DrHabanero — a Platform for Parallel Software Education in Java

Robert Cartwright, Vivek Sarkar
Department of Computer Science, Rice University
{*cork, vsarkar*}*@rice.edu*

Functional Decomposition as a Foundation for Teaching Parallelism The standard mechanism for incorporating parallelism in mainstream languages like Java, C#, and C++ is task-level or thread-level parallelism operating on shared data structures. In parallel applications based on thread-level parallelism, the semantics of execution is combinatorially more complex than in sequential applications, because the ordering of operations across threads on shared data can materially affect the results of computations. Program behavior is no longer deterministic, making program testing and debugging very difficult. Multi-threaded programs written in mainstream languages rely on explicit locking and thread confinement to constrain non-determinism, but these mechanisms are extremely difficult to get right and mistakes are often not revealed by conventional testing.

Some experts in high performance parallel computing have suggested that the complexities of parallel programming in mainstream languages like Java and C# cannot be tamed unless their semantics are deterministic by default [3]. We largely agree. *But for most programmers—notably undergraduate students—we don't believe that determinism alone is sufficient.* Under proper restrictions, parallel program decomposition is astoundingly simple; in fact, it is no more difficult than the composition of operations in algebra. Functional programmers discovered this property decades ago by observing that in a functional program (where all variables are immutable after being bound), the evaluation of the argument expressions in a function application are independent computations that can be performed in parallel. Early functional languages like MultiLisp[7] codified this idea using the concept of a *future*, an expression whose computation has been scheduled but whose value may not yet have been determined. The concept of futures is highly compatible with task-parallel programming models like Intel Threading Building Blocks [10], Cilk [2], X10 [4] and Habanero Java (HJ) [6]. In discussing revisions to the Rice core undergraduate curriculum, the authors discovered that the decomposition of Java programs into components with purely functional interfaces was common both to our preferred Java programming pedagogy and to our preferred approach to introducing parallel programming to undergraduate students.

In the context of sequential computation, a functional programming discipline imposes significant extra overhead on many computations because data structures must be copied instead of mutated. As a result, imperative solutions are often much more efficient. In the context of many-core parallel computing, however, the trade-offs can be very different. Distributing shared data to other processors requires copying that data to memory local to the target processors. In well-designed parallel programs, the overheads involved in supporting a functional model computation can often be folded into the communication costs inherent in parallel processing.

In introductory computing curricula that teach functional programming early, parallel computing can be folded into the curriculum as a strategy for making functional programs more efficient. In fact, students can be taught an execution model where additional independent tasks can be very inexpensive because they often run on otherwise idle processing cores. In the context of functional programming, the parallel evaluation of independent tasks such as arguments in a function call is a natural intuition. As experienced software developers accustomed to uniprocessors, we have learned to suppress this intuition because it does not apply to uniprocessors (or even to n-core machines for small n).

DrHabanero: On the Importance of IDEs in Teaching Parallelism Good programming pedagogy requires appropriate tools to support and enforce the abstractions that we want students to master. To support our functional approach to parallel programming, we are building a new software platform called DrHabanero to support parallel programming education in Java—leveraging our past experiences with

pedagogic IDEs for Java (DrJava [1, 5]) and extensions of Java that support parallel functional programming (HJ [6]). DrJava is a pedagogic IDE for Java developed at Rice under the direction of Prof. Cartwright that includes explicit support for sequential functional programming in Java. DrJava is used by many universities worldwide, and recently passed the milestone of one million downloads since its inception in 2002.

DrHabanero is being implemented as an extension of the DrJava code base creating a user-friendly IDE for Habanero Java that provides explicit support for a functional subset combining the functional core of Habanero Java with the functional language level of DrJava [8]. The DrJava functional language level involves no new constructs; it simply restricts programmers to defining Java classes where fields and variables are immutable and automatically generates the constructors, accessors, structural **`equals`**, and **`toString`** methods for these classes. As a result, a program defining functional lists (using the composite pattern) including an insertion sort method can be written in less than 20 lines of code.

In DrHabanero, we believe that students will be able to develop efficient parallel programs with a simple underlying functional semantics. To simplify decomposing programs into parallel tasks (subcomputations), the data interfaces between tasks in DrHabanero programs must be strictly functional (immutable). In purely functional programs, each task must be functional as well. In more general DrHabanero programs–written by advanced programmers–tasks may be locally imperative but the interfaces must still be strictly functional. The semantic integrity and correctness of compiled Habanero Java hinge on maintaining this constraint. The DrHabanero environment will ensure that all programs conform to the specified mutability constraints.

This merger entails a complete re-implementation of Habanero Java as a preprocessor that transforms Habanero Java source code to conventional Java code augmented by the Habanero run-time library. HJ is currently implemented as a stand-alone compiler built on top of the Polyglot compiler framework which only supports Java 1.4 functionality. As result, Habanero Java currently does not include any of the language extensions added in Java 5.0 and Java 6.0. In DrHabanero, Habanero Java translation will be implemented by a preprocessor patterned after the current DrJava functional subset translator.

We are also re-implementing full Habanero Java (with imperative features) as a source-to-source translator, sharing much of the DrHabanero code base. When the full language is used, static [11] and dynamic [9] debugging tools for checking determinism will also be available to help students quickly identify the source of their errors. In addition, research is underway to help establish when certain imperative patterns (such as SPMD execution with barriers and reductions) are essentially functional in their behavior.

References

[1] E. Allen, R. Cartwright, B. Stoler. DrJava: A lightweight pedagogic environment for Java. *SIGCSE 2002*, pp. 137-141.

[2] R. Blumofe *et al*. Cilk: An efficient multithreaded runtime system. *PPoPP 1995*, pp. 207–216.

[3] R. Bocchino, Jr. *et al*. A type and effect system for deterministic parallel java. *OOPSLA 2009*, pp. 97-116.

[4] P. Charles *et al*. X10: an object-oriented approach to non-uniform cluster computing. *OOPSLA 2005 Onward!*, pp. 519–538.

[5] DrJava Project. **`http://drjava.org`**, 2002.

[6] Habanero Java (HJ) Project. **`http://habanero.rice.edu/hj`**, 2009.

[7] R. Halstead, Jr. Multilisp: a language for concurrent symbolic computation. *ACM TOPLAS*, 7(4):501–538, 1985.

[8] J. Hsia, E. Simpson, D. Smith, R. Cartwright. Taming Java for the classroom. *SIGCSE 2005*, pp. 327–331.

[9] R. Raman *et al*. Efficient data race detection for async-finish parallelism. *RV 2010*.

[10] Thread Building Blocks Project. **`http://www.threadingbuildingblocks.org`**.

[11] Martin Vechev *et al*. Verifying determinism of structured parallel programs. *SAS 2010*.