Database Research at AT&T Bell Laboratories

H. V. Jagadish[1]

Bell Laboratories is the Research and Development arm of AT&T. There is today tremendous support for database research in Bell Labs and in AT&T. This is expected to continue since database technology is recognized as being central not just to Teradata, NCR, and AT&T's computing business but to its core communication business as well. What you see described below is a survey of the current state of a young and growing research effort in the database area. You should expect to see a stronger "resume" a few years down the road.

Research at AT&T Bell Labs is never directed from above, towards specific projects or objectives. Individual researchers select their areas of research based on a combination of factors that include individual skill and interest, other on-going activity and potential for collaboration, and the likelihood of intellectual or monetary impact. While research activity can range from the highly theoretical to the completely applied, it is expected that most research be performed within the framework of some larger objective; and this framework is typically provided by a prototype system.

In addition to the core set of people here full-time, often there are visitors. You will see the names of many of these in the publications listed. For instance, most recently Oded Shmueli from the Technion spent a year with us. Currently, Avi Silberschatz is visiting from Austin.

1. Database Theory (Inderpal Mumick, S. Sudarshan, Mihalis Yannakakis)

A variety of areas have been covered in the past, including concurrency control and locking, nested transaction models, query optimization, acyclic schemas, dependencies etc. In particular, recent results have been obtained in efficient algorithms for query processing problems; in understanding the power and limitations of recursive query languages, first as a means of expressing problems and second as a means of expressing algorithms; and in defining semantics for query languages with negation and aggregation.

[1] M. Yannakakis, "Graph Theoretic Methods in Database Theory," (invited paper), *Proc. 9th ACM Symp. on Principles of Database Systems*, 1990, pp. 230-242.

[2] J. D. Ullman, M. Yannakakis, "The Input/Output Complexity of Transitive Closure," *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, 1990, pp. 44-53.

[3] F. Afrati, S. Cosmadakis, M. Yannakakis, "On Datalog vs. Polynomial Time," *Proc. 10th ACM Symp. on Principles of Database Systems*, 1991, pp. 13-25.

[4] C. H. Papadimitriou, M. Yannakakis, "Tie-Breaking Semantics and Structural Totality," *Proc. 11th ACM Symp. on Principles of Database Systems*, 1992, pp. 16-22.

[5] I. S. Mumick and O. Shmueli, "Finiteness Properties of Database Queries," *Proceedings of the Fourth Australian Database Conference*, Brisbane, Australia, February 1993.

[6] A. Levy, I. S. Mumick, Y. Sagiv, and O. Shmueli, "Equivalence, Query-Reachability, and Satisfiability in Datalog," *Proceedings of the Workshop on Deductive Databases, Joint International Conference and Symposium on Logic Programming*, Washington DC, November 1992.

[7] I. S. Mumick and O. Shmueli, "Aggregation, Computability, and Complete Query Languages," *Proceedings of the Workshop on Structural Complexity and Recursion-theoretic Methods in Logic Programming, Joint International Conference and Symposium on Logic Programming*, Washington DC, November 1992.

[8] S. Sudarshan and Raghu Ramakrishnan, "Optimizations of Bottom-Up Evaluation with Non-Ground Terms," *Proceedings of the Workshop on Deductive Databases, Joint International Conference*

---

1. Portions of this article were written by N. Gehani, R. Greer, T. Griffin, B. Hillyer, D. Lieuwen, and A. Skarra. The author also acknowledges assistance from R. Brachman, D. McGuinness, L. Terveen, and M. Yannakakis.

and Symposium on Logic Programming, Washington DC, November 1992.

[9] S. Sudarshan, Divesh Srivastava, and Raghu Ramakrishnan, "Extending the Valid and Well-Founded Semantics for Aggregation," submitted for publication.

2. Ode (Alex Biliris, Narain Gehani, H. V. Jagadish, Dan Lieuwen, Bill Roome, Ted Roycraft, with contributions by several others, including most importantly Rakesh Agrawal, Steve Buroff, Shaul Dar, and Oded Shmueli.)

Ode is a database system and environment based on the C++ object paradigm. Ode provides facilities for creating persistent and versioned objects, defining sets, iterating over sets and clusters of persistent objects, and for associating constraints and triggers with objects. Currently there are over 150 installations of Ode. An initial release of Ode can be licensed at no charge to academic institutions. Please send mail to nhg@research.att.com if interested.

[1] R. Agrawal and N.H. Gehani, "Ode (Object Database and Environment): The Language and the Data Model," *Proc. ACM-SIGMOD 1989 Int'l Conf. Management of Data*, Portland, Oregon, May-June 1989, pp. 36-45.

[2] S. Dar, R. Agrawal, and N. H. Gehani, "The O++ Database Programming Language: Implementation and Experience," *Proc. IEEE 9th Int'l Conf. Data Engineering*, Vienna, Austria, 1993.

2.1 User Interface

Ode provides a variety of interfaces: O++ (a programming language), OdeView (graphical), OdeFS (file system), CQL++ (SQL-like), and Noodle (declarative language). The primary interface is the database programming language O++, which is based on C++. It offers one integrated data model for both database and general purpose manipulation.

OdeView is an X-windows based graphical interface. As a browser, it offers a consistent display model including "synchronized browsing", in which changes in one window cause the contents of other related windows to change in synchrony. In addition to browsing capabilities, OdeView also permits manipulation of the database by invoking methods and entering values, all in a point-and-click and fill-in-the-blank environment.

OdeFS is a UNIX-like file system interface to Ode. Users can access and manipulate objects as if they were files, using popular shell commands like `ls`, `cat`, and `vi`. OdeFS is an NFS server that intercepts NFS requests and passes them on, with appropriate modifications, either to the regular file system or to the Ode database.

`cql++` is an SQL-like language designed to define, access and manipulate an object oriented database. Noodle is another declarative language, with the advantage that it is much more expressive, though it is not SQL-like in syntax.

All the different interfaces for Ode are object compatible in that objects created through one interface can be accessed and manipulated by any of the others.

[3] R. Agrawal and N.H. Gehani, "Rationale for the Design of Persistence and Query Processing Facilities in the Database Programming Language O++," *2nd Int'l Workshop on Database Programming Languages*, Portland, OR, June 1989.

[4] R. Agrawal, N. H. Gehani, and J. Srinivasan, "OdeView: The Graphical Interface to Ode," *Proc. ACM-SIGMOD 1990 Int'l Conf. on Management of Data*, 1990, pp. 34-43.

[5] N. H. Gehani, H. V. Jagadish, and W. D. Roome, "OdeFS: A File System Interface to an Object-Oriented Database," AT&T Bell Labs Technical Memorandum, 1992.

[6] S. Dar, N. H. Gehani, and H. V. Jagadish, "CQL++: An SQL for a C++ Based Object-Oriented DBMS," *Proc. of Int'l Conf. on Extending Database Technology*, Vienna, Austria, Mar. 1992.

[7] I. S. Mumick and K. A. Ross, "SWORD: A Declarative Object-oriented Database Architecture," submitted for publication.

2.2 Query Optimization

The goal is to take a program in an imperative DBPL, such as O++, and optimize its execution using knowledge of database statistics, with substantial rewriting, as in traditional database query optimization. For instance, we use programming language analysis to determine whether or not the loops in O++ can be executed in a set oriented fashion instead of using tuple- (object-) iteration-semantics. If a loop can execute in a set-oriented fashion without violating programming language semantics, then database optimization techniques can be used to choose an efficient method of executing it in a set-oriented fashion. The technique has been used to produce improved sequential code and to parallelize code.

[8] Daniel Lieuwen and David DeWitt, "A Transformation-based Approach to Optimizing Loops in Database Programming Languages," *Proc. ACM-SIGMOD Int'l Conf. on the Mgmt. of Data*, June 1992.

[9] Daniel Lieuwen, David DeWitt, and Manish Mehta, "Parallel Pointer-based Join Techniques for Object-Oriented Databases," *Proc. 2nd International Conference on Parallel and Distributed Information Systems*, San Diego, CA, Jan. 1993.

2.3 Large Objects

Ode provides support for "large" objects. An object larger than a page is classified as large. The default page size is 4K bytes. Three interfaces are supported:

1. *Transparent access*: A large object can be accessed and manipulated possible just like a small object.
2. *File like I/O*:
   a. Portions of a large object can be accessed and manipulated without having to bring the whole object into memory.
   b. It is possible to do large object I/O without bringing the whole object into memory *all at once*.
3. *Asynchronous Reads*: Portions of large objects can be "prefetched" from disk to memory without blocking. Thus when this large object is accessed later, the transaction does not have to block for the object (portion) to be read from disk; instead, it will find the object (portion) in memory and immediately accessible.

[10] A. Biliris, "An Efficient Database Storage Structure for Large Dynamic Objects," *Proc. IEEE Data Engineering Conference*, Phoenix, Arizona, February 1992, pp. 301-308.

[11] A. Biliris, "The performance of Three Database Storage Structures for Managing Large Objects," *Proc. ACM-SIGMOD Conference*, San Diego, California, June 1992, pp. 276-285.

2.4 Transactions

Object-oriented databases are often used for non-traditional database applications where long-running transactions are often necessary, but cooperation is possible. We have developed a model for cooperation in which individual transactions can choose whether to cooperate, and the database system can actually manage this cooperation. We have also developed a set of primitives that can be used to provide a flexible transaction facility, thereby making it possible to use any Ode installation with a choice of extended transaction models.

More recently, we have started looking at implementation issues, first for traditional locks in an object-oriented environment, and subsequently for the advanced transaction models discussed above.

[12] H. V. Jagadish and O. Shmueli, "A Proclamation-Based Model for Cooperating Transactions," *Proc. of the 18th Int'l Conf. on Very Large Databases*, Vancouver, B.C., Canada, August 1992.

[13] A. Biliris, S. Dar, N. H. Gehani, H. V. Jagadish, and K. Ramamritham, "A Flexible Transaction Facility for an Object-oriented Database," submitted for publication.

[14] H. V. Jagadish and D. Lieuwen, "Multi-granularity Locks in an Object-Oriented Database," submitted for publication.

2.5 Compose -- Active Facilities

Ode permits the specification of integrity constraints and triggers. These are associated with class definitions. An integrity constraint in a class definition applies to all instances of the class, and all derived classes. A trigger in a class definition is considered a definition, and must be activated explicitly at specific object instances. In both cases, the action to be taken can be the execution of an arbitrary procedure.

Trigger and constraint conditions are *intra-object* in that the specified condition is evaluated with respect to a particular object only when the object is modified (or has some other *event* occur at it). Inter-object conditions can be specified declaratively, but are then compiled into one or more corresponding intra-object conditions associated with all the relevant classes.

While an integrity constraint represents an invariant that must be checked upon each update to an object, a trigger condition may be checked upon the occurrence of an arbitrary event, or event pattern. Besides the creation, modification, and deletion of an object, an event may be an object access, the execution of a method, the begin, abort or commit of a transaction, or even the ticking of a system clock. A pattern of events of interest, such as "fifth large withdrawal" or "abort of transaction following an invocation of a sub-transaction" can be specified declaratively using *event expressions*. An event in an event expression may have attributes, derived from the event itself, such as parameters of the method for a method execution event, or obtained from the state of the database at the time of event occurrence. The attributes of a

composite event expression are derived from the attributes of its constituent events.

[15] N.H. Gehani and H. V. Jagadish, "Ode as an Active Database: Constraints and Triggers," *Proc. 17th Int'l Conf. Very Large Data Bases*, Barcelona, Spain, 1991, pp. 327-336.

[16] H. V. Jagadish and X. Qian, "Integrity Maintenance in an Object-Oriented Database," *Proc. of the 18th Int'l Conf. on Very Large Databases*, Vancouver, BC, Canada, August 1992.

[17] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "Event Specification in an Active Object-Oriented Database," *Proc. ACM-SIGMOD 1992 Int'l Conf. on Management of Data*, San Diego, CA, 1992.

[18] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "Composite Event Specification in Active Databases: Model & Implementation," *Proc. of the 18th Int'l Conf. on Very Large Databases*, Vancouver, BC, Canada, Aug 1992.

[19] H. V. Jagadish and O. Shmueli, "Synchronizing Trigger Events in a Distributed Object-Oriented Database," *Proc. Int'l Workshop on Distributed Object Management*, Edmonton, Alberta, Canada, August 1992.

2.6 Versions

Object versioning in Ode is orthogonal to type, that is, versioning is an object property and not a type property. Versions of an object can be created without requiring any change in the corresponding object type definition, all objects can be versioned, and different objects of the same type can have a different number of versions. Both dynamic and static bindings to version references are supported. Temporal as well as derived-from relationships between versions are maintained automatically.

[20] R. Agrawal, S.J. Buroff, N. H. Gehani, and D. Shasha, "Object Versioning in Ode," *Proc. IEEE 7th Int'l Conf. Data Engineering*, Tokyo, Japan, Feb. 1991.

3. Datashare(Rick Greer, Naser Barghouti, John Mocenigo, Andrea Skarra, John Snively)

DataShare offers a full spectrum of data management and programming language services. DataShare's multi-paradigm language Cymbal synthesizes procedural commands with the declarative constructs of symbolic logic, set theory, and SQL. It supports:

- using the full SQL DML (i.e., selects plus transaction-based updates) to query UNIX flat file databases
- using a procedural language that has the power of awk
- embedding SQL in this procedural language to create a 4GL more expressive and easier-to-use than C-embedded SQL
- using such logic and set theory contributions to Cymbal as generalized transitive closure, set-formers, fully general quantification, complete missing value control, conditional definitions of new variables, additional and fully general aggregate functions, and data buffer reuse.
- combining freely any of the above language constructs.

Furthermore, both SQL and Cymbal data management may make use of set-valued fields, horizontal table partitioning, in-core databases, update transactions with concurrency and recovery, and user-provided C-extensions. And, of course, standard UNIX tools such as awk and grep can operate directly on the same data that DataShare operates on.

DataShare translates Cymbal to C (which is also compatible with C++). DataShare users frequently use DataShare to generate a high-level data management library which they call from their application code and which in turn may call user application code.

The DataShare architecture is not client-server. DataShare relies on the operating system to provide services typically provided by traditional database servers.

[1] R. Greer, "Datashare and the Fourth Generation Language Cymbal," *Proc. AT&T Database Day*, Sep. 1992.

3.1 SLEVE

SLEVE is a software package that takes as input a semantic lock specification from an application, and as output produces a functional interface to the operating system locking facility. The functions in the interface request locks from the OS in a way that simulates the concurrency behavior defined by the semantic specification. The operating system provides a fixed set of lock types, namely *Read* and *Write* with the usual conflict semantics. With SLEVE, the systems can define and implement arbitrary semantic locks for more concurrency; they are not limited to the operating system's lock types.

[2] A. H. Skarra, "SLEVE: Semantic Locking for EVEnt synchronization," *Proceedings of Ninth International IEEE Conference on Data Engineering*, 1993.

4. Persi(Naser Barghouti, Andrea Skarra; originally Alex Wolf)

Persi is a C++ library that provides a persistence mechanism to developers of C and C++ applications. It implements persistence through the native abstraction capabilities of the language, particularly classes, inheritance, and virtual functions, and it requires no modifications to either the C++ language processor or the run-time system. In this approach, persistence is simply another bit of compatible functionality that is placed at the disposal of application developers in the same way that other general-purpose abstractions are provided and supported.

In addition, Persi provides a basis for associative retrieval. It provides a way to create and maintain indices on collections of objects and a way to iterate over those indices. Current work on Persi involves the design of a concurrency control algorithm that focuses on heavily shared objects such as collections and indices. The goal of the algorithm is to optimize concurrent access to these objects by exploiting their semantics, while producing histories that satisfy semantic serializability. The algorithm uses several techniques to achieve its goal, including a strategy for delaying the execution of operations (and the acquisition of locks) and a protocol based on a set of semantic lock types, implemented using SLEVE.

[1] A. L. Wolf, "An Initial Look at Abstraction Mechanisms and Persistence," in *Implementing Persistent Object Bases: Principles and Practice. The Fourth International Workshop on Persistent Object Systems*, A. Dearle, G. M. Shaw, and S. B. Zdonik (eds.), Morgan Kaufmann Publishing Inc., San Mateo, CA, 1991.

[2] A. H. Skarra, "Localized Correctness Specifications for Cooperating Transactions in an Object-Oriented Database," *Office Knowledge Engineering*, Vol. 4, no. 1, Feb. 1991.

5. DDB(A. Singhal, R. Arlein, C-Y Lo, N. Parikh)

DDB is an object-oriented design database for VLSI/CAD. By exploiting knowledge of the CAD domain, the database is able to provide performance an order of magnitude better than is possible using general-purpose commercial object-oriented databases. DDB has been used in a diverse set of CAD applications and has improved productivity by rendering the architecture of CAD systems more modular.

[1] A. Singhal, R. Arlein, and C-Y Lo, "DDB: An Object-oriented Design Database for VLSI/CAD," *Proc. AT&T Database Day*, Sep. 1992.

[2] N. Parikh, C-Y Lo, A. Singhal, K. W. Wu, "HS: A Hierarchical Search Package for CAD data," *Proc. ICCAD Conf.*, Nov. 1989.

6. Classic(Ron Brachman, Alex Borgida, Deb McGuinness, Peter Patel-Schneider, Lori Alperin Resnick, and Tom Kirk)

CLASSIC is a data model that encourages the description of objects not only in terms of their relations to other known objects, but in terms of a level of intensional structure as well. The CLASSIC language of *structured descriptions* permits i) partial descriptions of individuals, under an 'open world' assumption, ii) answers to queries either as extensional lists of values or as descriptions that necessarily hold of all possible answers, and iii) an easily extensible schema, which can be accessed uniformly with the data. One of the strengths of the approach is that the same language plays multiple roles in the processes of defining and populating the DB, as well as querying and answering.

The CLASSIC database can actively discover new information about objects from several sources: it can recognize new classes under which an object falls based on a description of the object, it can propagate some deductive consequences of DB updates, it has simple procedural recognizers, and it supports a limited form of forward-chaining rules to derive new conclusions about known objects.

[1] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick, "CLASSIC: A Structural Data Model for Objects," *Proc. ACM-SIGMOD 1989 Int'l Conf. on Management of Data*, Portland, OR, 1989.

[2] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alex Borgida, "Living with CLASSIC: When and How to Use a KL-ONE Like Language," in *Principles of Semantic Networks: Explorations in the representation of knowledge*, John Sowa (ed.), Morgan-Kaufmann, San Mateo, CA, 1991.

7. RightPages(G. A. Story, L. O'Gorman, J. Q. Arnold, H. V. Jagadish, E. Szurkowski, with contributions from C. Faloutsos, D. Fox, L. L. Schaper)

RightPages is a multi-disciplinary research project to create a system that provides on-line library services. The intention is to retain the "look-and-feel" of the

original journals while enhancing their utility through image analysis and processing. Three versions are maintained for each journal page: a bit-map of the image, for display purposes; a text version, obtained through optical character recognition, for text indexing and retrieval; and a skeleton structure version, that can be used to interpret mouse clicks in particular regions of the page, and also for retrieval by image content (eventually). An initial prototype of the RightPages system has now been moved outside the laboratory and is undergoing field trials as a product.

Some of the database issues associated with RightPages are standard information retrieval issues for text. However, there are potentially sophisticated image analysis, indexing, and retrieval problems involved.

[1] G. A. Story, L. O'Gorman, D. Fox, L. L. Schaper, and H. V. Jagadish, "The RightPages Image-Based Electronic Library for Alerting and Browsing" *IEEE Computer*, Vol. 25, no 9, Sep 1992, pp. 17-27.

[2] C. Faloutsos and H. V. Jagadish, "Hybrid Index Organizations for Text Databases," *Proceedings of EDBT*, Vienna, Austria, 1992.

[3] H. V. Jagadish, "On Indexing Line Segments," *Proceedings of the 16th International Conference on Very Large Databases*, Brisbane, Australia, Aug. 1990.

[4] H. V. Jagadish and A. M. Bruckstein, "On Sequential Shape Descriptions," *Pattern Recognition*, 25(2), 1992, pp. 165-172.

[5] H. V. Jagadish, "A Retrieval Technique for Similar Shapes," *Proc. ACM-SIGMOD Int'l Conf. on the Management of Data*, Denver, CO, May 1991.

8. IMACS(R.J. Brachman, P.G. Selfridge, L.G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D.L. McGuinness, and L.A. Resnick)

Organizations have begun to view databases as potentially rich sources of new and useful knowledge. Various approaches to "discovering" such knowledge have been proposed. Our work identifies an important and previously ignored discovery task that we call "data archaeology". Data archaeology is a skilled human task, in which the knowledge sought depends on the goals of the analyst, cannot be pre-specified, and emerges only through an iterative process of data segmentation and analysis. We have built a system called IMACS that supports this task with a natural, object-oriented representation of an application domain; semi-automatic population of the representation from multiple, large databases, including periodic updates; and a powerful and flexible user interface that supports interactive exploration.

[1] R.J. Brachman, P.G. Selfridge, L.G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D.L. McGuinness, and L.A. Resnick, "Knowledge Representation Support for Data Archaeology," *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD, 1992.

9. Integrity Maintenance(T. Griffin, H. Trickey, C. Tuckey; early work by X. Qian)

AT&T's telecommunications switch, the 5ESS, contains a relational database subsystem used for controlling all aspects of the switch's operation. One research project has designed and implemented a declarative language for expressing database constraints. These constraints are automatically compiled to checking code. In addition, update constraints are generated from transaction specifications to ensure that transactions do not leave the database in a state violating the global integrity constraint. This represents a tremendous improvement over the former practice of expressing constraints in English and manually translating them into checking code.

[1] X. Qian, "An Effective Method for Integrity Constraint Simplification," *Proc. IEEE 4th Int'l Conf. Data Engineering*, 1988.

[2] T. Griffin, H. Trickey, and C. Tuckey, "Update Constraints for Relational Databases," submitted for publication.

10. BBFS--Broadband Filesystem (Bruce K. Hillyer, Bethany S. Robinson)

BBFS is a filesystem research effort to satisfy the storage needs of communication- and computation-intensive applications. The key idea is to adapt to the performance and semantic requirements of new applications via extensibility in the filesystem application interface and behavior set. Conventional filesystem calls (open, close, read, write, seek) accept an additional argument containing a list of attribute-value pairs. This provides interface extensibility and a means for applications to offer performance hints. A typed-file notion supports flexibility in the behavior and properties of files.

[1] B. K. Hillyer, B. Robinson, "Aspects of the BBFS broadband filesystem", *PDIS-91, Parallel and Distributed Information Systems*, Miami, December 4-6, 1991, p. 138.

[2] B. K. Hillyer, B. Robinson, "Issues in BBFS, a Broadband Filesystem", *USENIX File Systems Workshop*, Ann Arbor, May 21-22, 1992, pp. 129-130.

11. SWOOP (A. Asthana, A. Biliris, H. V. Jagadish, P. Krzyzanowski, and Jon Solworth)

SWOOP (Store With Object-Orientation and Parallelism) is a high performance, extensible storage system. High performance is achieved through a small carefully optimized kernel, which includes features of parallelism and adaptive logical page size. Extensibility and encapsulation are achieved using event and object-oriented programming. Files (or storage containers) in the storage system are typed, and this typing is distinct from and complementary to any typing that may be present in the stored objects. Object-oriented programming allows the structure to be both hidden and exploited. Event programming permits functions to declare interesting events as they happen, and thus provide "hooks" for extensions without compromising modularity.

[1] A. Asthana, A. Biliris, H. V. Jagadish, P. Krzyzanowski, and J. Solworth, "SWOOP: An Extensible High-Performance Storage System," submitted for publication.

12. SWIM (A. Asthana, M. Cravatts, H. V. Jagadish, P. Krzyzanowski, V. Schumakoff, with contributions from, among others, J. A. Chandross, W. C. Fischer, and S. C. Knauer)

SWIM (Structured Wafer-Scale Intelligent Memory) is a high bandwidth, multi-ported, storage system capable of storing, maintaining, and manipulating data within it, independent of any external processing units. Up to hundreds of active storage elements, each element having some storage and some associated processing logic, function independently or in groups to implement user-defined objects and data structures. Hundreds of transactions can concurrently be processed by mutually exclusive sets of elements. A fast response time is obtained due to the proximity of the processing with the memory, a specialized micro-architecture, and parallelism. We currently have working two prototype systems, each with sixteen active storage elements.

One interesting consequence of the SWIM architecture is that it becomes feasible to use transaction concurrency control paradigms to perform dynamic dependency analysis of a sequential program, in order to execute the program in parallel. The sequential program code is divided into small portions, and these portions are concurrently executed as separate "transactions". By using low-overhead concurrency control mechanisms, we ensure that the concurrent execution of the portions is logically equivalent to the sequential execution of the original program. Close to linear speed-up has been demonstrated in some cases.

[1] A. Asthana, H. V. Jagadish, and S. C. Knauer, "An Intelligent Memory Transaction Engine," *International Workshop on Database Machines*, Deauville, France, 1989.

[2] A. Asthana, H. V. Jagadish, P. Krzyzanowski, and N. Soparkar, "Using Concurrency Control to Parallelize Programs," submitted for publication.