EVALUATION OF JXTA AND JAVA FOR DEVELOPING MESSAGE PASSING PARALLEL PROGRAMS

Márcio Augusto de Souza[1], Luciano José Senger[1], Arion de Campos Junior[1] Marcelo Ferrasa[1]

[1] Universidade Estadual de Ponta Grossa – UEPG

msouza@uepg.br, ljsenger@uepg.br, arion@uepg.br, mferrasa@uepg.br

Abstract: This paper presents a study of the use Java and JXTA for developing message passing parallel applications. This work is divided in two parts: a comparison of a parallel application developed in C (Open MPI) and Java (MPJ Express); a performance comparison among the message passing routines provided by JXTA, parallel communication libraries and the native sockets of Java and C. The results show that Java is a viable alternative to the development of parallel applications, both in terms of execution time and in terms of communication. However, JXTA has a very large communication overhead and it is not recommended in situations where much communication occurs.

Keywords: Parallel computing; Message passing; MPI; Java; JXTA; Evaluation

1. INTRODUCTION

Computer clusters can be used in a wide variety of applications in commercial and academics environments. Currently, an approach widely used is the use of clusters for executing parallel applications, enabling institutions that do not have a parallel computer to build and run applications that require great computational power (DANTAS, 2005).

Executing a parallel application involves dividing it into smaller parts that can be processed independently on different processors. When a parallel application is executed in a cluster, it is very common to use the message passing programming model, in which the different processes that belong to the application synchronize and communicate by exchanging messages (DONGARRA et al, 2003; FOSTER, 1995; FOX, 2003).

Message passing communication libraries are generally based on MPI (Message Passing Interface), which is a standard that was defined by a consortium of research institutes of universities and industries. MPI defines a set of routines that a message passing library should have, and some parallel programming software studied in this paper follow the MPI standard (MPI FORUM, 2012; QUINN, 2008).

Computer clusters can be organized in various ways, and one approach that has been used recently is peer-to-peer (P2P), which defines a system that consists of independent computers (peers) that share resources without the need for a central server. The P2P model provides an efficient use of resources, with great flexibility for growing in scale and self-organization (AMORETTI, 2006; MILOJICIC et al, 2002; VU et al, 2010).

The High Performance Computing research group of State University of Ponta Grossa proposed a framework called P2Pcomp, which allows the development and

execution of parallel programs on clusters (or grids) using the P2P model (SENGER et al, 2010).

This framework was developed with Java language and the JXTA platform, which provides an infrastructure for creating P2P systems. With P2Pcomp it is possible to develop parallel programs that use the message passing paradigm through a API that provides simple communication routines implemented over the routines of communication available in JXTA (HALEPOVIC e DETERS, 2005; VERSTRYNGE, 2010).

To analyze the performance of a parallel program that executes over a framework like P2PComp, it is necessary to understand the overhead caused by the use of Java and JXTA for programming parallel programs. Generally, parallel software libraries are implemented using C or Fortran, which are languages that does not depend on the execution of a virtual machine for interpreting programs (DEITEL e DEITEL, 2012; KAMINSKY, 2009). In addition, the performance of parallel programs depends on the performance of the communication routines, and it is important to analyze the adequacy of Java and JXTA for implementing communication routines.

In this context, this paper presents a performance analysis divided in two parts. Initially, it is studied the performance of Java for the implementation of sequential and parallel applications. Next, it is presented a comparative analysis of the performance of various communication routines implemented in C and Java, such as: TCP and UDP sockets; routines offered by parallel programming software organized according the MPI model (Open MPI and MPJ Express); routines offered by JXTA.

This article is organized as follows: section 2 provides a discussion of related work. Section 3 discusses the software and communication routines studied in this work. Section 4 presents the results of the performance evaluation of Java parallel program and communication routines, compared to their counterpart in C. Finally, in Section 5 the conclusions obtained in this work are presented.

2. RELATED WORK

The use of Java for parallel computing has been explored in several works that propose software that can run parallel applications and exchange messages with Java, such as (BAKER et al, 2006a; BORNEMANN et al, 2005; NIEUWPOORT et al, 2005). Besides that, efficient forms of communication between Java processes have been proposed in (AAMIR E JAWAD, 2009; BAKER et al, 2006b; GROPP e THAKUR, 2005; TABOADA et al, , 2012).

The use of P2P systems for executing parallel and distributed applications has been explored in (THERMING e BENGTSSON, 2005; VERBEKE et al, 2002). When JXTA is studied, there are woks that analyze the performance of JXTA as a whole, as (DAI et al, 2006; HALEPOVIC e DETERS, 2005), or specifically the performance of its communication routines, as (ANTONIU et al, 2005; ANTONIU et al, 2008).

This paper, unlike the related works cited in this section, presents a comparative evaluation of Java and C used for parallel programming, and a performance analysis of communication routines of JXTA when compared to widely used C and Java message passing parallel software. It is important to note that efficient communication is very important for the development of applications that exchange messages frequently.

The next section describes all software studied in this work and the communication routines offered by them.

3. MATERIALS AND METHODS

This section presents a discussion of the languages C and Java (subsection 3.1), from which all software analyzed in this work were derived. Subsection 3.2 discusses the message passing programming libraries used in the experiments, and subsection 3.3 presents the JXTA platform.

Java and C Languages

Java and C use two different models for making executable code: C compilers generate native machine code that is executed directly by the CPU and Java compilers generates bytecode that need to be interpreted by a virtual machine (called JVM – Java Virtual Machine).

Both C and Java provide sockets as the basic communication primitive. There are two types of sockets: a reliable and ordered communication based on connecting the two communication parts (TCP) and a connectionless package based communication with no guarantee of delivery (UDP). For the latter, there are a limit of 600 kilobytes per package (STEVENS e RAGO, 2005). Both forms of communication were analyzed in this paper.

Message Passing Libraries

The message-passing paradigm defines a model of parallel programming that provides high-level routines that allow parallel processes to communicate exchanging messages. There is a standard created by universities and industries, called MPI, which defines a set of routines that message-passing parallel programming libraries must offer.

MPI defines different variants of send and receive routines, but in this work it was only considered standard routines, which must be implemented in an efficient way according the MPI standard.

The MPI standard was initially created to be used with C and Fortran languages, but it was adapted for Java in a standard called MPJ (Message Passing Java). In this work, it was considered the following implementations of MPI and MPJ:

Open MPI: Open MPI is an open-source implementation of MPI that was developed by a consortium of companies, universities and research institutions. The Open MPI was based on three implementations of MPI: FT-MPI, LA-MPI and LAM (GRAHAM et al, 2005; HURSEY et al, 2009).

MPJ Express: The MPJ Express implements the MPI standard in Java and uses the Java NIO library as the communication basis. The Java NIO was created to provide high performance I/O routines (AAMIR e JAWAD, 2009; BAKER et al, 2006).

JXTA Programming

The P2P computing model defines the organization of distributed systems as a set of cooperating computers, called peers, which share their resources without requiring a central server. Files, disk storage and processor cycles are examples of resources that can be shared in a P2P system. There is an open-source specification, created by Sun Microsystems, which defines routines for creating, maintaining and exchanging messages in P2P networks, called JXTA.

From the JXTA specification it was defined the JXSE library, which is a Java implementation of the routines defined in JXTA. JXSE was used in the experiments performed in this work.

JXTA defines a basic type of communication called pipe, which is a channel that carry data flows. A pipe can optionally perform reliable communication, and this option can be configured at the time the pipe is created. Messages sent by a pipe can have a maximum size of 600 kilobytes.

When a message is received in a pipe, an event is generated, and the process being executed is interrupted so that the message can be received and processed. The process of event-driven message reception does not follow the model of point-to-point communication defined by sockets or by the messaging libraries defined earlier, in which there is a receiving routine that blocks the process until the message is received.

The JXTA socket, implemented over pipes, defines sending and receiving routines, and has no limitation to the size of a message. Both models of communication were analyzed in this work, using reliable and not reliable forms of communication (VERSTRYNGE, 2010).

4. RESULTS AND DISCUSSION

Analysis of Java and C parallelism

As it was explained in the previous section, a compiled C program is transformed into machine code and executed directly by the CPU. A Java program, on the other hand, is compiled to an intermediate code called bytecode, which is executed by a virtual machine. The JVM is important to ensure the portability of programs written in Java, but it can negatively affect the performance of a program (DEITEL e DEITEL, 2012; KAMINSKY, 2009).

To evaluate the performance difference between parallel applications developed in Java and C, it was developed a sequential and a parallel program for matrix multiplication using both languages. The parallel versions were developed with Open MPI 1.6 and MPJ Express 0.38

The tests were executed on two computers with AMD Athlon64 5200+ CPU with 2,7 gigahertz of clock and 2 gigabytes of memory. The software versions used are:

- Linux with kernel version 2.6.33;
- JVM 1.7;
- Compiler g++ 4.4.4.

Table 1 shows the results of the execution of the sequential versions in three scenarios:

- Compiled with g++ without optimization;
- Compiled with g++ using the compile option "O3", which generates optimized machine code;
- Compiled with Java

The results obtained in these tests show that, independently of the existence of a JVM, the runtimes of C and Java versions are very similar. In addition, if the standard C compilation is used, the Java program is 64% faster, which shows that Java compiler generate efficient bytecode.

Table 1. Runtime of the sequential matrix multiplication program using C and Java compilers

Compiler	Runtime in seconds
g++ standard compile	198,89 s
g++ optimized compile	129,09 s
Java	127,30 s

Tables 2 and 3 show the results obtained by the execution of the parallel versions of the matrix multiplication algorithm divided in two processes, using two computers or two cores of the same computer.

Table 2. Runtime of parallel implementation of matrix multiplication using Open MPI and MPJ Express on two computers

Software	Runtime in seconds
Open MPI	67,78 s
MPJ Express	68,09 s

Table 3. Runtime of parallel implementation of matrix multiplication using Open MPI and MPJ Express on two cores

Software	Runtime in seconds
Open MPI	64,52 s
MPJ Express	64,37 s

It can be noted that both versions offer similar runtimes. This application does not have a large communication overhead, which can be perceived by the small difference of runtimes obtained on cores and on different computers. These results show that Java is a good solution for the development of parallel applications, and the virtual machine does not affect the performance of a parallel application.

In the next subsection, it is presented an evaluation of the communication routines of Java, C and JXTA.

Communication routines performance evaluation

The performance evaluation of communication routines was performed by calculating the round-trip time of a message exchanged between two computers. This metric is measured by the execution of a benchmark called ping-pong, which calculates the time expended for a message to reach his destination and return to the original sender.

The benchmark transmits messages that carry byte data types and it was implemented in Java and C. The following sizes of message were considered:

- Small: 2, 200, 400, 600, 800 e 1,000 bytes
- Medium: 20,000, 40,000, 60,000, 80,000 e 100,000 bytes

- Large: 200,000, 400,000, 600,000, 800,000 e 1,000,000 bytes

The experiments were conducted on two computers with Intel Core i7 processor with 3.4 gigahertz of clock and 4 gigabytes of memory. The computers are connected to a gigabit Ethernet network. The software versions used are:

- Linux with kernel version 2.6.37;
- Compiler g++ 4.5.2;
- JVM 1.7;
- Open MPI 1.6;
- MPJ Express 0.38;
- JXSE 2.5 and 2.6.

All graphics shown in this subsection show in the x-axis the message size in bytes, and in the y-axis the communication time in milliseconds.

Figure 1. Transmission time for small messages, not including JXTA.

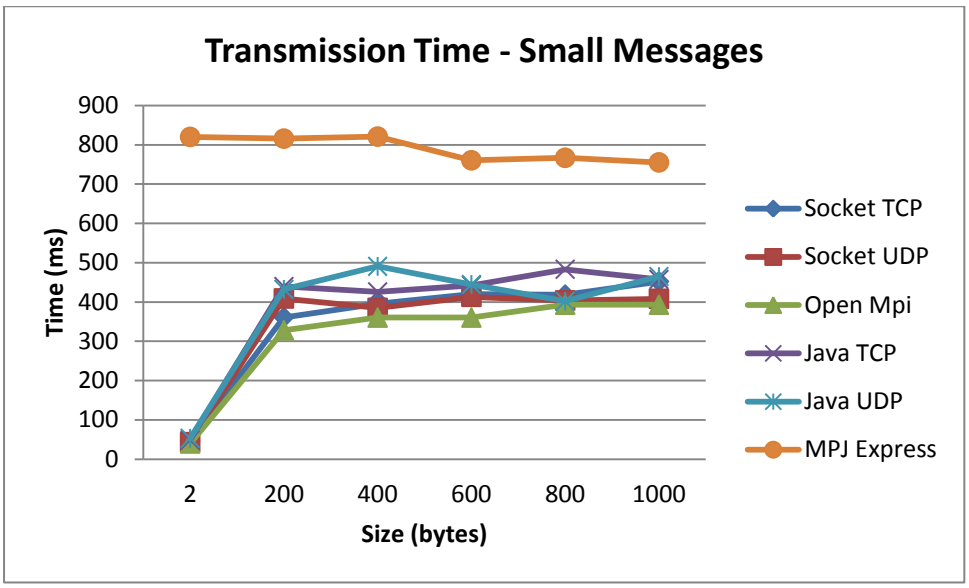
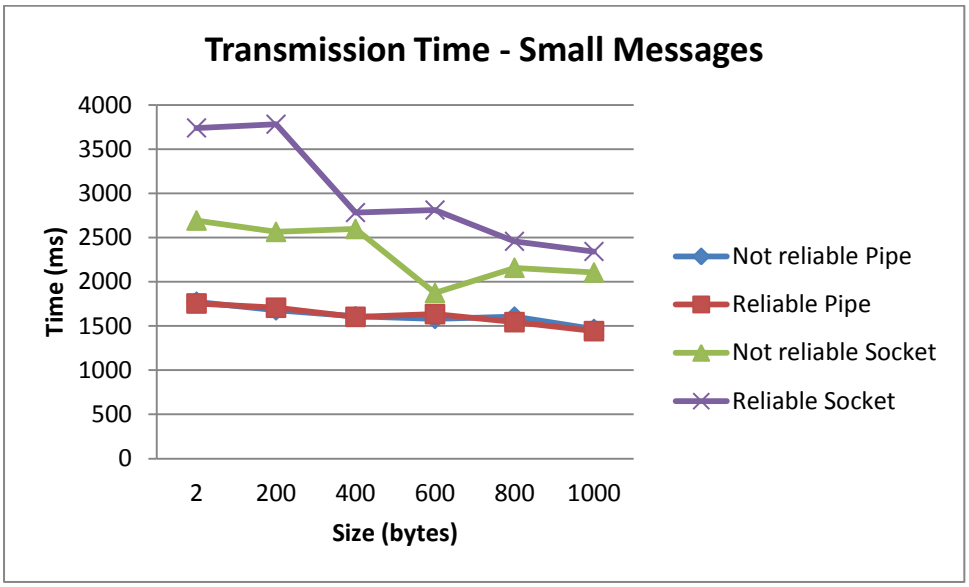


Figure 1 shows the communication time for small messages for all software analyzed, except for JXTA. The figure shows that, with minor variations, all communication routines have a similar performance. The exception is MPJ Express software, which has a considerably larger transmission time.

Figure 2. Transmission time for small messages of JXTA version 2.5, including reliable modes.



The evaluation of small JXTA messages was performed in three parts: the influence of using a reliable communication; the difference between versions 2.5 and 2.6; comparison of JXTA with all the other communication routines.

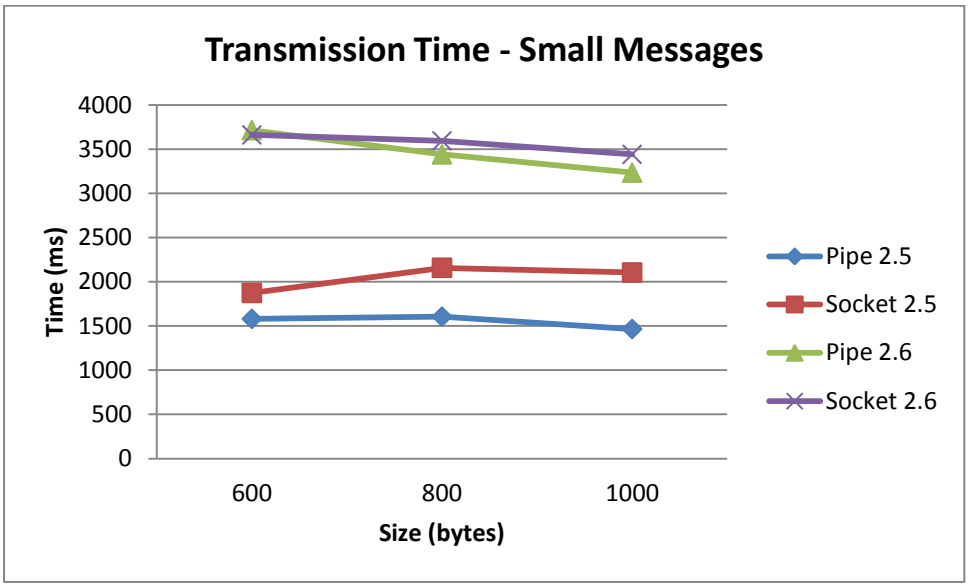
As discussed in subsection 3.3, JXTA presents reliable versions of communication routines, and figure 2 shows the performance of JXTA pipes and sockets of the two kinds.

Figure 2 shows that, for pipes, being reliable does not influence the performance. On the other hand, for a socket, there is a performance improvement when it is not used a reliable version. One characteristics of the two versions of JXTA analyzed is that the transmission time lowers as the size of the message increases, showing that it is preferable to use messages of larger sizes.

Figure 3 shows the communication time for unreliable communication routines for versions 2.5 and 2.6. The version 2.6 presented a great problem of variation in the measured communication times, and it was not possible to evaluate the performance of communication for message sizes lesser than 600 bytes. Figure 3 also shows that version 2.6 has poor performance compared to version 2.5.

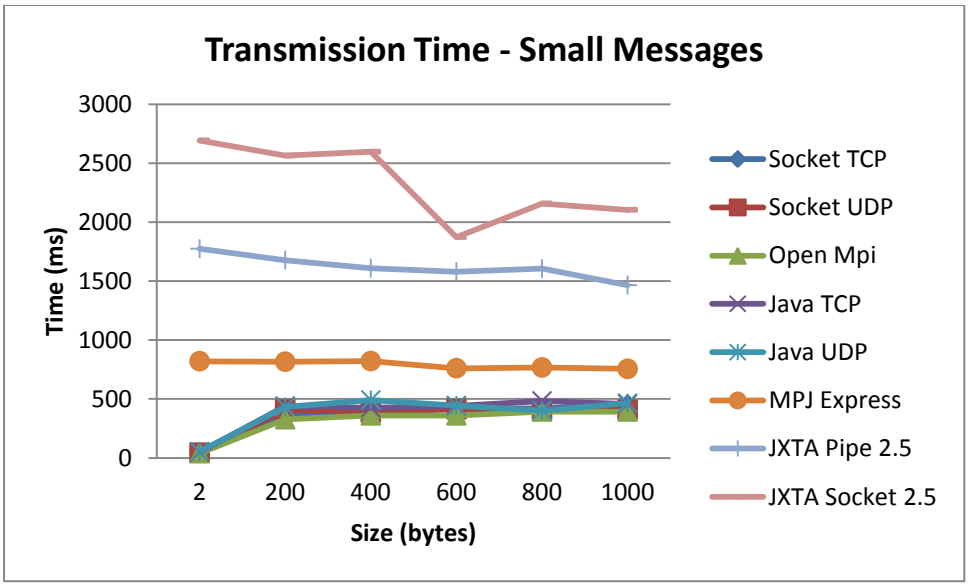
For JXTA, it was considered only unreliable routines of version 2.5 for the comparison with other communication routines, as they presented the best performance among all JXTA variants. Version 2.6 will no longer be addressed in this paper since the use of this version of JXTA is not recommended because of performance issues and high instability.

Figure 3. Transmission time for small messages of JXTA versions 2.5 and 2.6.



The Figure 4 shows a comparison among the transmission time of JXTA routines compared to communication routines available in C and Java, and it is possible to note that the performance of communication routines of JXTA is inferior to the others. For example, for a size of 200 bytes, a message sent using a Java socket takes 439 milliseconds, while a JXTA pipe transfers the same message in 1677 milliseconds.

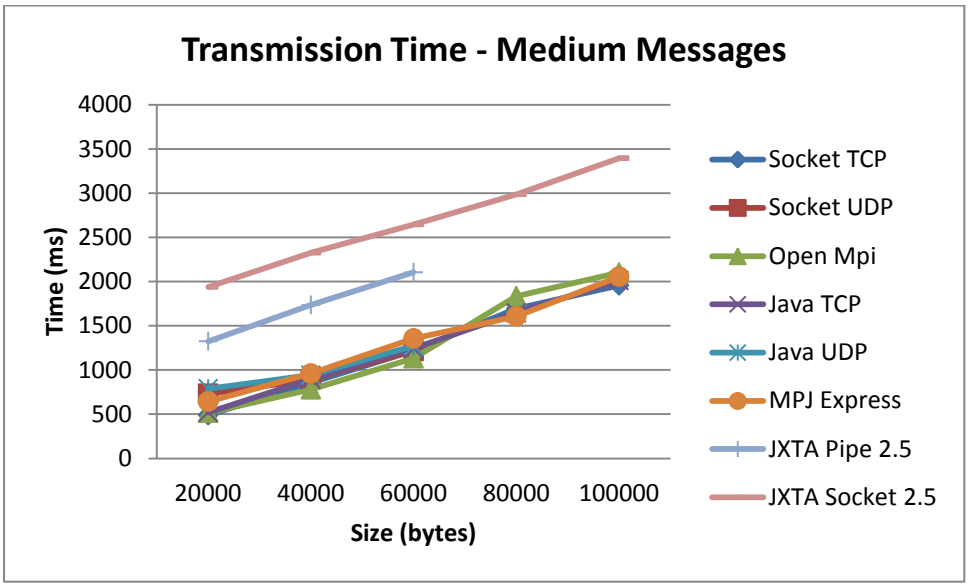
Figure 4. Transmission time for small messages of all communication routines.



This problem not occurs because of Java, as proved in Table 1, which shows that there is no considerable difference in performance between Java and C. The poor performance of the communication libraries of JXTA is derived from the structure of the software. This conclusion can also be reached for MPJ Express, which did not presented good performance.

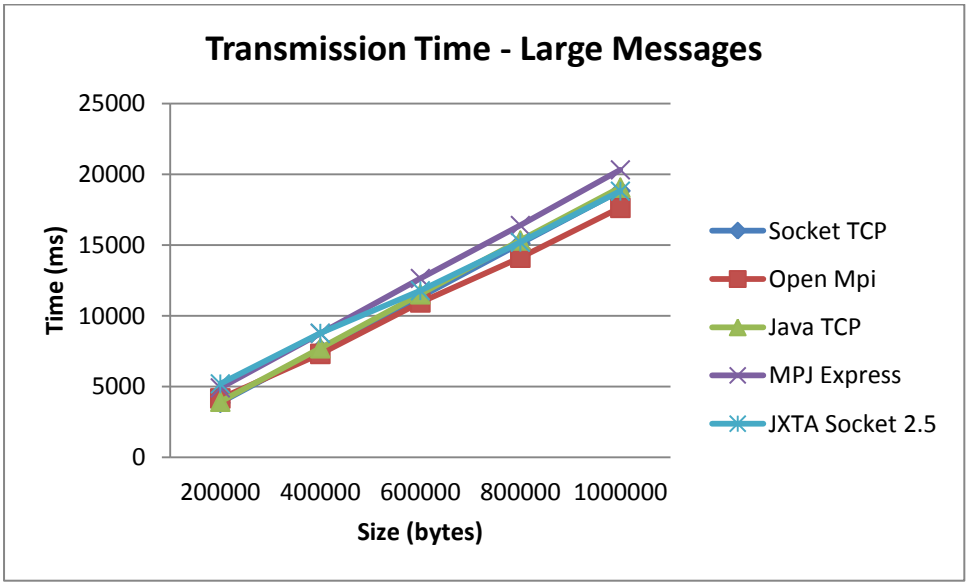
The Figure 5 shows performance values for medium-sized messages, and it can be noted that the performance difference between communication with JXTA and the other routines is lesser, but it is still significant. For a message of 600 kilobytes, the maximum size that can be transmitted in a JXTA pipe, there is a difference of 165% to a Java socket.

Figure 5. Transmission time for medium messages of all communication routines.



The communication routines have similar performance only when large messages, ranging from 200 megabytes to 1 gigabyte, are transmitted, as it can be seen in Figure 6.

Figure 6. Transmission time for large messages of all communication routines.



The results presented in this section show that JXTA have low performance communication routines, which in some cases can influence the overall performance of a parallel application.

6. CONCLUSION

The use of Java is a viable and attractive alternative for parallel computing, as the satisfactory results obtained from the experiments performed in this paper show. In addition, MPJ Express is a good option for implementing parallel Java applications, but in this particular case, it is recommended to use larger messages when the application is network-bound.

Regarding the performance of communication routines, the libraries studied in this work provided good performance, even when compared to sockets that are natively offered by programming languages. In this context, it was not noticed significant differences between Java and C for UDP and TCP, confirming that there is no difference between the two languages regarding communication performance.

JXTA 2.5, on the other hand, showed poor performance for exchanging small and medium messages. Version 2.6 was even worst in terms of performance and stability. Therefore, this work do not recommend the use of version 2.6 of JXTA.

The use of JXTA for developing parallel applications depend on the quantity of communication made by the application. However, it is important to remember that the use of P2P technology brings many advantages, so its cost / benefit must be always considered.

REFERENCES

AAMIR, S.; JAWAD, M. **Towards Efficient Shared Memory Communications in MPJ Express**. Java Workshop at the 23rd IEEE International Parallel and Distributed Processing Symposium, may, 2009.

AMORETTI, M. **Peer-to-peer based grid architectures.** PhD thesis, Universita Degli Studi Di Parma, january, 2006.

ANTONIU, G.; JAN, M.; NOBLET, D. **A practical example of convergence of P2P and grid computing: An evaluation of JXTA's communication performance on grid networking infrastructures.** IEEE International Symposium on Parallel and Distributed Processing, april, 2008.

ANTONIU, G.; HATCHER, P.; JAN, M.; NOBLET, D.A. **Performance evaluation of JXTA communication layers.** IEEE International Symposium on Cluster Computing and the Grid, may, 2005.

BAKER, M.; CARPENTER, B.; SHAFI, A. **MPJ Express: Towards Thread Safe Java HPC**. IEEE International Conference on Cluster Computing, september, 2006a.

BAKER, M.; CARPENTER, B.; SHAFI, A. **An Approach to Buffer Management in Java HPC Messaging.** Lecture Notes in Computer Science, vol. 3992, 2006b.

BORNEMANN, M.; NIEUWPOORT, R.; KIELMANN, T. **MPJ/Ibis: a flexible and efficient message passing platform for Java.** Proceedings of 12th European PVM/MPI Users' Group Meeting, p. 217-224, 2005.

DAI, Z.; FANG, Z.; HAN, X.; XU, F.; YANG, H. **Performance Evaluation of JXTA Based P2P Distributed Computing System.** 15th International Conference on Computing, november, 2006.

DANTAS, M. **Computação distribuída de alto desempenho: Redes, clusters e grids computacionais**. Axcel books, 2005.

DEITEL, H; DEITEL, P. **Java - How to Program.** 9th ed., Prentice Hall, 2012.

DONGARRA, J.; FOSTER, I.; FOX, G.; GROPP, W.; KENNEDY, K.; TORCZON, L.; WHITE, A**.** ***The Sourcebook of Parallel Computing***. Morgan Kaufmann Publishers, 2003.

FOSTER, I. **Designing and Building Parallel Programs: Concepts and tools for Parallel Software Engineering**. Addison Wesley Publishing Company, 1995.

FOX, J. **Messaging Systems: Parallel Computing in the Internet and the Grid.** Lecture Notes in Computer Science, vol. 2840, 2003.

GRAHAM, R.; WOODALL, T.; SQUYRES, J. **Open MPI: A Flexible High Performance MPI**. Proceedings of the 6th Annual International Conference on Parallel Processing and Applied Mathematics, 2005.

GROPP, W.; THAKUR, R. **An Evaluation of Implementation Options for MPI One-Sided Communication.** Proceedings of the 12th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2005), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol. 3666, september, 2005.

HALEPOVIC, E.; DETERS, R. **The JXTA performance model and evaluation.** Future Generation Computer Systems, vol. 21, issue 3, 2005.

HURSEY, J.; MATTOX, T.; LUMSDAINE, A. **Interconnect Agnostic Checkpoint/Restart in Open MPI.** Proceedings of the 18th ACM international symposium on High Performance Distributed Computing HPDC, p. 49-58. 2009.

KAMINSKY, A. **Building Parallel Programs: SMPs, Clusters & Java (Computing).** Course Technology – Cengage Learning, 2009.

MILOJICIC, D.; KALOGERAKI, V.; LUKOSE, R.; NAGARAJA, K.; PRUYNE, J.; RICHARD, B. **Peer-to-peer computing.** Technical Report HPL-2002-57, HP Laboratories, march, 2002.

MPI FORUM. **MPI: A Message-Passing Interface Standard: Version 3.0.** High Performance Computing Center Stuttgart, 2012.

NIEUWPOORT, R. V.; MAASSEN, J; WRZESINSKA, G.; HOFMAN, R.; JACOBS, C., KIELMANN, T.; BAL, H. **Ibis: a flexible and efficient Java based grid programming environment.** Concurrency and Computation: Practice and Experience, 17(7-8):1079-1107, june, 2005.

QUINN, M. **Parallel Programming in C with Mpi and Openmp**. McGraw-Hill Education, 2008.

SENGER, L.; SOUZA, M.; FOLTRAN JR, D. **Towards a peer-to-peer framework for parallel and distributed computing.** 22nd International Symposium on Computer Architecture and High Performance Computing, 2010.

STEVENS, R. W.; RAGO, S. A. **Advanced Programming in the UNIX® Environment.** 2nd ed, Addison Wesley, 2005.

TABOADA, G.; TOURIÑO, J.; DOALLO, R. **F-MPJ: scalable Java message-passing communications on parallel systems**. The Journal of Supercomputing, vol. 60, issue 1, april, 2012.

THERNING, N.; BENGTSSON, L. **Jalapeno: decentralized grid computing using peer-to-peer technology**. Proceedings of the 2nd conference on Computing frontiers, 2005.

VERBEKE, J.; NADGIR, N.; RUETSCH, G; SHARAPOV, I. **Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment.** Proceedings of the Third International Workshop on Grid Computing, 2002.

VERSTRYNGE, J. **Practical JXTA II: Cracking the P2P puzzle**. Lulu.com , 2010.

VU, Q.; LUPU, M.; OOI, B. **Peer-to-Peer Computing: Principles and Applications.** Springer, 2010.