Compilation of Parallel Applications via Automated Transformation of BMF Programs (Summary of Thesis)

Brad Alexander

January 12, 2006

Transformation is crucial to any program improvement process. Highly transformable notations pave the way for the application of deep and pervasive program improvement techniques. Functional programming languages are more amenable to transformation than their more traditional imperative counterparts. Moreover, functional programs specify only true dependencies between values, making improvements that reveal and exploit parallelism much easier. Some functional programming notations are more transformable than others. Bird-Meertens-Formalism (BMF) is a functional notation that evolved as a medium for transformational program development. A substantial, and growing, body of work has created novel tools and techniques for the development of both sequential and parallel applications in BMF.

Formal program development is at its most useful when it can be carried out automatically. Point-Free BMF, where programs are expressed purely as functions glued together with higher-order operators, provides enhanced scope for automated development because many useful transformations can be expressed as easily-applied rewrite rules. In addition, BMF contains primitives with highly parallel implementations and, with its direct linking of producers and consumers of data, exposes the communication, implicit in these links, to further processing. Realistic sequential and parallel static cost models can be attached to BMF code so the relative merits of applying various transformations can be accurately assessed.

In spite of its potential merits there has been little work that has utilised point-free BMF, in a pervasive manner, as a medium for automated program improvement. This thesis describes a prototype implementation that maps a simple point-wise functional language into point-free BMF which is then optimised and parallelised by the automated application of, mostly simple, rewrite rules in a fine-grained and systematic manner. The implementation is shown to be successful in improving the efficiency of BMF code and extracting speedup in a parallel context. The report provides details of the techniques applied to the problem and shows, by experiment and analysis, how reductions in high data-transport costs are achieved. We also describe techniques used to keep the optimisation task tractable by alleviating the hazard of case-explosion.

The report is structured according to the stages of the compilation process, with related work described at the end of each chapter. We conclude with our main finding, namely, the demonstrated feasibility and effectiveness of optimisation and parallelisation of BMF programs via the automated application of transformation rules. We also restate techniques useful in achieving this end, the most important of which is the substantial use of normalisation during the optimisation process to prepare code for the application of desirable transformations. We also present a brief summary of potential future work including the introduction of more formally described interfaces to some of the transformative rule-sets, the automatic production of annotated proofs and a facility to display static estimates of the efficiency code during transformation.