Atlantique Research Overview

Robert A. Paige
Computer Science Department
New York University/Courant Institute
251 Mercer St.
New York, NY 10012, USA
paige@cs.nyu.edu *

The main objective of our research is to improve software development technology for non-numerical algorithm implementation. In particular, a major goal is to achieve at least a five-fold improvement in productivity without sacrificing any performance when compared with conventional manual programming in a low level algebraic language like C. The proposed software development methodology makes use of generic fully automatic program transformations that capture algorithm design principles. These transformations are tested on paper by using them for algorithm explanations and inventions. The methodology is implemented in the APTS transformational programming system [34]. Feasibility of the approach is tested by conducting experiments with APTS to compare productivity of our methodology with conventional handicraft programming.

1 Transformational Methodology

Our transformational methodology is based on three main generic (with respect to language paradigm and computational model) transformations. The first transformation simulates associative access on a sequential RAM in real-time [31,8,12]. The result is a way to implement primitive set theoretic operations efficiently. The second transformation develops a syntactic characterization of amortized complexity to detect, maintain, and exploit program invariants [33,7]. One major consequence is a way to implement efficient searching for subsets that satisfy a property. The final transformation makes use of a dominated convergence argument to compute least and greatest fixed point operations efficiently [10]. All three of these transformations are combined to form functional specification languages (making use of fixed point operations) that can be compiled into programs with worst case asymptotic time and space bounded by a specific function of the input and output space [32,7,14].

The problems of efficient implementation of associative access, of indexes to support searching a set for a property, and of fixed points have concerned the programming language and database communities for over twenty years (e.g., included within the programming language community are Jay Earley at UC Berkeley [19,20,21], Jim Morris at Los Alamos [28,42], Low and Rovner at the U. of Rochester [27,35], Barstow [4] and Kant [26] at Stanford, Fong and Ullman at Princeton [22], Jack Schwartz at NYU [37,38,30,36,17], and Cousot and Cousot [16]; included within the database community are Naqui and Henschen [29,24], Aho and Ullman [1], Selinger [39], Banchilhon [2], Ramakrishnan [3], and Willard [43,44]).

*This research is partially supported by Office of Naval Research Grant N00014-93-1-0924, Air Force Office of Scientific Research Grant AFOSR-91-0308, and National Science Foundation Grant MIP-9300210.

One key contribution of our work has been to determine the level of speedup achieved by application of our transformations. One key open problem is to reformulate our transformations for parallel models of computation.

2 Algorithm Design

The main point of our algorithm design work is to test the effectiveness of our transformational methodology before launching into an implementation. A major benefit of algorithm design by transformation is that it produces an actual program along with an algorithm. Consequently, we can conduct experimental benchmarks to obtain computational results that complement the theoretical improvements. This gives our work a potential for pragmatic as well as theoretical impact. Our transformational methodology has been used to invent new algorithms for propositional Horn Clause Satisfiability on a sequential pointer machine [32], transformation from regular expressions to DFA's [15], and ready simulation [6]. We are currently working on transformational explanations and proofs of parallel algorithms.

3 Transformational System

A meta-transformational environment called APTS was built in order to implement the preceding transformations [34]. APTS has meta-tools for syntax, semantics, conditional rewriting, and built-in finite differencing (an abstract mechanization of top down stepwise refinement [18]) supporting a single integrated transformational language called RSL. This language has been designed to implement very complex transformations together with the program analysis needed to justify when such transformations can be applied. Compiler construction is a major application.

Compilers written in RSL make use of the following APTS components. The RULE DATABASE (RDB) contains inference rules specifying program properties, e.g. type or dataflow. Inference rules are specified in a language similar to Datalog [41] augmented with function symbols and primitive pattern matching predicates. An inference engine analyzes the program (being compiled) for properties specified in the RDB, and it outputs the PROGRAM DATABASE (PDB) of finite relations storing the program's properties. The TRANSFORMATION DATA BASE (TDB) contains conditional meaning preserving program transformations of two kinds - rewriting or finite differencing. A transformation engine selects a transformation T by matching T with a portion of the program and by ensuring that the applicability condition for T, when instantiated with the current PDB, is satisfied. It then applies transformation T to the program to obtained a new transformed program. Consequently, the PDB must be updated to be consistent with the new program and the RDB.

The inference and transformation engines make use of our highly efficient bottom up pattern matching algorithm [13]. The inference engine used to calculate RDB relations [11] combines our fast bottom up pattern matching with RETE style pattern matching [23] and seminaive evaluation of Datalog [5,2,41].

The most pressing and perhaps most difficult open problem in APTS system research is in devising a scheme to maintain the consistency of semantic relations incrementally in the presence of conditional rewriting. Current methods work for extensional relations such as *monotone* and *type*. Automatic methods fail for intensional relations such as *free* and *bound* variables, *control flow*, and most subtype relations. Enhancing the system with meta-level support (to prove transformations correct) is another major goal. Finally, partial evaluation [25] will be investigated as a way to improve APTS performance by removing interpretive overhead.

4 Productivity Experiments

We have used APTS to build two compilers for conducting experiments to test the feasibility of our transformational methodology. One of these compilers translates a major fragment of SETL2 [40] into C. The other translates SQ2+ [10] (a strongly typed functional subset of SETL2 augmented with fixed point expressions) into C. Our SQ2+ and SETL2-to-C compilers are written in about 10,000 lines of RSL. Perhaps the most interesting aspect of these compilers is the subtype inference method used to implement data structure selection based on the real-time simulation of an abstract set machine on a RAM [12]. The subtype inference algorithm was shown to be based on a logic in which all type variables represent unique finite minimal domains that could be computed efficiently. These domains are the kernels of data structure aggregates supporting the real-time simulation of set theoretic primitive operations.

The preceding translators were used in productivity studies, and the results were highly encouraging and surprising [9]. These results suggest that our approach can achieve at least a five-fold improvement in programming productivity of nonnumerical algorithm implementation (compared with conventional handicraft programming) without sacrificing more than 50% in run-time performance. The experiments tested productivity in C for sequential machines, and the largest C program was 12 pages long. Although our approach conserves manual programming labor, it falls short of a practical technology, because the automated software production costs are too high. That is, the rate at which user-supplied mathematical specifications are automatically translated into efficient C code is too slow - about 3.3 lines of SETL2 translated into 24 lines of C per minute.

5 Future Plans

The key open problem for the future is to build a new practical software development technology in which large-scale reliable and efficient software can be produced with (1) translation rates two orders of magnitude better (than the current 24 lines of C per minute), (2) a convincing five to fifteen-fold improvement in productivity over the best hand-crafted programming, (3) no sacrifice in performance over the best hand-crafted code, and (4) concurrent as well as sequential target architectures. A grand challenge would be to use this technology to translate all of APTS into high performance C.

References

[1] A. Aho and J. Ullman. Universality Of Data Retrieval Languages. In *Proc. ACM 6th POPL*, pages 110-117, Jan, 1979.

[2] F. Bancilhon. Naive Evaluation of Recursively defined Relations. *On Knowledge-Base Management Systems*, N, Brodie and H, Mylopoulos, editors, pages 165-178, 1986.

[3] F. Bancilhon, and R. Ramakrishnan. An amateur's introduction to recursive query-processing strategies. In *ACM SIGMOD*, pages 16-52, 1986.

[4] D. Barstow. Knowledge-Based Program Construction, Elsevier North-Holland, 1979.

[5] R. Bayer. Query evaluation and recursion in deductive database system. unpublished manuscript, 1985.

[6] B. Bloom and R. Paige. Computing Ready Simulations Efficiently. In *Proc. First North American Process Algebra Workshop*, A. Zwarico and S. Purushothaman, editors, Workshops in Computer Science Series, Springer-Verlag, pages 119 - 134, 1992.

[7] J. Cai and R. Paige. Binding Performance at Language Design Time. In *Proc. ACM POPL*, pages 85 - 97, Jan, 1987.

[8] J. Cai and R. Paige. Look Ma, No Hashing, And No Arrays Neither. In *Proc. ACM POPL*, pages 143 - 154, Jan, 1991.

[9] J. Cai and R. Paige. Towards Increased Productivity of Algorithm Implementation. In *Proc. ACM SIGSOFT*, pages 71 - 78, Dec. 1993.

[10] J. Cai and R. Paige. Program Derivation by Fixed Point Computation. *Science of Computer Programming*, 11:197-261, 1988/89.

[11] J. Cai. A Language for Semantic Analysis. Technical Report 635, Courant Institute/New York University, 1993.

[12] J. Cai, P. Facon, F. Henglein, R. Paige, and E. Schonberg. Type Transformation and Data Structure Choice. In *Constructing Programs From Specifications*, B. Moeller editor, North-Holland, pages 126 - 124, 1991.

[13] J. Cai, R. Paige, and R. Tarjan. More Efficient Bottom-Up Multi-Pattern Matching In Trees. *Theoretical Computer Science*, 106:(1):21 - 60, Nov. 1992.

[14] J. Cai and R. Paige. Languages Polynomial in the Input Plus Output. In *Algebraic Methodology and Software Technology*, M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, Workshops in Computing Series, Springer-Verlag, pages 287 - 302, 1992.

[15] C.-H. Chang and R. Paige. From Regular Expressions to DFA's Using Compressed NFA's. In *Proc. CPM '92*, A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, Springer-Verlag, Lecture Notes in Computer Science number 644, pages 88 - 108, 1992.

[16] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific J. Math.*, 82:(1):43-57, 1979.

[17] R. Dewar, A. Grand, S. C. Liu, J. Schwartz, and E. Schonberg. Program by Refinement as Exemplified by the SETL Representation Sublanguage. *TOPLAS*, 1:(1):27-49, July, 1979.

[18] E. W. Dijkstra. A Discipline of Programming, Prentice-Hall, 1976.

[19] J. Earley. Toward an Understanding of Data Structures. *CACM*, 14:(10):617-627, Oct. 1971.

[20] J. Earley. High Level Operations in Automatic Programming. In *Proc. Symp. on Very High Level Langs.*, Sigplan Notices, 9:(4), Apr, 1974.

[21] J. Earley. High Level Iterators and a Method for Automatically Designing Data Structure Representation. *J of Computer Languages*, 1:(4):321-342, 1976.

[22] A. Fong and J. Ullman. Induction Variables in Very High Level Languages. In *Proc. Third ACM POPL*, pages 104-112, Jan, 1976.

[23] C. Forgy. RETE, a fast algorithm for the many patterns many objects Match problem. *Aritifical Intelligence*, 19:17-37, 1982.

[24] L. Henschen and S. Naqvi. On Compiling Queries in Recursive First-order Database. *CACM*, 31:47-85, 1984.

[25] N. Jones, C. Gomard, and P. Sestoft. Partial Evaluation and Automatic Program Generation, Prentice Hall, 1993.

[26] E. Kant. Efficiency in Program Synthesis, UMI Research Press, 1981.

[27] J. Low and P. Rovner. Techniques for the automatic selection of data structures. In *Proc Third POPL*, pages 58 - 67, 1976.

[28] J. Morris and M. Wells. The Specification of program flow in Madcap VI. In *Proc. ACM Natl. Conf.*, pages 755 - 762, 1972.

[29] S. Naqvi and L. Henschen. Synthesizing Least Fixed Point Queries into Non-recursive Iterative Programs. In *Proc. IJCAI 83*, pages 25-28, Aug, 1983.

[30] R. Paige and J. Schwartz. Expression Continuity and the Formal Differentiation of Algorithms. In *Proc. 4th ACM POPL*, pages 58-71, Jan, 1977.

[31] R. Paige. Real-time Simulation of a Set Machine on a RAM. In *Proc. ICCI '89*, Computing and Information, Vol II, R. Janicki and W. Koczkodaj, editors, pages 69-73, 1989.

[32] R. Paige and F. Henglein. Mechanical Translation of Set Theoretic Problem Specifications Into Efficient RAM Code - A Case Study. *Journal of Symbolic Computation*, 4:(2):207-232, Aug., 1987.

[33] R. Paige. Programming With Invariants. *IEEE Software*, 3:(1):56-69, Jan, 1986.

[34] R. Paige. APTS Reference Manual. unpublished manuscript, 1993.

[35] P. Rovner. Automatic Representation Selection for Associative Data Structures. Technical Report 10, Dept. of Computer Science, University of Rochester, 1977.

[36] E. Schonberg, J. Schwartz, and M. Sharir. An Automatic Technique for Selection of Data Representations in in SETL Programs. *ACM TOPLAS*, 3:(2):126-143, Apr, 1981.

[37] J. Schwartz. Automatic Data Structure Choice in a Language of Very High Level. *CACM*, 18:(12):722-728, Dec, 1975.

[38] J. Schwartz. Optimization of Very High Level Languages, Parts I, II. *J. of Computer Languages*, 1:(2,3):161-218, 1975.

[39] P. Selinger, et.al. Access path selection in a relational database management system. In *Proc. ACM SIGMOD*, pages 23-34, May, 1979.

[40] K. Snyder. The SETL2 Programming Language. Technical Report 490, Courant Institute/ New York University, 1990.

[41] J. Ullman. Principles of Database and Knowledge-Base Systems, I, Computer Science Press, 1988.

[42] M. Wells and J. Morris. The unified data structure capability in Madcap VI. *Intl. Journal of Comp. and Info. Sci.*, 1:(3):193 - 208, Sept. 1972.

[43] D. Willard. Abstract Predicate Retrieval Theory. Technical Report 83-3, State University of New York at Albany, Aug, 1983.

[44] D. Willard. Quasilinear Algorithms for Processing Relational Calculus Expressions. In *Proc. ACM PODS*, pages 243-257, 1990.