Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Performance evaluation of virtual execution environments for intensive computing on usual representations of multidimensional arrays

Francisco Heron de Carvalho Junior *, Cenez Araújo Rezende

Mestrado e Doutorado em Ciência da Computação, Universidade Federal do Ceará, Brazil

ARTICLE INFO

Article history:
Received 19 April 2014
Received in revised form 16 March 2016
Accepted 22 April 2016
Available online xxxx

Keywords:
Virtual execution
High performance computing
Performance evaluation
Java virtual machine (JVM)
Common Language Infrastructure (CLI)

ABSTRACT

This paper evaluates the sequential performance of virtual execution environments (VEE) belonging to the CLI (Common Language Infrastructure) and JVM (Java Virtual Machine) standards, for the usual approaches of representing multidimensional arrays in programs that present intensive array access. Such programs are mostly found in scientific and engineering application domains. It shows that the performance profile of virtual execution is still highly influenced by the choice of both the multidimensional array representation and the VEE implementation, also contradicting some results of previous related works, as well as some beliefs of HPC programmers that come from their practice with native execution languages, such as C, C++, and Fortran. Finally, recommendations are provided both for HPC programmers that desire to take advantage of VEE performance for their array-intensive code and for VEE developers that are interested in improving the performance of virtual execution for the needs of HPC applications.

1. Introduction

Object-oriented programming languages based on virtual execution environments (VEE), such as Java and C#, are consolidated in the software industry, motivated by the security, safety and interoperability they provide to software products and the productivity provided by their supportive programming environments to software development and maintenance. These languages have caught the attention of Fortran and C users from computational sciences and engineering, interested in modern techniques and tools for improving the productivity of programming large-scale software in their particular domains of interest [1]. However, most of them still consider that the performance of VEEs does not attend to the needs of numeric- and array-intensive HPC (High Performance Computing) applications.

In the mid-1990s, the performance bottlenecks of JVM (Java Virtual Machine) for HPC have been widely investigated [2]. New techniques have been proposed to circumvent them, also implemented by VEEs that follows the CLI (Common Language Infrastructure) standard [3], such as .NET and Mono. In the mid-2000s, most of these efforts moved to the design of new high-level programming languages for HPC [4]. However, during this period, the performance of virtual machines was significantly improved with *just-in-time* (JIT) compilation. Also, CLI introduced new features for addressing HPC requirements, such as rectangular arrays, direct interface to native code, language interoperability and unsafe memory pointers.

* Corresponding author.
E-mail addresses: heron@lia.ufc.br (F.H. de Carvalho Junior), cenezaraujo@lia.ufc.br (C.A. Rezende).

http://dx.doi.org/10.1016/j.scico.2016.04.005
0167-6423/

However, after a decade of advances in the design and implementation of VEEs in the 2000s, there is still a lack of rigorous studies about their serial performance in the context of HPC applications [5]. Recent works have evaluated current JVM machines by taking parallel processing into account, including communication and synchronization costs [6,7], both in shared-memory and distributed-memory platforms, using a subset of NPB-JAV [8], a Java implementation of the NPB (NAS Parallel Benchmarks) suite [9]. Also, to our knowledge, W. Vogels reported the only comprehensive performance comparison between JVM and CLI machines published in the literature [10]. Finally, the existing works barely take into account the evaluation of different approaches of implementing multidimensional arrays in Java and C#, even though most of the computational-intensive algorithms implemented in scientific computing programs have multidimensional arrays as the main data structure. This fact becomes critical for performance, because the access time to values stored in multidimensional arrays may dominate the total execution time of these programs. The sources of overhead in intensive access to multidimensional arrays depend on the way they are represented, including poor cache performance, pointer indirections, index calculation arithmetic, array-bounds-checking (ABC), etc.

This paper evaluates the performance of a set of numeric programs that perform intensive access to the elements of multidimensional arrays, with the following goals in mind:

- to compare the current performance of the virtual execution engines that follow the CLI and JVM standards for different approaches for implementing multidimensional arrays;
- to quantify the current performance overhead of virtual execution, compared to native one; and
- to identify bottlenecks in the most popular current implementations of virtual machines, regarding the support of multidimensional arrays.

As far we know, this paper is the first one to report a comprehensive study about the performance of distinct approaches for implementing multidimensional arrays in the most popular programming languages based on virtual execution, which are:

- ***embedded arrays***, where the list of indexes for accessing a value in a multidimensional array are mapped to an index of a unidimensional array;
- ***jagged arrays***, also known as *arrays-of-arrays*, where an array of n dimensions, where $n > 1$, is implemented as an unidimensional array of n references to arrays of $n - 1$ dimensions;
- ***rectangular arrays***, where array elements are stored in contiguous memory addresses, such as in Fortran and C.

The results of this paper show that some results reported in previous related works cannot be generalized, neither for all VEEs belonging the same standard (e.g. JVM or CLI) nor for all numeric- and array-intensive programs. Therefore, they complement the results of these related works [6,11,10,8], finding that they lack the analysis of different forms of implementing multidimensional arrays. Also, they show that some common-sense beliefs of HPC programmers are not valid depending on the choice of VEE, and present useful recommendations for helping them on how to optimize the performance of array-intensive code according to the choice of VEE and multidimensional array representation. Finally, VEE developers could find, in this paper, evidences of performance bottlenecks that may help them to improve the performance of VEE implementations. It is not one of the objectives of this work to identify the source of each observed performance bottleneck. To that end, it is necessary a deeper systematic study, including the design and use of specific-purpose microbenchmarks for testing a potentially large number of hypothesis, which we plan as a further work. The data collected in the experiments the results of which are reported in this paper is available at `http://npb-for-hpe.googlecode.com`.

In what follows, Section 2 introduces virtual execution technology and motivates the research work reported in this paper. Section 3 presents the methodology adopted in the experiments. Section 4 summarizes the data collected in the experiments and discusses their meaning. Finally, Section 5 concludes the paper, by emphasizing its contributions and suggesting ideas for further works.

2. Context and related works

Programming languages based on virtual execution environments (VEE) abstract away from the hardware and operating systems of computer systems. In heterogeneous environments, the benefits of virtual execution, such as security and cross-platform portability, are clear, in particular for applications that are distributed across a network, such as the internet.

Java was launched by Sun Microsystem in the 1990s. The Java Virtual Machine (JVM) has been implemented for most of the existing computer and operating system platforms. Its intermediate representation, called *bytecode*, is dynamically compiled through just-in-time (JIT) compilation. The JIT compiler dictates the performance of virtual execution engines such as JVM. The most important industrial-strength implementations of JVM are now supported by Oracle [12] and IBM [13]. Oracle has two JVM implementations, called JRockit and HotSpot, whereas J9 is the name of the IBM's JVM implementation.

JRockit was firstly introduced by Appeal Virtual Machines. It was further acquired by BEA Systems, in 2002, and later on by Oracle, in 2008. Since 2011, JRockit was made free and publicly available by Oracle. In turn, HotSpot virtual machines are owned by Oracle since the acquisition of Sun Microsystem in 2010. Oracle maintains both a commercial and an open-source

Please cite this article in press as: F.H. de Carvalho Junior, C.A. Rezende, Performance evaluation of virtual execution environments for intensive computing on usual representations of multidimensional arrays, Sci. Comput. Program. (2016), http://dx.doi.org/10.1016/j.scico.2016.04.005