BROWN UNIVERSITY

Final Technical Report
ONR N00014-83-K-0146
DARPA Orders 4786, 6320

Department

of

Computer Science
Final Technical Report
ONR N00014-83-K-0146
DARPA Orders 4786, 6320
Final Report on Multiparadigm Design Environments
ONR Contract N00014-83-K-0146 and
DARPA Order Nos. 4786 and 6320

Co-Principal Investigators:
Eugene Charniak
Thomas L. Dean
Thomas W. Doeppner
Paris Kanellakis
Daniel Lopresti
Steven P. Reiss
John E. Savage
Andries van Dam
Jeffrey S. Vitter
Peter Wegner
F. Kenneth Zadeck
Stanley Zdonik

Abstract

The goal of the research under the “Multiparadigm Design Environments” project was to develop prototype environments to support the design of complex software and VLSI systems. Research on this project has produced the following results:

1. New methods for programming in terms of conceptual models
2. Design of object-oriented languages
3. Compiler optimization and analysis techniques for high-level languages, including object-oriented languages
4. Design of an object-oriented database, including development of query languages and optimization methods
5. Development of operating system support for parallel programming
6. Algorithm development for I/O efficiency and incremental computation
7. Determining the computational complexity of ML, type inference
8. A new architecture for programmable systolic arrays
9. New parallel algorithms for the graph partitioning problem and proof that key heuristics for it, including simulated annealing, are P-complete

We experimented with object-based methods for programming directly in terms of conceptual models, object-oriented language design, computer program optimization, and object-oriented database construction. We also examined the expressive power and optimization of database/programming languages, and the expression and exploitation of parallelism. The theoretical and experimental fruits of this research are being widely distributed and used, and we expect these results to have a strong influence on future design environments.
1 Conceptual Programming

Conceptual programming was studied using the GARDEN object-oriented prototype environment and the FIELD program development environment.

GARDEN The GARDEN programming environment provides workstation-oriented support for object-oriented systems. It provides graphical, textual, and browser-style editors that offer multiple consistent views of the underlying objects. It features a visual display facility that allows the rapid customization of program and data structure diagrams. It offers debugging facilities, automatic program visualization, a system browser, and an internal compiler for object-based programs. All these tools are provided with a consistent mouse and menu-oriented interface.

GARDEN has been used to prototype a wide variety of different languages, mainly visual languages, reflecting different paradigms. This includes various dataflow languages, flow charts, finite-state automata, Petri nets, message passing with named ports, LINDA, dataflow design diagrams, and MultiLisp futures.

FIELD The FIELD project was an effort to show that highly integrated, interactive environments like those on personal machines can be implemented on workstations for classical languages and large-scale programs. At the same time, it showed that the more advanced capabilities of workstations can be used to attain a more productive and powerful environment by providing functionality not found on personal machines or in standard software engineering environments. FIELD was designed to provide a production environment for both research and instructional programming at Brown. To make the environment operational as soon as possible and to ensure that it could be maintained and would work in an educational environment, the system was designed to be simple and inexpensive. We made use of existing tools wherever possible: standard UNIX tools, workstation software developed at Brown, and other available tools and software. At the same time, the system attempts to provide a testbed environment in which new tools such as program animation systems can be easily incorporated.

These goals were accomplished in FIELD by providing a consistent graphical front end and a simple integration framework that allows existing and new UNIX tools to cooperate. The front end was based on an updated version of the set of tools called the Brown Workstation Environment. This included a variety of input interfaces including static, pull-down and pop-up menus, dialog boxes and scroll bars; a powerful and extensible base editor; a geometry package; drawing packages including one for the automatic layout and display of structured diagrams; an integrated help facility; and an application window manager.

The FIELD Integration Framework The integration framework, a principal contribution of the FIELD system that is now used in the HP SoftBench product, allows a wide variety of tools to be tied together with minimal effort. It combines a communications mechanism that we call "selective broadcasting" with an annotation editor that provides consistent access to the source in multiple contexts and with a set of specialized interactive analysis
tools. In selective broadcasting all tools talk to a central message server. Each tool registers a set of patterns with the server. Tools communicate by sending messages to the server and receiving those messages that match their registered patterns. This approach is easy to implement and extend and offers several advantages over the more traditional integration mechanisms involving program databases or a single, massive system. The FIELD system demonstrates that this simpler approach is both feasible and desirable.

The environment includes a debugger, editors, visual interfaces to UNIX make and gprof, event and stack viewers, a cross reference tool based on a relational database, a call-graph viewer, and a dynamic data structure display facility.

2 Compiler Optimization

Work on compiler optimization focused on developing powerful analysis and transformation techniques applicable in languages that make heavy use of data abstraction or object-oriented programming constructs. The analysis techniques developed include constant propagation [28], value numbering [27], and the analysis of pointer and structure usage [25]. These techniques determine a set of facts about a program that are true on all possible executions of that program. Such knowledge can be used by the transformation techniques to produce more efficient code. The transformation techniques that have been developed include the detection and removal of loop-invariant program constructs, redundant computation elimination and efficient conversion to an intermediate form that makes analysis faster and more precise [26].

3 Incremental Computation

In the area of incremental computation, research focused on the specification and incremental evaluation of generalized constraint satisfaction systems [31]. Such systems can be used to model a wide variety of problems in such areas as computer graphics, VLSI and programming environments.

4 Object-Oriented Databases

At the time of the completion of the contract a prototype object-oriented database, Encore/ObServer, was being completed to be used to examine cooperative design, optimization, and performance issues, some of which are theoretical [14]. We succeeded in producing an object-oriented query algebra which fits cleanly into our model of strong type checking and encapsulation [15,16,17]. It solves some of the problems, including the meaning of equality, that are introduced by object identity. We implemented this algebra in the context of the Encore database system.

We defined a new mechanism for ObServer to handle cooperative transactions in a more structured way [11,13]. We developed a precise definition of a transaction hierarchy, which is a mechanism for nesting transactions and transaction groups in a way that reflects the
natural structure of the underlying applications. Each internal node in the hierarchy is called a transaction group. Each transaction group is a locus of control for its cooperating member transaction groups and transactions.

The concurrency control and recovery mechanisms for transaction groups need to be tailored to reflect the needs of its members [18,19,20]. We defined algorithms to allow the transaction group more control over the concurrency control mechanisms. Rather than use serializability as the correctness criterion, we allow the transaction group to define and enforce patterns of acceptable behavior. We also developed a theory of recovery for cooperative transactions [12]. In addition, we defined a set of data structures and algorithms which can be used to recover from aborts and failures of transaction group members.

5 Theoretical Foundations for Databases and Typing

In the work on object identity as a query language primitive we clarified some of the foundations of object-oriented databases. We showed that, while these foundations do involve an excursion into new realms, they are wholly compatible with the classical theory of logic and databases. In particular, we demonstrated that the concept of object identity is a powerful programming primitive for database query languages by having it as the centerpiece of a data model with a rich type system and a complete query language called Identity Query Language (IQL) [6]. Our structural and operational framework generalizes most of the previous work on complex-object databases and on logical database query languages. Programs in IQL consist of logical rules that are strongly typed. Also, because we allow union types, we can easily add type inheritance to the model.

There is varied folklore regarding the complexity of ML typing, and even some erroneous previously published results (both linear and quadratic time complexities have been claimed for the problem) [9]. We identified a fundamental inefficiency that is due to the presence of LET. This is a surprising result and the techniques used in its proof are likely to lead to a better understanding of type inference algorithms. Our result [8] is also a contribution to mathematical logic because it is directly applicable to first-order typability of the lambda calculus with marked redexes.

Other problems studied in this area include schema updates [7] and the complexity of unification [2,10].

6 Support for Medium-Grained Parallelism

Support for shared-memory parallel programming with parallel I/O was developed with our lightweight process package, Threads [4]. This package supports multiple threads of control on a variety of single-processor workstations and on a shared-memory multiprocessor (the Encore Multimax). Threads is a strictly user-level implementation of lightweight processes for UNIX: it requires no changes to UNIX (though in its multiprocessor version it assumes a means for sharing memory among processes) and allows users to program with multiple (parallel) threads of control with a minimal execution-time penalty, thus enabling the user to
take full advantage of multiple processors within a computation. Our threads are relatively inexpensive to create (less than 200 microseconds is required to create a thread on the Encore with NS32332 processors); their effective creation cost can be greatly reduced by using various caching techniques. We support a number of synchronization techniques, including semaphores and monitors. Both preemptive (time-sliced) and non-preemptive scheduling are available for those situations in which there are more threads than processors. We support the concepts of exceptions and interrupts and give the programmer the means for dealing with both.

Brown Threads has been distributed to two or three dozen sites worldwide; it has been turned into a product by Encore (Encore Parallel Threads). Our work was listed as being influential in the design of Posix Threads, a proposed standard. We also worked with the Encore on operating system support for medium-grained parallelism [5].

7 Exploiting I/O

A framework for analyzing the effect of I/O limitations on performance was developed [30]. Optimal algorithms and matching lower bounds in terms of the required number of I/O operations between main memory and secondary storage were developed for the problem of performing lattice computations in VLSI. The optimal architecture was also outlined [29]. In the realm of incremental computing, complexity models were developed, and several problems were classified as to their degree of incremental complexity. This explains why some problems have not been able to be made dynamic, in the areas of incremental compiling and database queries. In addition, efficient incremental algorithms were designed for some important combinatorial and geometric applications.

8 Fault Tolerance in Parallel Algorithms

We developed a software approach to fault tolerance that provides an effective way of transforming existing parallel algorithms so that when processor fail-stop errors occur, the algorithm dynamically reconfigures its computation and can successfully proceed to completion, as long as at least one processor survives. Most importantly, the transformed parallel algorithm is robust in the sense that the efficiency of the original parallel algorithm is not significantly affected: for any pattern of fail-stop errors, the original (parallel-time x processors) performance is increased, in the worst case, by a multiplicative factor polylogarithmic in the input size. Our contributions were: (i) defining the notion of robust parallel computation, and (ii) showing its feasibility and wide applicability [3].

9 Programmable Systolic Arrays

We developed a general architecture for programmable systolic arrays that incorporates the following features: regular topology with nearest-neighbor connections, synchronous SIMD control, interprocessor communication using shared registers, and stream-based I/O. At the
time of the completion of the contract we were testing the implementation of a prototype of this architecture, the Brown Systolic Array (B-SYS), in 2-micron CMOS [22]. B-SYS is a highly parallel array of simple processing elements tuned for solving combinatorial problems, including sequence comparison. We programmed a number of algorithms using B-SIM, a software emulator that has helped us refine the architecture. Preliminary estimates indicated that B-SYS will provide supercomputer performance for the applications of interest.

10 Coping with Uncertainty in VLSI CAD

We developed a theoretical framework based on interval algebra to address the problem of uncertain data in the analysis of integrated circuits [21]. We explored applications for this framework in several areas of VLSI design, including switch-level simulation, timing analysis, and placement. For each area, we created an application-specific interval algebra and incorporated this algebra within an existing algorithm for solving the problem. The result is a new algorithm which computes bounds on the result when given interval inputs.

The different applications reveal the strengths and weaknesses of the interval paradigm. Among its many strengths are its versatility, simplicity, and speed. Its main liability is that it may produce overly conservative bounds for some applications. For these cases, we proposed alternative interval-based techniques that improve the bounds but require more computation time.

11 Parallelism in VLSI Synthesis

We were developing parallel algorithms for VLSI synthesis and testing them experimentally on the Connection Machine. We were also developing complexity-theoretic results showing that some well-known heuristics for these problems are not parallelizable [23]. We developed a parallel algorithm for computing the transitive reduction of an interval DAG which is equivalent to a parallel algorithm for computing a minimum-distance constraint DAG from a VLSI layout [24]. We also showed that key heuristics for graph partitioning, an NP-complete problem with applications to VLSI placement, memory segmentation and processor allocation, are P-complete or harder [1]. We also developed parallelizable heuristics for this and related graph embedding problems that are very efficient. Our graph partitioning algorithm can handle graphs with at least one million vertices and more than two million edges, graphs of previously prohibitive sizes, and within a few minutes gives partitions that are within 2% of the best ever found.

12 Bibliography


Databases


VLSI CAD


Compilers


ARTICLES, CONFERENCE PROCEEDINGS PAPERS, AND BOOK CHAPTERS SINCE 1986

ONR N00014-83-K-0146, DARPA Orders 4786, 6320

1986


1987


1988


1989


Zdonik, Stanley, and David Maier, eds., Readings in Object-Oriented Databases, Morgan-Kaufmann, 1989.


1990


Reiss, Steven, "Interacting with the FIELD Environment," *Software Practice and Experience*, 1990.


1991


Reiss, Steven, "Interacting with the FIELD Environment," *Software Practice and Experience*, June 1990.


Savage, J.E., and M.G. Wloka, "Parallelism in Graph-Partitioning," to appear in *Journal of Parallel and Distributed Computing*.

Savage, J.E., and M.G. Wloka, "Parallel Graph-Embedding and the Mob Heuristic," submitted for publication.


TECHNICAL REPORTS UNDER N00014-83-K-0146


Marc H. Brown and Steven P. Reiss, *Toward a Computer Science Environment for Powerful Personal Machines*, CS-83-04, October 1983


Bernard Chazelle and Janet Incerpi, *Unraveling the Segment Tree*, CS-83-15, June 1983

Jeffrey S. Vitter, *Random Sampling with a Reservoir*, CS-83-17, July 1983


Stavros S. Cosmadakis and Paris C. Kanellakis, *Functional and Inclusion Dependencies: A Graph-Theoretic Approach*, CS-83-21, October 1983

Bernard Chazelle and Janet Incerpi, *Triangulation and Shape Complexity*, CS-83-22, October 1983


Steven P. Reiss, *Graphical Program Development with PECAN Program Development Systems*, CS-84-04, February 1984


Steven P. Reiss, *An Approach to Incremental Compilation*, CS-84-09, April 1984


H. Kocak, M. Merzbacher and M. Strickman, *Dynamical Systems with Computer Experiments at the Brown University Instructional Computing Laboratory*, CS-84-14, June 1984


Alessandro Giacalone, *Algebraic Definitions in Programming Environments*, CS-84-17, August 1984


Wen-Chin Chen and Jeffrey S. Vitter, *Deletion Algorithms for Coalesced Hashing*, CS-84-20, November 1984


Wen-Chin Chen, *The Design and Analysis of Coalesced Hashing*, CS-84-22, November 1984

Mark E. Sommer and John E. Savage, *SAS, A Systolic Array Simulator*, CS-85-02, January 1985

William R. Cook and Brian A. Dalio, *The Design of a Memory Management Unit: A SLAP/Lucifer Case Study*, CS-85-03, February 1985

Christopher J. Van Wyk and Jeffrey S. Vitter, *Hashing with Lazy Deletion is Optimal in Time and Space*, CS-85-05, April 1985

Russell Ellsworth, *An Ideographic System Build and Execute Facility*, CS-85-06, April 1985


George Lipsker, *DEXEC - A Distributed Exec System*, CS-85-12, May 1985


Swaminathan Manohar and Gerard M. Baudet, *PE-1: A Ser-Coprocessor for Solving PDEs*, CS-86-06, March 1986
John J. Bowe, *Mail System Interface Issues With a UNIX-Macintosh Implementation*, CS-86-10, April 1986 (Sc.M. project)
Peter Wegner, *Classification as a Paradigm for Computing*, CS-86-11, May 1986
Steven P. Reiss, *Visual Languages and the GARDEN System*, CS-86-16, September 1986
Steven P. Reiss, *An Object-Oriented Framework for Graphical Programming*, CS-86-17, March 1986
Steven P. Reiss, *GARDEN Tools: Support for Graphical Programming*, CS-86-18, April 1986
Steven P. Reiss and Joseph N. Pato, *Displaying Program and Data Structures*, CS-86-19, April 1986
Robert V. Rubin, Steven P. Reiss and Eric J. Golin, *Compiler Aspects of an Environment for Programming by Demonstration*, CS-86-21, July 1986
Peter Wegner, *Perspectives on Object-Oriented Programming*, CS-86-25, December 1986
Peter Z. Revesz, *Graph Reachability Problems*, CS-87-02, January 1987
Brian A. Dalio, *DeCo - A Hierarchical Device Compilation System*, CS-87-08, Ph.D. Dissertation, May 1987
Richard P. Hughey and Daniel P. Lopresti, *Location of Protein Encodings in Nucleic Acids*, CS-87-09, May 1987


Gerard M. Baudet and Swaminathan W. Manohar, *Supercomputing with VLSI: Sorting*, CS-87-12, June 1987

Peter Wegner, *The Object-Oriented Classification Paradigm*, CS-87-13, June 1987

Peter Wegner, *Dimensions of Object-Based Language Design*, CS-87-14, July 1987

Lynn A. Stein, *Delegation is Inheritance*, CS-87-15, July 1987


Peter Wegner and Stanley B. Zdonik, *Why Like Isn’t Like Is-a (or As You Like It)*, CS-87-17, July 1987


Kim B. Bruce and Peter Wegner, *An Algebraic Model of Subtype and Inheritance*, CS-87-21, August 1987


Bowen Alpern, Mark N. Wegman, and F. Kenneth Zadeck, *Detecting Equality of Variables in Programs*, CS-87-24, October 1987


Thomas W. Doeppner and Alan J. Gebele, *C++ on a Parallel Machine*, CS-87-26, November 1987


Peter Wegner, *Object-Oriented Concept Hierarchies*, CS-88-06, March, 1988


Lynn Andrea Stein and Stanley B. Zdonik, *Clowers: The Dynamic Behavior of Types and Instances*, CS-89-42, November 1989


Roberto Tamassia and Jeffrey S. Vitter, *Parallel Transitive Closure and Point Location in Planar Structures*, CS-89-45, October 1989; revised June 1990


Steven P. Reiss, *Interacting with the FIELD Environment*, CS-89-51, May 1989


Lynn Andrea Stein, *A Preference-Based Approach to Inheritance*, CS-90-08, April 1990


Cheryl L. Harkness and Daniel P. Lopresti, *VLSI Placement using Uncertain Costs*, CS-90-12, May 1990


Robert P. Goldman. *A Logic-programming Approach to All-paths Parsing*. CS-90-17, August 1990


Paul G. Howard and Jeffrey Scott Vitter. *Analysis of Arithmetic Coding for Data Compression*. CS-91-03, January 1991


Peter Wegner, *Paradigms of Interpretation and Modeling*, CS-91-09, February 1991


Dilip K. Barman, *RelType: Relaxed Typing for Intelligent Hypermedia Representations*, CS-91-26, April 1991

Marian H. Nodine, *Conflict, Queueing; and Deadlocks in Cooperative Transaction Hierarchies*, CS-91-27


