1 Bio

I am a PhD-student in the Computer Science Department at UC Berkeley, advised by Professor Edward A. Lee. I study models of computation, programming languages, and systems design. I have a BS in Computer Science and an MS in Grid Computing, both from University of Amsterdam. My current research is focused on preserving correctness in the face of concurrency, not so much exploiting parallelism for performance gain. I predominantly take this course as an opportunity to brush up on my parallel programming skills.

2 Application: Parallel SAT Solver

SAT, or the Boolean Satisfiability problem, entails finding an assignment of truth values to the terms in a Boolean formula such that the entire formula yields true. If such an assignment can be found, the formula is said to be “satisfiable,” otherwise, it is “unsatisfiable.” This problem is known to be NP-complete. In fact, SAT was the first-known NP-complete problem [1].

SAT problems appear in a broad variety of applications such as model checking, hardware verification, automated planning and scheduling, computational biology, and many others. In spite of the discouraging computational complexity of SAT, a lot of progress has been made toward efficient SAT solvers. Most current-day SAT solvers are sequential, based on a back-tracking graph search algorithm commonly known as Davis-Putnam-Logemann-Loveland (DPLL) [5], and usually combined with some heuristics and optimization techniques.

SAT solvers could well benefit from parallelization. There are two general approaches: 1) divide-and-conquer, and 2) a portfolio-based approach. The former attempts to distribute parts of the search problem among concurrent workers (e.g., [4]), whereas the latter runs differently tweaked solvers in parallel (e.g., [2]). Both approaches face significant challenges [3], making efficient parallelization of SAT a particularly interesting open research problem.
References


