Book Review: Automatic Re-Engineering of Software Using Genetic Programming

Author: Conor Ryan

Parallel processing is becoming cheaper and more accessible due to advances in technology and availability of cheap parallel architecture machines such as Beowulf systems. Software needs to be written (or re-written) in such a way that it can take advantage of these parallel architectures. Auto-parallelization is the name given to the systematic rewriting of sequential programs so that they can be executed on a parallel architecture.

Conor Ryan’s book, is an extension of his PhD thesis, addressing the application of GP techniques to fully automated parallelization of sequential imperative programs. The book starts by describing genetic programming, covering the techniques and approaches used in the later parts of the book, which describe the application of GP to auto-parallelization. It is largely self-contained and could be read and appreciated by researchers and practitioners in either genetic programming and/or software engineering.

Traditional auto-parallelization approaches involve data dependence analysis. This analysis seeks to find statements and constructs which cannot affect one-another. Such statements are written sequentially in the program code only because the programming language notation enforces such sequencing. There is no reason why such statements should not be executed in parallel. The goal of auto-parallelization is to identify as much of the program as possible which can be executed in parallel, thereby speeding up execution. The statement constructs which can have most effect on the speed-up are clearly loops, as their bodies are typically executed many times. Auto-parallelization work therefore tends to focus upon transformations which can be applied to loops.

Perhaps the most interesting aspect of the book is its comparison of two radically different approaches to auto-parallelization, both grounded in GP. The first approach, Paragen I, treats the program itself as the individual to be optimised, whereas the second approach, Paragen II, treats the sequence of transformations to be applied to the program as the individual to be optimised. Thus, Paragen II uses a form of ‘meta-level genetic programming,’ in which the original program is parallelized by a sequence of transformations, which are themselves the subject of GP. The transformation sequence can be thought of as a very simple program in a programming language, in which primitive instructions are transformation steps and sequencing of instructions is the only statement construct. The GP operators are applied, not to the original program, but to a meta-program (the transformation sequence). Transformation tactics are often defined using looping and conditionals. It is not hard to imagine, how the ‘transformation language’ could be extended.
to include them. Their addition would make the transformation language Turing complete, thereby making Ryan's approach fully a meta-GP approach.

By contrast, the Paragen I approach, uses the genetic operators directly on the program to be optimised. These include traditional GP mutation and cross over operators and therefore, the resulting program is not guaranteed to behave identically to the original. This means that the fitness function needs to consider two objectives; that of speed-up and that of correctness. Correctness is assessed using an automated testing-based approach, in which the original and transformed program are compared with respect to a sequence of test cases. As Ryan indicates, this approach to correctness is inherently unsafe, as no amount of testing can guarantee correctness. Certainly, much work on transformation has taken correctness preservation as fundamental and essential to the successful application of transformation. However, recent approaches, such as Ernst's work on Assertion Generation [1] have considered unsafe approaches which have nonetheless been shown to be useful.

Paragen I and II represent two approaches to auto-parallelization using evolutionary techniques. There is a third possibility, considered in a PhD thesis by Kenneth Williams [2], but not addressed by Ryan's book. Williams simply used mutation and selection to optimise programs for parallel execution. For Williams, the transformations applicable are the mutation operators, and fitness is defined purely in terms of the speed of execution within the parallel architecture. In this approach, crossover is not used, as this was not definable in a meaning-preserving manner. This approach is more like hill climbing, as no cross-over operation is employed.

Williams compared this 'hill climbing' approach to one in which the transformation sequence is the individual to be optimised and for which cross over is possible in addition to mutation. He found that the mutation and selection approach outperformed the transformation-sequence based approach. Work on auto-parallelization by Ryan and Williams's appear to have taken place concurrently, and so neither author has been able to compare their work with that of the other. As the transformation-sequence based approach is essentially the approach adopted by Ryan in Paragen II, a comparison of Ryan's work with Williams's work would be extremely interesting.

Ryan's book considers auto-parallelization, but it has to be said that software re-engineering is a far broader field. The approaches considered could equally well be applied to a number of problems typically associated with re- and reverse engineering, for example slicing,1 maintenance. However, these topics are not considered in the book and so the title is a little more general than the content.

Ryan provides a good starting point for research into the wider application of GP techniques to automated re-engineering. Re-engineering is a fundamental part of software engineering, which cannot be avoided as successful implementations will always be required to evolve and adapt in ways unanticipated by the original designers. Re-engineering is also a very expensive activity as problems like the year 2000 problem and euro conversion have shown, so automation or semi-automation are extremely attractive.

For future work on the application of GP to automated transformation, it is likely that the Paragen II approach will be the most suitable, though Williams's use of mutation-only may also bear fruit in some application areas. Both these