

PCJ development:

Marek Nowicki (WMiI UMK)
Michał Szynkiewicz (UMK, ICM UW) – fault tolerance

PCJ Tests, examples:

Łukasz Górski (UMK, ICM UW)
Magdalena Ryczkowska (UMK, ICM UW)

Acknowledgments:

NCN 2014/14/Z/ST6/00007 (CHIST-ERA grant),
EuroLab-4-HPC,
PRACE for awarding access to resource HazelHen at HLRS,
ICM UW (GB65-15, GA69-19)