DrHJ — the cure to your Multicore Programming Woes

(Demonstration Proposal)

Vincent Cavé Vivek Sarkar Jarred Payne Raghavan Raman Mathias Ricken Corky Cartwright

Department of Computer Science, Rice University

{vcave, vsarkar, jrp1, raghav, mgricken, cork}@rice.edu

Abstract

DrHJ extends DrJava with support for the pedagogic Habanero-Java language derived from X10, and used to teach parallel programming at the sophomore level. The demonstration will show how a rich and powerful set of parallel programming capabilities can be easily introduced to anyone familiar with the basics of sequential programming in Java.

1. Presenters

Vincent Cavé has been a Research Programmer at Rice University since 2008, and the development team lead for the Habanero-Java implementation [5, 8]. He completed a Research Masters degree in concurrent and distributed systems in 2006 from the University of Nice-Sophia Antipolis / INRIA. While visiting IBM as a research intern in 2006, he contributed to an early implementation of the X10 runtime system based on the `java.util.concurrent` library release in Java 1.5 [3]. While at INRIA, Vincent worked on the open source implementation of the Proactive grid middleware project. In addition to leading the development of the HJ system, Vincent's responsibilities at Rice include leading the build, test and release of other compilers and runtimes in the Habanero Multicore Software Research Project. Vincent is well qualified to present this demonstration, since he has a deep knowledge of the internals of the DrHJ IDE and the HJ compiler and runtime system.

Vivek Sarkar conducts research in multiple aspects of parallel software including programming languages, program analysis, compiler optimizations and runtimes for parallel and high performance computer systems. Prior to joining Rice in July 2007, Vivek was Senior Manager of Programming Technologies at IBM Research. His responsibilities at IBM included leading IBM's research efforts in programming model, tools, and productivity in the PERCS project during 2002- 2007 as part of the DARPA High Productivity Computing System program. His past projects include the X10 programming language, the Jikes Research Virtual Machine, the ASTI optimizer used in IBM's XL Fortran product compilers, the PTRAN automatic parallelization system, and profile-directed partitioning and scheduling of Sisal programs. Vivek became a member of the IBM Academy of Technology in 1995, the E.D. Butcher Professor in Engineering at Rice University in 2007, and was inducted as an ACM Fellow in 2008. He holds a B.Tech. degree from the Indian Institute of Technology, Kanpur, an M.S. degree from University of Wisconsin-Madison, and a Ph.D. from Stanford University. Vivek has given several tutorials on a wide range of topics related to programming models and compilers at past conferences including PLDI 1993, POPL 1996, ASPLOS 1996, PLDI 2000, Java Grande 2001, OOPSLA 2003, ECOOP 2004, OOPSLA 2006, PPoPP 2007, PLDI 2007, PLDI 2008, PLDI 2009 and PLDI 2011. Vivek is well qualified to present this demonstration, since he has used HJ and DrHJ extensively when teaching the sophomore-level class on "Fundamentals of Parallel Programming" that he created at Rice [1]. He also has a deep knowledge of the design and implementation of the HJ language as well as its roots in X10 and other parallel programming models.

OOPSLA'11 Date, Location

2. Background

DrHJ extends DrJava with support for the pedagogic Habanero Java (HJ) parallel programming language that was derived from the earlier Java-based definition of the X10 language [6]. DrHJ builds on past experiences at Rice with developing the DrJava IDE [2] and the HJ language. DrJava is a free, open-source lightweight IDE for Java. It is designed primarily for students, providing an intuitive interface and the ability to interactively evaluate Java code in an *Interactions Pane*. It also includes powerful features for more advanced users, enabling (for example) the DrJava team to develop DrJava completely within DrJava. Since the inception of the DrJava project in 2002, it has been downloaded over 1.1 million times and is being used by many universities world-wide. DrJava has also been used as a teaching tool in books published by Pearson Education and Wiley Higher Education.

Figure 1 shows the general architecture of DrHJ, which can be divided into two main parts, the graphical user interface (GUI) and a collection of compiler plug-ins. It is composed of three elements: a Navigation Pane that shows documents currently opened; a Definitions Pane that contains the source code being edited; and a set of bottom panes that includes the Interactions Pane and panes for compiler messages and program output. The Definitions Pane allows users to edit HJ source code and provides syntax highlighting for HJ keywords. Compilation and execution of HJ programs can be done directly in the IDE.

The Habanero-Java (HJ) language [10] was developed at Rice University during 2007-2010 as a pedagogic extension to the original Java-based definition of the X10 language [6]. In addition to its use as a research language in the Rice Habanero Multicore Software research project [9], HJ is used in a new sophomore-level course on "Fundamentals of Parallel Programming" (COMP 322) which has become a required course for all Computer Science majors at Rice. The HJ extensions to Java are primarily focused on task parallelism — task creation (`async`, `future`, `forall`) [8], termination (`finish`, `get`), mutual exclu-

sion (`isolated`), phasers [12][1], and hierarchical places. Similar extensions to C and Scala are being pursued in the Habanero C and Habanero Scala projects at Rice. HJ has also been used as a foundation to implement higher-level programming models such as Intel's Concurrent Collections [4].

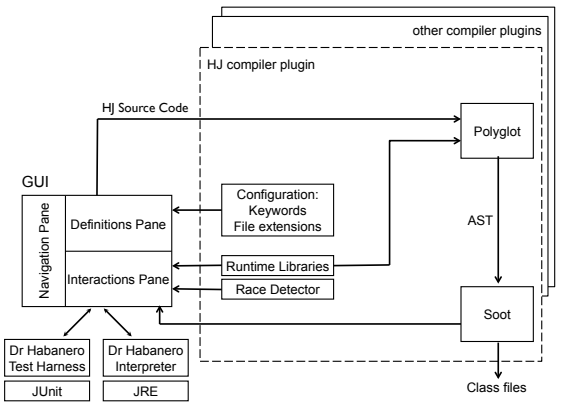

Figure 1. Architecture of the DrHJ IDE

3. Description of Demonstration

This demonstration shows how a rich and powerful set of parallel programming capabilities can be easily introduced to anyone familiar with the basics of sequential programming in Java using the HJ language and the DrHJ IDE. The demo will start with a publicly available download of a single jar file for DrHJ that requires no installation, assuming that a Java Runtime Environment (JRE) is available. We will step through a selected number of laboratory assignments from the COMP 322 class at Rice. For each example, we will show how a sequential Java program can be easily converted to a parallel program. The HJ compiler is integrated with the DrHJ IDE using a standard plug-in interface. Compiler error messages are transferred back to the IDE and displayed in one of the bottom panes. If compilation is successful, the user can invoke the program by pressing the Run toolbar button, or by using the `run` keyword in the Interactions Pane, followed by the name of the program's main class, and an optional list of arguments for the program (for example: `run Fib 10`).

DrHJ also includes a tool to detect data races in HJ programs, based on the ESP-bags algorithm developed for HJ [11]. The ESP-bags algorithm is a generalization of the SP-bags algorithm developed for Cilk's spawn and sync constructs [7]. Like SP-bags, ESP-bags works by following a depth-first execution of a sequentialized version of the parallel program. The extensions in ESP-bags were necessary because the set of computation graphs generated by async-finish constructs in HJ is more general than the graphs generated by spawn-sync constructs in Cilk. The DrHJ data race detector currently supports HJ programs that contain only `finish`, `async` and `isolated` constructs, since those programs can be easily sequentialized. An important property of the DrHJ data race detector is that it uses the depth-first execution to report all *potential* races that may be encountered for a given input.

While this demonstration will show how multicore programming can be performed simply using the HJ language and the DrHJ IDE, a number of technical challenges had to be overcome in building this system. First, the compiler and the runtime system are all implemented in Java to ensure portability. In fact, HJ's parallel runtime system leverages a number of low-level capabilities in the `java.util.concurrent` library. Second, we have to distinguish between the main JVM that executes the DrHJ IDE, and the the "Interpreter JVM" that executes HJ applications. DrHJ's main JVM communicates with the Interpreter JVM using Java's Remote Method Invocation (RMI) API. Any output produced by the Interpreter JVM is forwarded back to DrHJ to be displayed in the Interactions Pane. Third, DrHJ includes a state-of-the-art data race detector that tracks locations in Java objects and arrays. Finally, all the parallel primitives have highly efficient and scalable implementations on multicore processors.

[1] The latest release of `j.u.c` in Java 7 includes Phaser synchronizer objects, which are derived in part from the phaser construct in HJ.

References

[1] Comp 322: Fundamentals of parallel programming. URL `https://wiki.rice.edu/confluence/display/PARPROG/COMP322`.

[2] Eric Allen, Robert Cartwright, and Brian Stoler. Drjava: a lightweight pedagogic environment for java. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, SIGCSE '02, pages 137–141, New York, NY, USA, 2002. ACM. ISBN 1-58113-473-8. doi: http://doi.acm.org/10.1145/563340.563395. URL `http://doi.acm.org/10.1145/563340.563395`.

[3] Rajkishore Barik, Vincent Cave, Christopher Donawa, Allan Kielstra, Igor Peshansky, and Vivek Sarkar. Experiences with an SMP Implementation for X10 based on the Java Concurrency Utilities. In *Workshop on Programming Models for Ubiquitous Parallelism (PMUP), held in conjunction with PACT 2006, Sep 2006*, 2006.

[4] Zoran Budimlić, Michael Burke, Vincent Cavé, Kathleen Knobe, Geoff Lowney, Ryan Newton, Jens Palsberg, David Peixotto, Vivek Sarkar, Frank Schlimbach, and Sağnak Taşırlar. Concurrent Collections. *SIAM PP10, Special Issue on Scientific Programming*, 2010.

[5] Vincent Cavé, Zoran Budimlić, and Vivek Sarkar. Comparing the usability of library vs. language approaches to task parallelism. In *Evaluation and Usability of Programming Languages and Tools*, PLATEAU '10, pages 9:1–9:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0547-1. URL `http://doi.acm.org/10.1145/1937117.1937126`.

[6] P. Charles et al. X10: an object-oriented approach to non-uniform cluster computing. In *OOPSLA'05*, pages 519–538, New York, NY, USA, 2005. ISBN 1-59593-031-0. doi: http://doi.acm.org/10.1145/1094811.1094852.

[7] Mingdong Feng and Charles E. Leiserson. Efficient detection of determinacy races in cilk programs. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 1–11. ACM, 1997. ISBN 0-89791-890-8. doi: http://doi.acm.org/10.1145/258492.258493.

[8] Yi Guo, Jisheng Zhao, Vincent Cavé, and Vivek Sarkar. Slaw: a scalable locality-aware adaptive work-stealing scheduler. In *IPDPS '10: Proceedings of the 2010 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–12, Washington, DC, USA, Apr 2010. IEEE Computer Society.

[9] Habanero. Habanero multicore software research project web page. http://habanero.rice.edu, January 2008.

[10] Habanero. Habanero Java. http://habanero.rice.edu/hj, Dec 2009.

[11] Raghavan Raman, Jisheng Zhao, Vivek Sarkar, Martin Vechev, and Eran Yahav. Efficient data race detection for async-finish parallelism. In *RV'10, Proceedings of the 1th International Conference on Runtime Verification*. Springer, Nov 2010.

[12] J. Shirako et al. Phasers: a unified deadlock-free construct for collective and point-to-point synchronization. In *ICS '08*, pages 277–288, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-158-3. doi: http://doi.acm.org/10.1145/1375527.1375568.

Figure 2. DrHJ GUI screenshot featuring the HJ data race detector. This screenshot shows a simple program *ArraySum.hj*, that attempts to use two tasks to sum the elements of an array. The child `async` task in lines 50–54 computes the sum of elements $X[0 \ldots mid]$ in $X[0]$. The parent task computes the sum of elements $X[mid \ldots n-1]$ in $X[mid]$ in parallel with the child task, and then adds $X[0]$ and $X[mid]$ in line 60 after the `finish` statement which ensures the completion of the child task. However, this code contains an error because $X[mid]$ is read by the child task, and is also read and written by the parent task. This error is reported as a data race by DrHJ, as shown in the Interactions Pane in figure 2. The error report includes the source coordinates for the two conflicting accesses as well as the index of the array location on which the race occurs.