HJ-Viz: A New Tool for Visualizing, Debugging and Optimizing Parallel Programs

Peter Elmers
Rice University
pe4@rice.edu

Hongyu Li
Rice University
hl33@rice.edu

Shams Imam
Rice University
shams@rice.edu

Vivek Sarkar
Rice University
vsarkar@rice.edu

1. Motivation

The proliferation of multicore processors warrants parallelism as the future of computing, increasing the demand to write parallel programs for increased application performance. Previous experience has shown that writing explicitly parallel programs is inherently more difficult than writing sequential programs. Programmers need parallel programming models, constructs, and tools that can simplify writing of parallel programs. In this poster, we present an innovative new tool, HJ-Viz, which generates interactive Computation Graphs (CGs) of parallel programs by analyzing event logs. The visual feedback is valuable for a programmer to efficiently optimize program logic and to eliminate the presence of potential bugs which may otherwise be difficult to detect. For example, in cases of deadlocks, HJ-Viz enables users to visualize and easily diagnose the deadlock scenario.

CGs provide an intuitive graphical view of a parallel program's execution. A CG is an acyclic graph that consists of: *a*) a set of nodes, where each node represents a step consisting of sequential computation, and *b*) a set of directed edges that represent ordering constraints among steps. A task can be partitioned into multiple steps, the key constraint is that a step should not contain any parallelism or synchronization.

We incorporate Abstract Execution Metrics (AEM) as well as Real Time Metrics (RTM) in the visualization. AEM describe the performance of a program by measuring the cost of abstract operations, such as floating-point, comparison, stencil, or data structure operations. RTM inserts timing calls to record the time elapsed between consecutive synchronization events.

Programmers can use the visualization of the CG by HJ-Viz to pinpoint potential sources of bugs and points of improvement for parallel performance. HJ-Viz highlights the program's critical paths and displays the amount of work performed in each step of computation based on the collected AEM or RTM. Our event logging infrastructure also maintains precise source code locations for each event, allowing HJ-Viz to display the code involved in the creation of every node in the CG.

SPLASH '14, Oct 20-24 2014, Portland, OR, USA.

http://dx.doi.org/10.1145/2660252.2660395

2. Implementation

Our current implementation of HJ-Viz processes event logs produced by the HJlib runtime [5]. HJlib is an implementation of a pedagogic parallel programming model used at Rice University to teach a sophomore-level course titled "Fundamentals of Parallel Programming" [1]. Built on top of the Java Concurrency library [2], HJlib facilitates an effective learning process in topics like parallel patterns, thread safety and data race avoidance using a wide range of parallel constructs including async tasks, isolated, futures, data-driven tasks, phasers and actors [3]. HJ-Viz can be used to visualize HJlib programs written using **any** combinations of these constructs.

Usage of parallel constructs trigger events in the HJlib runtime which are used by the event logger to create the related entries in the event log. AEM data is also included when available. The event log is built incrementally and the complete event log becomes available upon program termination. This log is then processed offline by HJ-Viz to generate a dot representation of the CG. The dot file is then laid out and converted to a scalable vector graphic by Graphviz [4], and displayed in the user's browser with interactivity features implemented in JavaScript. Using web browsers as the renderer ensures cross-platform compatibility. As a result, HJ-Viz can be hosted on users' machines or on a central server over the Internet.

3. Visualization Examples

One of the goals of HJlib is to introduce students to the fundamentals of parallel programming. By providing visualizations for parallel programs, HJ-Viz makes it easier for students who have no prior experience in parallel programming to grasp the fundamental ideas in this field. Figure 1 shows a simple program written in HJlib that a student may come

across early on. This program deadlocks while using DDFs since the tasks at lines 4 and 5 are waiting on dependencies (B and A respectively) that are never satisfied unless their corresponding bodies are run. Having a visual representation of simple programs like these, as well as more complicated programs later on, helps in a student's understanding of the fundamental parallel constructs being employed. Figure 2 displays the CG rendered by HJ-Viz when this program is run. The two nodes with red borders highlight deadlocked tasks (as they are floating leaf nodes with missing join edges) which allows the user to obtain a better understanding of where the deadlock stems from. As seen in the figure, hovering over nodes displays the relevant source code snippet participating in the deadlock.

```
1 finish(() -> {
2   HjDataDrivenFuture<Long> A = newDDF();
3   HjDataDrivenFuture<Long> B = newDDF();
4   asyncAwait(B, () -> A.put(B.get() + 3));
5   asyncAwait(A, () -> B.put(A.get() + 5));
6 });
```

Figure 1: Deadlock with DDFs.

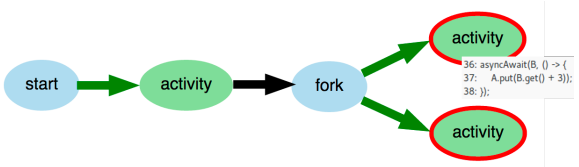

Figure 2: DDF deadlock CG.

Figure 3 shows a parallel MergeSort program. The CG in Figure 4 shows that under `finish` scope, there are two new activities being spawned by the main program. The two newly forked tasks work on different portions of the data with the same instructions. The two `finish` start nodes on the third level arise from the recursive call to MergeSort(), and in this case, each asynchronous call splits the list into two smaller parts, until the list is of unit length.

```
1 final int mid = M + (N - M) / 2;
2 finish(() -> {
3   async(() -> mergesort(A, M, mid));
4   async(() -> mergesort(A, mid + 1, N));
5 });
6 merge(A, M, mid, N);
```

Figure 3: MergeSort (merge code snippet omitted for brevity).

The CG can also be used to generate the parallelism profile of the program in terms of AEM or RTM. Depending on the number of participating nodes at each abstract work unit or real time interval in the CG, the chart shows how the degree of parallelism varies over time. Traversal of Figure 4 generates the bar chart shown in Figure 5. Users can write different variants of the same program (as shown using matrix multiplication in our accompanying poster) and

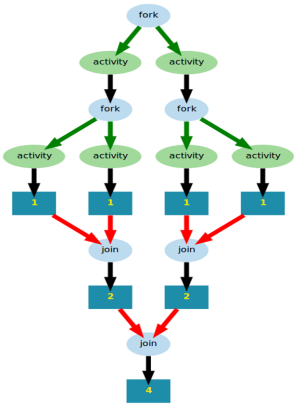

Figure 4: MergeSort CG on input array of size 4. The bold edges represent the critical path of the computation. Since all the edges are in bold, it means the computation is evenly load balanced.

generate the CGs and parallelism profiles for different parallel algorithms for the same problem. These visualizations are particularly useful when comparing the performance of the different algorithms and choosing the best performing parallel implementation.

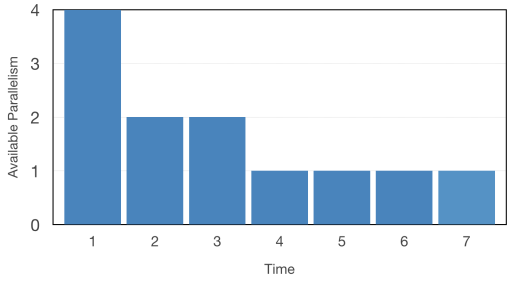

Figure 5: MergeSort available parallelism chart.

In summary, HJ-Viz renders the CG of a parallel program, providing an intuitive graphical view of the program's execution. The visual feedback allows a user to reason about performance problems and to optimize program logic to maximize available parallelism. The visualization also enables fixing bugs such as deadlocks which may otherwise be difficult to diagnose. As a result, HJ-Viz will be used at Rice University to teach a sophomore-level course in parallel programming. We plan to support visualizing nodes participating in data race bugs in HJ-Viz.

References

[1] COMP 322: Fundamentals of Parallel Programming. https://wiki.rice.edu/confluence/display/PARPROG/COMP322, 2014.

[2] Java Concurrency Utilities. http://docs.oracle.com/javase/8/docs/technotes/guides/concurrency/, 2014.

[3] V. Cavé, J. Zhao, Y. Guo, and V. Sarkar. Habanero-Java: the New Adventures of Old X10. In *PPPJ'11*, pages 51–61, 2011.

[4] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11):1203–1233, 2000.

[5] S. Imam and V. Sarkar. Habanero-Java Library: a Java 8 Framework for Multicore Programming. In *PPPJ'14*. 2014.