Benchmarking Java on Ethernet Cluster

Benchmarking PCJ – a library for parallel computing in Java

Marek Nowicki

Nicolaus Copernicus University in Toruń, Poland

NICOLAUS COPERNICUS UNIVERSITY IN TORUŃ
Faculty of Mathematics and Computer Science

Introduction

The aim of this project was to:
- check the scalability of parallel, network-intensive microbenchmarks and application written in Java
- using the PCJ library
- on the 1000 Core Cluster – Ethernet-based cluster

The PCJ Library

The PCJ library (`https://pcj.icm.edu.pl`) [1]
- is HPC Challenge 2014 award-winning Java library for high-performance parallel computing
- allows writing applications that utilize both multiple cores of a node and multiple nodes of a cluster
- implements PGAS (Partitioned Global Address Space) paradigm
- complies with Java standards and works with the newest Java version (currently Java 13)
- is an open source library (BSD license) (`https://github.com/hpdcj`)
- is distributed as a single jar (can be downloaded from Maven Central Repository)

PCJ – Case Studies

- Finding parameters of the neural network simulating connectome of C. Elegans [2]

Image source: `http://web.science.uu.nl/developmentalbiology/boxem/images/adult_worm.jpg`

- Parallel BLAST execution – searching for similar DNA sequences in the genome database [3]

- Traversing of a large sparse graph that imitates real scale-free networks with small-diameter [4]

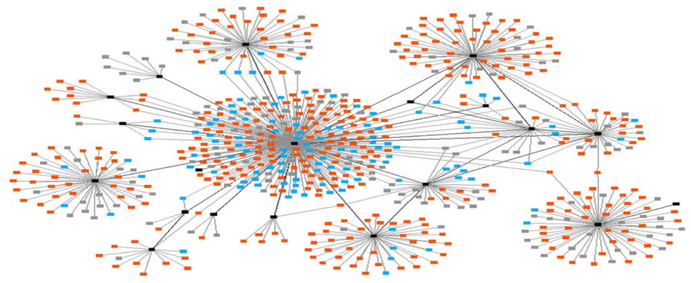


Image source: `https://www.flickr.com/photos/ajc1/2553555562`

. . . to name only few.

Hardware

1000 Core Cluster, each node:
- 4× Intel Xeon E7-4870 processors (@ 2.40 GHz; 10 cores, 2 HT)
- 1 TB of RAM
- 10 Gigabit Ethernet (Intel 82599ES 10-Gigabit Ethernet Controller)
- NAS mounted drive with 4 TB capacity

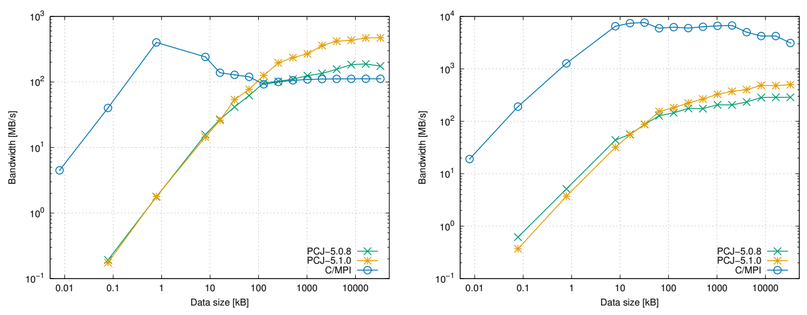
Software

- Ubuntu Linux in version 16.04.4 LTS *(Xenial Xerus)*
- SLURM (18.08.7)
- C/MPI: *mpicc* – *GCC* (5.4.0 20160609) & *MPICH* (3.2)
- Oracle JDK 13
- Apache Hadoop 3.2.1

Ping-pong: Overview

- Ping-pong tests the bandwidth between a pair of threads
- One thread sends a `double[]` 100-times and waits for confirmation
- The total time needed for the operation is measured
- The minimum average value from 5 repeats is taken as a result
- The size of array varies from 1 element (8 B) to 4,194,304 elements (32 MB)

Ping-pong: Results



(a) Ping-pong using 2 nodes, 1 thread on each node.

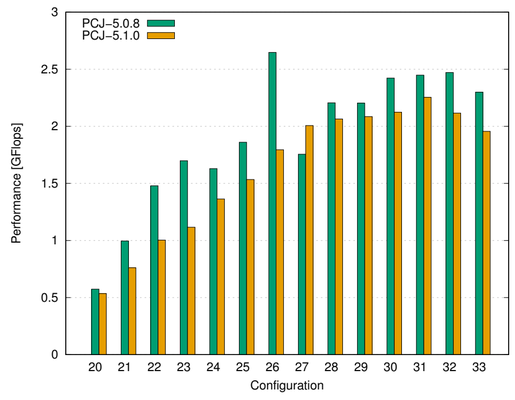
(b) Ping-pong using 1 node with 2 thread.

Figure 1: Result of ping-pong microbenchmark.

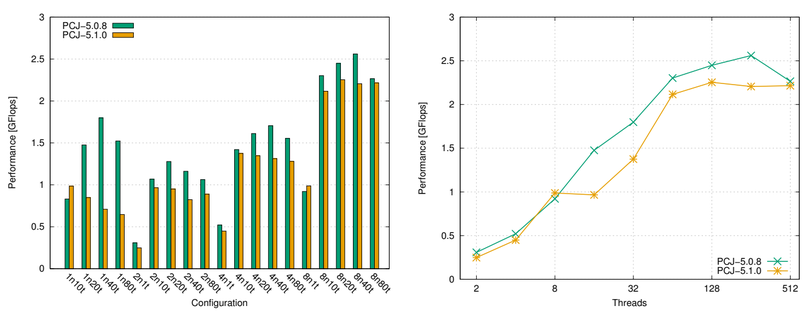
FFT: Overview

- FFT (Fast Fourier Transform) is an algorithm that computes the discrete Fourier transform (DFT)
- The implementation of the algorithm in the PCJ is based on the PGAS implementation developed for Coarray Fortran 2.0 [5]
- The used FFT algorithm requires to use a number of threads equal to a power of 2, so the cores of the cluster nodes were not totally used for computational purposes

FFT: Results



(a) Performance results for FFT on an increasing number of elements to process (2^N) running on 8 nodes and 20 threads on each node.



(b) Performance for FFT on 2^{30} elements. *XnYt* stands for *X* nodes and *Y* threads on each node.

(c) Scalability for FFT on 2^{30} elements with an increasing number of threads.

Figure 2: Performance results of FFT application.

TeraSort: Overview

- The *TeraSort* benchmark requires creating proper input data using `teragen` application
- Input file contains multiple 100-bytes records that consist of 10-bytes key and 90-bytes value
- Records are sorted using key values
- Output is written to one file
- The key is compared using a standard byte to byte comparison

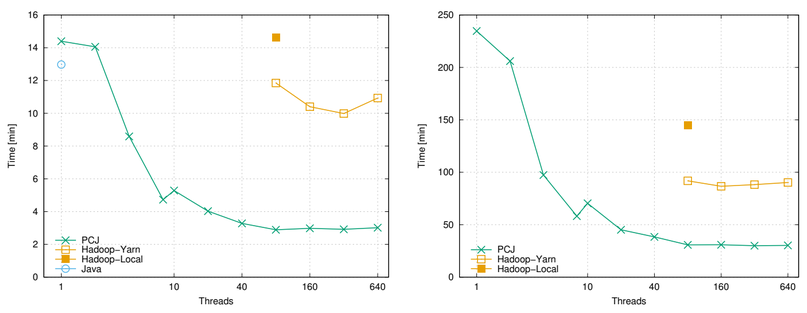
TeraSort: Hadoop preparation

The Hadoop implementation was the one provided with the *Apache Hadoop* in the *examples* jar file.
- one SLURM job for *Hadoop Master*: starts *namenode*, *datanode* and *resourcemanager* daemons
- computational nodes were started on other SLURM job by *srun*ning *nodemanager* in background, just before submitting a job to the *resource manager*
- computational nodes use the *Hadoop Master* as a host for a default file system (*hdfs://*) and for resource manager.

TeraSort: PCJ implementation

The PCJ implementation is divided into 5 stages, similar to Edahiro's *Mapsort* described in [6]:
1. calculating pivots, the selected records from the input, that will be used for dividing input data into buckets
2. reading input and placing records into proper buckets
3. exchanging buckets data between threads
4. sorting data stored in buckets
5. saving data from buckets to one single output file; each thread has to write to file in a correct order

TeraSort: Results



(a) 10^8 elements

(b) 10^9 elements

Figure 3: Time needed to sort data depending on the allocated thread number.

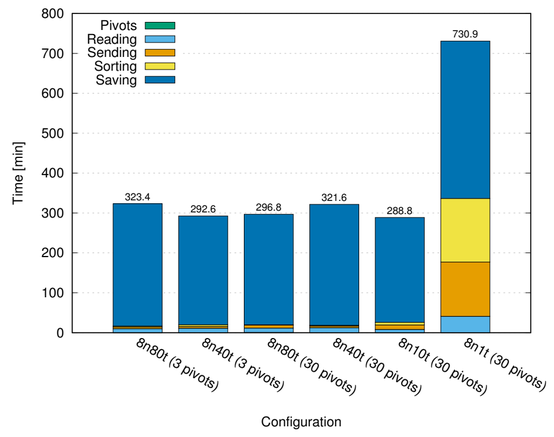


Figure 4: Time needed to sort 10^{10} 100-bytes records (about 1 TB of data) on various runtime configurations using PCJ. *XnYt* stands for *X* nodes and *Y* threads on each node.

Conclusions

The results show good and very good performance of the PCJ library on the cluster. However, there are some weaker parts of the library like intranode communication. Moreover, the source code for FFT should be examined to optimize it for the new version of the library.
The performance results of *TeraSort* application are very good.
The performance of Apache Hadoop prompts to an investigation of the better Hadoop configuration.

Acknowledgements

Special thanks to:
- Future SOC Lab, Hasso Plattner Institute for Digital Engineering for awarding access to the 1000 Core Cluster.
- colleagues Łukasz Górski, Łukasz Mikulski, and Jakub Narębski for their help during preparing and running benchmarks.

References:

[1] M. Nowicki, Ł. Górski, P. Bała, PCJ–Java Library for Highly Scalable HPC and Big Data Processing, in: 2018 International Conference on High Performance Computing & Simulation (HPCS), IEEE, 2018, pp. 12–20.

[2] Ł. Górski, P. Bała, F. Rakowski, A case study of software load balancing policies implemented with the PGAS programming model, in: 2016 International Conference on High Performance Computing Simulation (HPCS), 2016, pp. 443–448.

[3] M. Nowicki, D. Bzhalava, P. Bała, Massively Parallel Implementation of Sequence Alignment with Basic Local Alignment Search Tool Using Parallel Computing in Java Library, Journal of Computational Biology 25 (8) (2018) 871–881.

[4] M. Ryczkowska, M. Nowicki, P. Bała, Level-synchronous BFS algorithm implemented in Java using PCJ library, in: 2016 International Conference on Computational Science and Computational Intelligence (CSCI), IEEE, 2016, pp. 596–601.

[5] J. Mellor-Crummey, L. Adhianto, G. Jin, M. Krentel, K. Murthy, W. Scherer, C. Yang, Class II submission to the HPC Challenge award competition Coarray Fortran 2.0.

[6] D. Pasetto, A. Akhriev, A comparative study of parallel sort algorithms, in: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, ACM, 2011, pp. 203–204.