Paul Holmes, pwh2125

GPU Computing, COMS W4995

Term Project Report

**Jiminy – An Extension to the Geppetto Programming Language**

I’m very interested in artificial intelligence, so any time I have a project to do for some class, I always try to do something in the AI domain. For this class, I chose to do an extension to a language I wrote for my Programming Languages and Translators class last year. The language is called Geppetto, named for the puppeteer from the movie *Pinocchio*. The purpose of Geppetto is to be “a language for the modeling of intelligent agents” – to bring a lifeless program to life, so to speak. At least, that’s the theory. That claim turned out to be a *little* grandiose considering what the language ultimately ended up being able to do, but I like to think that the potential is still there.

In a very condensed nutshell, the premise of the language is basically to model a system, then “wind it up, let it go, and see what happens”. More specifically, Geppetto allows the programmer to define objects called Entities, which have Attributes and Properties, and the Rules that operate on them. Each time unit (called a Cycle), the language evaluates all the Rules, potentially changing the states of the Entities, until some predefined end state is reached or a certain number of Cycles have been executed.

The problem – or one of them, anyway – was that the evaluation of the Rules each Cycle could become extremely time-consuming. While a Rule *may* refer to one specific Entity, that undercuts the purpose of the language. The real power of Geppetto is unleashed when Rules are generic and can thus refer to a whole class of Entities. So for example, in a system with 20 Entities, a Rule that refers to three Entities might have to be evaluated times, once for each possible combination of Entities. It’s clear that this could quickly become computationally infeasible. Ultimately I ended up putting restrictions on the language that made it possible to simulate only the most trivial of systems.

Jiminy (named for the helpful cricket in *Pinocchio*) is intended to help alleviate that problem. The idea behind Jiminy is that it can use the GPU to perform the Rule calculations in parallel, thus greatly decreasing processing time.

Each Rule in Geppetto is a logical expression – that is, a statement that evaluates to either true or false – coupled with a statement to execute if the expression evaluates to true. As originally formulated, the logical expressions in a Rule are completely arbitrary – any statement that can be formed with the syntax of the language is legal provided it evaluates to a boolean. But the evaluation of a totally arbitrary boolean expression would be inefficient on a GPU because the threads would be too divergent. In addition, it would make the scope of problem we’re trying to solve perhaps a little too large for this project. Therefore I decided to limit the kinds of expressions we’re trying to evaluate to a just particular kind of boolean expression: decision lists.

For the uninitiated, a decision list is an ordered sequence of nodes, each of which has a condition which, if true, selects a true or false value for the whole expression. In other words, it’s basically a sequence of if-then-else statements, with the “then” clause always being either “value = true” or “value = false”. It’s a classic data type often used in CS theory classes to demonstrate limits on computational complexity. Most logical expressions can be expressed as a decision list, so it doesn’t limit the problem significantly. In theory, we could use any arbitrary logical expression in the expression in each node of the decision list, but to keep things simple I used randomly-generated “if variable == value” expressions, with the understanding that we could use any arbitrary logical expressions in their place if we so chose.

Another simplification I added to the project is that I didn’t try to integrate Jiminy directly with the Geppetto language. That would have required a significant rewrite of Geppetto’s Rule-processing engine, which I decided was beyond the scope of this project.

I did, however, decide to write Jiminy in Java, which will make it much easier to integrate with Geppetto if and when I decide to do so. Since I’m using NVIDIS’s CUDA API to interface with the GPU, it also means I had to use CUDA’s Java bindings, which actually turned out to be not so bad compared to the other third-party libraries I’ve used in the past. It’s clearly just a thin wrapper around the C language CUDA libraries (as evidenced by its rather awkward “Pointer” classes), but it is at least exceptionally well-documented and, within its limits, seems to be fairly robust and well-coded.