PROGRAMMING LANGUAGES AND COMPILERS

Programming Language Design

Carnegie Mellon University; Guy E Blelloch; ***NSF Young Investigator (NYI) Award: A Functional Data-Parallel Language for High Performance Computers*****; (CCR-9258525 A003); $25,000; 12 months.**

Current research on a first-order functional language (NESL) that supplies parallelism through a set of data-parallel constructs based on vectors is extended in several directions. One is the testing of the language via scientific applications algorithms. Another is the addition of polymorphic types to the language. Yet another is the examination of lazy load-balancing based upon dynamic testing. In addition, implementations on several newer high performance machines is carried out.

University of Washington; Alan H Borning; ***Constraint Imperative Programming*****; (CCR-9402551 A001); $60,000; 12 months.**

This project continues work on Constraint Imperative Programming (CIP) languages. A constraint represents a relation that should be maintained by a computer program. Constraints have a number of desirable properties that make them useful in building systems such as interactive graphical applications. They are declarative statements, rather than imperative statement; a programmer need only to state "what" rather than "how". Additional advantages result from embedding support for constraints in the programming language itself. First, it becomes convenient for programmers to use constraints. Second, the language implementation can ensure that the constraints are checked automatically. Third, a more uniform and elegant language can result from expressing standard constructs as constraints. Finally, more opportunities for optimization are opened to the compiler. In this project, the activities include investigating techniques for compiling object-oriented constraint imperative languages, such as Kaleidoscope, to produce efficient code. To accomplish this, the system does as much compile time analysis of the constraints as possible. Other activities include developing tools for writing, browsing, and debugging constraint imperative programs, investigating the compilation of Kaleidoscope programs to run on shared memory MIMD machines, and testing the utility of this class of languages by programming a set of representative applications. Another important focus is the development of a more efficient and powerful constraint satisfaction algorithm. A final activity is work on debugging support for constraint systems. The work on constraint satisfaction algorithms and debugging is useful both to support constraint imperative languages, as well as other kinds of constraint-based systems.

Williams College; Kim B Bruce; ***Research at Undergraduate Institutions (RUI): Semantics and Design of Object-Oriented Languages*****; (CCR-9424123 A000); $46,712; 12 months.**

This award is for research on the design and semantics of statically-typed object-oriented programming languages. The project has several goals. (1) It builds on and extends the design of the polymorphic, statically-typed object-oriented programming language, PolyTOIL, to provide more power and expressibility while retaining type-safety. (2) It investigates the design of an object-oriented language whose primary relation between types is matching rather than subtyping. This will likely req uire the development of a partial type-inference system to allow the programmer to avoid the proliferation of bounded type parameters. (3) It seeks to create better mathematical (denotational) models for foundational calculi for object-oriented languages. (4) It investigates the development of verification techniques for object-oriented programming languages. (5) It applies current knowledge of the semantics of object-oriented languages to the understanding of concurrent object-oriented languages. These problems are being attacked using formal specifications of type-checking rules and operational semantics. Subject-reduction-style theorems are being used to prove the safety of type systems. Interpreters are being written in order to provide a better understanding of the strengths and weaknesses of the systems and in order to invetigate scaling up of theoretical designs to full-strength programming languages. The project seeks to validate verification techniques by proving the consistency of an operational and axiomatic semantics.

Wake Forest University; Jennifer J Burg; ***Research at Undergraduate Institutions (RUI): Strategies for Fast Execution of Constraint Logic Programs Over a Real-Number Domain;*** **(CCR-9408298 A000); $45,000; 24 months.**

Constraint logic programming is a declarative programming paradigm which has proven to be expressive for computationally difficult problems such as scheduling, air traffic flow management, simulation of dynamic physical systems, solution of differential equations, and options trading analysis. However, while CLP languages are expressive, their execution is not always efficient enough for the problems to which they apply. The goal of this research is to develop strategies for fast execution of constraint logic programs, particularly in the domain of real or rational arithmetic constraints. To this end, incremental Gaussian elimination-based and simplex-based constraint satisfaction algorithms are compared for their efficiency in the context of CLP; the feasibility of intelligent backtracking and independent and-parallelism is investigated; work on asynchronous parallel execution of the inference engine and solver is continued; and parallel incremental algorithms for constraint solving are being developed. The research program has strong undergraduate involvement and research components. Students are involved in the design, implementation, and testing of algorithms, and are exposed to concepts from the theory of computation, programming language design, the linear programming problem, AI search and backtracking strategies, and parallel computation.

Oregon State University; Margaret M Burnett; *Research Initiation Award (RIA): Visual Data Abstraction for Declarative Visual Programming Languages*; (CCR-9308649 A001); $10,000; 0 months.

Visual programming languages (VPLs) offer the hope of higher reliability at lower labor costs, but only if they can be scaled up for realistically-sized programming projects. This research addresses a critical factor in scaling up declarative VPLs, by developing an interactive and visual yet powerful approach to data abstraction. The visual approach adds expressive power to VPLs through simplicity, rather than the traditional approach of providing them at the expense of simplicity. The concepts developed are being implemented in a research declarative VPL and made available to interested researchers. This work builds upon recent work in data abstraction, event- handling, and the scaling-up problem. Visual design issues are addressed, and the current implementation is being prepared to serve as a base research system. The research seeks the incorporation of a visual and flexible approach to code reuse. At the same time, a simple approach to exception handling based on the earlier work on events is being developed. Additional work regarding visual impact and visual support for the approach to data abstraction is expected to enhance the simplicity and effectiveness of the results.

University of Washington; Craig Chambers; *Analysis and Optimization of Object-Oriented Languages*; (CCR-9503741 A000); $49,908; 12 months.

Object-oriented languages offer one promising approach to making programs easier to write and easier to modify and extend. Unfortunately, object-oriented language constructs often incur a substantial run-time cost. To improve the run-time performance of object-oriented languages, previous research applied intraprocedural class analysis and compile-time class prediction to determine the actual classes of arguments to a dynamically bound call at compile-time, thereby statically binding the call to the target procedure and enabling inline expansion to eliminate the overhead of the call. This work was performed primarily in the context of purely object-oriented languages. The goal of this project is to extend previous work in three directions. First, the scope of analysis will be expanded to the whole application program. This necessitates developing and evaluating techniques for constructing call graphs for object-oriented languages and performing interprocedural class analysis. Second, profile information will be incorporated into the analysis and optimization process. In particular, profile information can provide much more accurate class prediction information than can static estimation techniques. Third, procedure specialization will be selectively applied to the optimization of object-oriented languages. The goal will be to investigate the relative trade-offs in cost and benefit among inline expansion, procedure specialization, and interprocedural analysis for object-oriented languages. The suitability of these techniques across a range of object-oriented programming languages, both pure and hybrid, will be evaluated in this effort.

University of North Carolina – Chapel Hill; Siddhartha Chatterjee; *CAREER: Automatic Data and Computation Partitioning for Array-Parallel Languages*; (CCR-9501979 A000); $134,725; 36 months.

This project will develop a set of tools that will automatically map data and computation for HPF programs, thereby reducing the programming time and effort required for producing efficient and portable parallel programs. The system of tools will use the alignment-distribution graph as the internal representation of Fortran 90 (and HPF) programs and use graph-theoretic, linear- algebraic and discrete optimization techniques to determine alignment and distribution of arrays that minimize program completion time. The system will not be limited to the "owner- computes" rule often employed by HPF compilers, but will choose the computation rules that minimize running time of the program. The system will also provide a back-end for generating code for message passing systems, and a graphical user interface for debugging and visual feedback. The project will develop algorithms for inter-procedural layout analysis, and heuristics for reducing the complexity of analysis algorithms. The educational component of this project will develop undergraduate courses with laboratory emphasizing experiment design, measurement and use of statistical analysis.

Brandeis University; Jacques Cohen and Timothy Hickey; *Parallel Processors for Constraint Languages*; (CCR-9403427 A001); $63,936; 12 months.

The focus of this project is the development of compilers and interpreters for Logic and Constraint Programming (CLP) languages executing on sequential and parallel machines. Prolog is an example of such a language, and it offers the users capabilities of solving problems in domains including reals, booleans and strings. The four specific topics of the project are: (1) Constraint Parallelism. Constraint solvers for several domains (including integers, reals and booleans) are designed, implemented and analyzed on parallel machines. These solvers are used to prototype constraint parallel Prolog systems; (2) Program Analysis and Transformations. Tools for the analysis of parallel constraint and logic programs are developed. Some of the techniques that are considered are partial evaluation and abstract interpretation based global analysis; (3) Extension of CLP. High-level, C-like languages based on constraints and interval arithmetic are designed, implemented and analyzed. The possibility of integrating, functional, set-theoretic and CLP languages in a single higher order logic programming framework is also studied; and (4) Applications of Parallel CLP. Several classes of parallel CLP applications are considered, including genome parsing, combinatorial search problems (such as scheduling), and optimization of non-linear constraint systems.

University of Arizona; Saumya K Debray; ***Scalable Parallel Implementations of Declarative Programming Languages*****; (CCR-9502826 A000); $57,638; 12 months.**

This project will develop compilation and implementation techniques for declarative programming languages on distributed memory architectures. The project consists of three parts. First, it will develop compile- time techniques for cost analysis of message- passing implementations of declarative languages. Second, it will develop and implement algorithms for granularity control and task partitioning of languages with fine- grained implicit concurrency on distributed memory systems. Finally, the project will develop compile- time and runtime techniques for efficient execution of declarative languages in the context of multi-level memory hierarchies.

University of Wisconsin – Madison; Charles N Fischer; ***Advances in Register Allocation and Efficient Run-Time Program Analysis*****; (CCR-9505922 A000); $179,952; 36 months.**

This project deals with register optimization techniques for modern computer architectures. In modern pipelined architectures code is carefully scheduled to avoid unnecessary delays. However, some delay slots may be unavoidable. This project will develop techniques to "glean" unfilled delay slots to perform work normally considered not worthwhile, such as splitting live ranges into shorter ranges. Shorter ranges induce fewer register conflicts, allowing registers to be used for other purposes. Normally,registers are not used across procedure boundaries. The project will develop techniques to optimally allocate registers at the interprocedural level. More specifically, the techniques will attempt to avoid unnecessary register spilling on caller/callee boundaries. The project will also investigate the use of idle processors to perform a variety of program correctness analyses. This is achieved using a "shadow" process which executes only when a processor becomes idle.

Carnegie Mellon University; Robert W Harper; ***CAREER: Type Theory and Operational Semantics for Programming Languages*****; (CCR-9502674 A000); $35,000; 12 months.**

This CAREER award supports an investigation into the use of type theory and operational semantics in the design and implementation of programming languages. Type theory has proved to be an important organizing principle in programming language design. This is well exemplified by the use of type theory in the design of modularity and abstraction constructs. This investigation seeks to consolidate advances in type theory into the design of the ML2000 programming language. Type theory has proved important for the implementation of programming languages. Conventional syntax-directed compilation methods are generalized to type-directed compilation, allowing type information to be exploited during compilation. Sophisticated type systems such as those derived from the Girard- Reynolds polymorphic lambda-calculus suggest new implementation strategies based on passing type information at link- and run- time. This research investigates the use of type-based compilation techniques. Operational semantics provides a suitable framework for addressing issues of compiler correctness and proving properties of programs. Typed programming languages such as Standard ML provide a rich setting in which to discuss high-level programming techniques such as data abstraction, modularity, and separate compilation. Types are essential for reasoning about programs. A procedure can in general be deemed to have certain input/output properties only under the assumption that its arguments have suitable types. Operational semantics is a useful pedagogical tool for teaching undergraduate programming, both as an explanatory device and as the basis for proving properties of programs and languages. Important advantages are expected through the interplay between research and education.

Brown University; Pascal Van Hentenryck; ***NSF Young Investigator (NYI) Award: Constraint Programming Languages*****; (CCR-9357704 A002); $62,500; 12 months.**

Constraints are ubiquitous in many application areas including operations research (packing and scheduling), hardware design (circuit verification), computational linguistics, computer graphics, and software engineering. Constraint programming languages hold the promise of reducing the development time of these applications significantly, while preserving a reasonable efficiency. The purpose of this project to study the design, implementation, and applications of constraint programming languages. Both applicative and object-oriented languages are considered. Design issues focus on how best to support established paradigms for the applications considered. Implementation research considers the design of algorithms for constraint solving, entailment, and generalization for a variety of constraint systems. It also investigates the static analysis of constraint programming languages to reduce the performance gap between procedural and constraint-based languages, leading to the development of optimizing compilers for a variety of languages. Finally, applications validate the technology and implementation developed in the project and demonstrate the potential impact of the technology.

Loyola University of Chicago; Radhakrishnan Jagadeesan; *CAREER PROGRAM ATDCC: A Language for Communicating Reactive Process*; (CCR-9501943 A000); $34,430; 12 months.

The focus of this CAREER research proposal is the design, semantics, and implementation of ATDCC, a language for a network of asynchronously communicating reactive programs. Reactive systems are tightly coupled, event-driven systems that react continuously with their environment at a rate controlled by the environment. Asynchronous systems are those in which processes are loosely coupled and communication can take arbitrary amount of time. Complex applications, arising commonly in practice, typically require the combination of both these ideas, thus motivating ATDCC. ATDCC is expected to have the following advantages: (1) support for hierarchical construction of programs/specifications; (2) good foundational properties; (3) efficient implementation as a network of asynchronously communicating automata; (4) verification of safety properties, including real time constraints. This research is aims towards the creation of a complete programming environment for modeling and analyzing complex electro-mechanical systems. This CAREER project includes active involvement in education, in conjunction with the research program. The research program itself involves the participation of students, who will benefit from the exposure to industrial experience. The education plan includes course and curriculum development, for both graduate and undergraduate students, incorporating current research in the general area of interest in the project. An undergraduate parallel programming laboratory is planned to collaborate, disseminate and share resources with institutions in the Chicago area. The laboratory and curriculum development are intended to lead to undergraduate participation in research.

University of Illinois – Urbana; Samuel N Kamin and Uday S Reddy; *Functional Programming and Scientific Computing*; (CCR-9303043 A002); $121,282; 12 months. (Joint support with the Operating Systems and Systems Software Program)

The goal of this research is the design, implementation, and realistic application of a functional programming language supporting the development of scientific codes. Functional languages and scientific computing appear to be singularly mismatched. The problem is not with the expressiveness of these languages, which is excellent, but their efficiency, which is poor. Recent research in functional languages suggests that assignment may in fact be consistent with referential transparency, if care is taken in the overall language design. This research uses one such language, the Imperative Lambda Calculus (ILC), to develop programs in the domain of random process modelling. Although inefficiency on sequential processors has long been a drawback of functional languages, their efficiency on parallel processors has been a strong attraction. An important goal of this research is to obtain an efficient parallel implementation of ILC. The potential for success in this endeavor is high because the presence of assignment in ILC obviates the transformation to imperative form, leaving the compiler to handle only the (well understood) problems of higher-order functions and lazy evaluation themselves. To fully exploit modern parallel architectures, especially non-uniform memory access machines, control over the placement of data and scheduling of tasks needs to be provided by way of parallel language extensions to ILC.

Boston University; Assaf J Kfoury, Pawel Urzyczyn, and Jerzy Tiuryn; *Combinatorial Problems in Typed Lambda-Calculi*; (CCR-9417382 A000); $100,000; 12 months. (Joint support with the Theory of Computing Program)

This award supports research in typed lambda-calculi, with special emphasis on the combinatorics, complexity, and logic of type inference. Specific problems concern type- reconstruction, type-checking, and related aspects, motivated by design and implementation issues in programming languages. Particular attention is given to algebraic characterizations of these problems. Such characterizations give rise to new and more general forms of unification, in particular semi-unification, often with applications in other parts of computer science (term-rewriting systems, database theory, computational linguistics). Based on some of the same techniques, research is also conducted in two other problem areas, both with important applications: higher-order matching (for automatic theorem proving) and extensions of ML-style polymorphism (for polymorphically typed programming languages).

University of Pennsylvania; Dale A Miller; *U.S.-France Cooperative Research (INRIA): Structuring of Proof Search in the Logic Programming Paradigm*; (INT-9412553 A000); $18,000; 36 months. (Joint support with the Western Europe Program Program)

This three-year award supports U.S.-France cooperative research in logic programming between Dale A. Miller of the University of Pennsylvania and Pierre Deransart of the French National Institute for Research in Computer Science and Applied Mathematics (INRIA), Rocquencourt, France. The objective of their research is to update and improve Prolog, a programming language designed from logic programming principles. Although Prolog offers many novel programming possibilities, it does not incorporate software principles of modules, types, higher-order programming, and data abstraction. The investigators will use two approaches to address these deficiencies: intuitionistic theory of logic programming and modularity in logic programming. The U.S. investigator and colleagues bring to this collaboration considerable expertise in the development of logic programming languages that encompass modular programming. This is complemented by the research at INRIA-Rocquencourt on modular programming in logic programming and application and theory of the intuitionistic theory of clauses. The project seeks to advance development of a module system for logic programming and enhance understanding of language design with modular constraints.

Stanford University; John C. Mitchell; ***Programming Language Analysis and Design*****; (CCR-9303099 A002); $128,548; 12 months.**

This research is concerned with a framework for understanding syntactic, operational and semantic properties of contemporary programming languages. In addition to extending a descriptive theory of programming based on type theory and denotational semantics, the research contributes to the development of new programming languages. The programming language goals are increased ease and efficiency of programming. This involves both cleaner language design, so that programs are easier for programmers to read, write and maintain, and careful attention to features that make it possible for programming tools to manipulate programs meaningfully. The theoretical aspects of the research are aimed at establishing the semantic and algorithmic basis for the development and implementation of programming tools and environments. A common thread through the research is the use of type systems, both as a mathematical framework for analysis and a syntactic basis for automatic program analysis. Some specific problems are the static type analysis of object-oriented programming languages, aimed toward establishing properties of programs that may be used in compilation and program transformation, type inference algorithms for functional languages that include imperative and control features (exceptions and continuations), and the use of linear logic to obtain more precise static analysis of programs. A pragmatic language design effort is a type and module system that is useful for the design and development of object-oriented programs in existing languages. Some longer-term foundational problems are the model theory of linear logic, operational and denotational methods for reasoning about imperative programs with higher-order and polymorphic constructs, and treatments of concurrency in a higher-order or object-oriented language framework.

Colgate University; Philip S Mulry; ***Research at Undergraduate Institutions (RUI): Categorical Methods in Computation and Semantics*****; (CCR-9506383 A000); $32,870; 12 months.**

This project exploits recent advances in the use of sophisticated mathematical methodologies to address related fundamental questions in the foundations of computation theory and programming language semantics. Many important questions arise out of a desire and need to capture in a semantic setting both minor and major variations in computational strategies. The significance in investigating such variations lies in the fact that they can both provide a means of interpreting increasingly sophisticated programming language constructs, and also isolate fundamental mathematical processes at work that help define and distinguish between different notions of computation. The main focus of the research entails the utilization of new approaches in the use of algebras, monads and comonads, categories, logic and sheaves to provide a systematic and conceptual approach to the interpretation and solution of these questions. Questions to be addressed include: finding an algebraically robust interpretation for partial data types and partial maps; generating an intrinsic characterization of fixed point constructions, including the generation of recursive data types, and isolating the special role of invariance in this process; defining a categorical foundation for semantics in general and describing a synthetic domain theory in particular; and identifying and utilizing in a semantic setting logical properties resident in sheaf-theoretic structures. General areas of investigation include partial data types and computation, fixed point semantics, monadic and comonadic computation, invariance and algebras, and partial sheaf semantics. Technical tools include functorial liftings on categories of algebras, refined partial map classifiers, algebraic uniformity, induction and coinduction principles, natural and dinatural transformations, monadic and comonadic strength, domain theory and extensional PERs.

Carnegie Mellon University; Frank Pfenning and Robert W Harper; ***Design, Implementation, & Application of a Framework for the Formalization of Deductive Systems*****; (CCR-9303383 A002); $130,833; 12 months.**

Formal deductive systems play a central role in the areas of programming languages and logics. First, they are used to define languages and their semantics at a very high-level of abstraction (e.g., type systems or operational semantics). Second, they form the basis for the implementation of algorithms pertaining to languages (e.g., type inference or interpretation).

Third, they provide a common basis for the study of meta-theory of programming languages and logics (e.g., preservation of types under evaluation). Motivated by the tremendous variety of deductive systems of interest in computer science and logic, general meta-languages for their specification have been investigated. These meta-languages are often referred to as logical frameworks. The objective of this effort is to further the theory and practice of logic-independent, computer-assisted formal reasoning and meta-reasoning. This research addresses definitional, operational, and meta-theoretical aspects of logical frameworks comprising work on further design, implementation, and application of such frameworks.

Winston – Salem State University; Atanas A Radenski; ***Research at Undergraduate Institutions (RUI): Parallel Object-Oriented Programming with Multiple Inheritance*****; (CCR-9509223 A000); $36,257; 12 months. (Joint support with the Operating Systems and Systems Software Program)**

The transition from sequential object-oriented programming to parallelism has been the focus of research over the past decade. Experimental languages that try to integrate objects and parallelism often compromise the inheritance capability for parallel objects. Even the languages that permit some parallelism and inheritance support only single-class inheritance. In this project support for multiple-class inheritance for parallel objects will be investigated. The technique employed is based on "separate" methods which generate processes and provide rendezvous type coordination. The project will apply this technique to Oberon language. The research will be performed under the Research for Undergraduate Institutions (RUI) program.

Carnegie Mellon University; John C Reynolds; ***The Design, Definition, and Implementation of Programming Languages*****; (CCR-9409997 A000); $101,154; 12 months.**

This project conducts research on the design, definition, and implementation of programming languages, with emphasis on type structure and on bridging the gap between functional and imperative programming. In studying how to implement the programming language Forsythe, it investigates better methods of type checking for intersection types. Considerable improvement is expected from using a top-down rather than the present bottom- up algorithm. A more flexible approach to the implementation of procedures is also investigated, where a portion of the procedure body is compiled into closed code while the rest of the body is compiled into inline code at each call site. This is expected to substantially alleviate the overhead associated with call by name. The research seeks to unify two of the intractable dichotomies that complicate and divide the study of programming languages. The goal is to unify semantics based on domains, which permits partial and infinite computations, with semantics based on partial equivalence relations and intuitionistic type theory, which is limited to always-terminating computations. Also investigated is the unification of earlier work on the syntactic control of interference in Algol-like languages with the more recent development of low-level functional languages based on linear logic. In particular, this research would connect the concept of passive types with the modal operators of linear logic.

University of California – Santa Barbara; Klaus Schauser; ***Postdoc: Efficient Algorithms for Networks of Workstations*****; (ASC-9504291 A000); $40,200; 24 months. (Joint support with the New Technologies Program)**

The object of this proposal is to study parallel algorithms that run efficiently on modern multiprocessors and, in particular, on networks of workstations (NOWs). This research will use a simple but realistic model of communication, the LogP model. The goal is to use the model and its parameters to predict and improve the performance of algorithms on various architectures. In particular, it will provide a methodology for porting efficient code from one machine(e.g., a low-end NOW) to another (e.g., a high-end NOW/MPP), such that both versions are efficient on their respective architectures. This work will be mostly of an experimental nature, and will be carried out on the Meiko CS-2. Some benefits to this research are : accurately predicting the performance of parallel code on other architectures, efficiently optimizing the ported algorithms, and making the use of low-end NOWs as test platforms much more viable.

Kansas State University; David A Schmidt and Allen Stoughton; ***Analysis and Classification of Programming Languages*****; (CCR-9302962 A002); $73,000; 12 months.**

This project investigates the structure of semantics definitions of programming languages. First, models of the "core" of a programming language are analyzed and classified; next, extensions of the core are analyzed for their harmony (correspondence) and orthogonality, and models of these extensions are related to each other and to models of the core. The ultimate goal is a "design and analysis workbench," in which a language can be synthesized, classified, and prototyped. Such a workbench will aid language designers and provide experimental evidence that languages with harmonious designs have clean and correct implementations.

Yale University; Zhong Shao; ***CAREER: Type-Directed Compilation*****; (CCR-9501624 A000); $32,819; 12 months.**

Compile-time type information is believed to be very valuable in efficient compilation of functional languages such as

Standard ML. But little is known on how to support type-directed compilation in real compilers and how much performance gain one can get from various type-based optimizations. The goal of this research is to design and identify important type-directed compilation techniques that can significantly improve the performance of ML programs. These new compilation techniques are expected to lead to the following nice results: (1) data objects (including recursive data structures) in ML programs will be able to use unboxed data representations that are as efficient as those in C programs; (2) polymorphic functions and parametrized modules will be implemented efficiently through cross-module type specialization and inlining expansion; (3) type information propagated into the compiler back end will be used to assist program analysis such as closure analysis and instruction scheduling. The educational component of this CAREER award seeks two objectives: the construction of a new compiler lab kit (based on the above type-based ML compiler) and the incorporation of state-of-the-art compiler technology into the undergraduate and graduate curriculum.

American University; Geoffrey S Smith; *On-Line Polymorphic Type Inference in Imperative Languages*; (CCR-9414421 A000); $0; 24 months.

Although sophisticated polymorphic type systems have become a standard part of functional programming languages, such type systems have had little impact on imperative languages. This research is aimed at developing a new type discipline for a class of implicitly-typed imperative programming languages with subtyping, coherent global overloading, and polymorphism. Technical problems include preserving principal-typing and subject-reduction properties with polymorphic references, developing an appropriate notion of coherence, and finding algorithms to decide whether a set of subtyping and overloading constraints is satisfiable. A novel aspect of the work is the implementation of an incremental algorithm for on-line type inference, suitable for use in programming environments where partial definitions may be given and updated. The implementation is a structured editor generated automatically by the Synthesizer Generator from an attribute grammar specification. Incremental type recomputation is implicit in this paradigm.

Wesleyan University; Ramesh Subrahmanyam; *Research Initiation Award (RIA): Types, Abstraction and Programming Language Expressiveness*; (CCR-9406202 A001); $32,531; 12 months.

The notion of types is an important concept in both the design of programming languages and in the theoretical investigations of programming languages. While a significant understanding of type theoretic issues in a programming language context exists, fairly basic questions remain open. Similarly, while it has been suggested that sophisticated type systems enhance the expressive power of languages, very little has been done to set-up a formal work to investigate this idea. Two related issues form the core of this work: language expressiveness, and types and operational semantics. A notion of expressiveness for typed programming languages is described. The expressive power of a language relative to another is measured by comparing the sets of abstract data types implementable in the two languages. Preliminary results suggest that the resulting expressiveness hierarchy of programming languages in non-trivial. This research conducts a detailed investigation into the relationship between this notion of expressiveness, and various type disciplines and language mechanisms. Answers to these questions have a definite impact on the design of programming languages. The research investigates some questions about types in an operational semantics context; these are related to questions studied in the context of lambda calculus and type theory. Two specific classes of questions that are addressed follow: (1) orthogonal extension questions: does type system A, when added to a languages based on type system B, invalidate any existing observational equivalences; and (2) type isomorphism questions: given a pair of types in a programming language is there a pair of definable functions (from one to the other) which are observationally inverse. These lead to further questions about axiomatization of observational equivalence in fragments of typed programming languages.

Portland State University; Andrew P Tolmach; *Flexible and Efficient Implementation of Polymorphic Programming Languages*; (CCR-9503383 A000); $119,825; 36 months. (Joint support with the Advanced Scientific Computing Centers Program)

Polymorphism is a key feature of modern, strongly-typed programming languages including Ada, C++, and ML. This research program investigates a new approach to implementing polymorphism, based on associating type information with each activation of a polymorphic function, rather than with data. This approach permits type information to be utilized effectively at runtime or link-time, not just at compile-time. It enables compilers to generate polymorphic object code from parametrically polymorphic source code, while still using specialized, space-efficient data formats that can be shared with other software components. The approach also supports a large variety of powerful non-parametric features in the source language and in the development and runtime environments. The focus of the research will be on generating prototype compiler implementations and language extensions, that can be evaluated using performance benchmarks. The underlying aim of the research is to make polymorphic programs easier to develop and to integrate into existing systems, thereby, hastening the acceptance of modern languages by industrial and commercial users as well as by academics.

Naval Postgraduate School; Dennis M Volpano and Geoffrey S Smith; *On-Line Polymorphic Type Inference in Imperative Languages*; (CCR-9400592 A000); $53,362; 24 months.

Although sophisticated polymorphic type systems have become a standard part of functional programming languages, such type systems have had little impact on imperative languages. This research is aimed at developing a new type discipline for a class of implicitly-typed imperative programming languages with subtyping, coherent global overloading, and polymorphism. Technical problems include preserving principal-typing and subject-reduction properties with polymorphic references, developing an appropriate notion of coherence, and finding algorithms to decide whether a set of subtyping and overloading constraints is satisfiable. A novel aspect of the work is the implementation of an incremental algorithm for on-line type inference, suitable for use in programming environments where partial definitions may be given and updated. The implementation is a structured editor generated automatically by the Synthesizer Generator from an attribute grammar specification. Incremental type recomputation is implicit in this paradigm.

Semantics Theory

University of California – Berkeley; Alexander Aiken; *NSF Young Investigator (NYI) Award: Constraint-Based Program Analysis*; (CCR-9457812 A001); $62,500; 12 months.

In this project the work is focused on the design and implementation of advanced set constraint based program analysis algorithms, and their application to program optimization for a wide variety of programming languages. The basic theory, key algorithms, implementation techniques, and applications of the areas are simultaneously explored. Within the teaching plan, the constraint-based program analysis system is further developed and disseminated in support of the graduate programs. Work at the undergraduate level focuses on preparing students to work in the software industry.

University of Oregon – Eugene; Zena M Ariola; *Research Initiation Award (RIA): Intermediate Languages Based on Graph Rewriting*; (CCR-9410237 A001); $30,000; 12 months.

This research pursues the development of a formal semantics for real implementations of functional languages, modeling sharing and cycles, and for parallel functional languages with state. Traditionally both the operational semantics and the compilation process of functional languages are given by translating them to either the lambda-calculus, combinatory logic or abstract reduction systems. However, these computational models do not capture sharing and direct recursion, and thus are inadequate for reasoning about compiler issues and implementations. This research develops a sound theory for cyclic structures involving lambda abstraction. The need for new mathematical foundations arises when state is introduced in functional languages. Previous theories apply only to languages with sequential semantics. This research seeks to provide a framework for reasoning about state in a parallel setting by extending the above graph calculus.

Cornell University ; Bard Bloom; *Operational and Equational Semantics of Process Algebras*; (CCR-9223183 A003); $87,282; 12 months. (Joint support with the Operating Systems and Systems Software Program)

Process algebras are a basic tool in the theory of concurrency: a mathematical model simple enough to analyze and understand, yet complex enough to illustrate many of the subtleties and problems of concurrency. They have been used for theory, programming language design, and specification and verification languages. For sequential languages, a few models of computation suffice. However, there are a variety of forms of concurrency, and hence a variety of process algebras. It if therefore essential to: (1) understand the connections between the many schools of process algebra, (2) have theoretical and computer tools for designing and analyzing process algebras, and (3) understand commonalties and metatheory of process algebra, to avoid unnecessarily repeating the same work in many settings. A general framework for investigating process algebras in terms of their structured operational semantics (SOS) is formulated and a surprising connection between SOS and equational definitions established. In particular, operations that respect the most commonly-used process equivalences are axiomatizable from their SOS specifications. It is believed that this connection is theoretically fundamental and of substantial practical value to researchers. An elaboration of this paradigm should unify two major communities in process algebra, and provide valuable tools to researchers in concurrency.

Carnegie Mellon University; Stephen D Brookes; *Semantics of Parallel Programs*; (CCR-9412980 A000); $65,000; 12 months.

This project seeks to develop denotational semantics for a variety of programming languages, tailored to support compositional reasoning about various important forms of program behavior. It studies a typical functional programming language, and two typical parallel imperative languages: a language for shared-variable parallelism, and a language of communicating processes related to Hoare's CSP and Milner's CCS. Notions of behavior considered for the parallel languages include safety and liveness properties, and partial and total correctness, assuming various forms of fairness properties that correspond to natural assumptions about the runtime execution of concurrent processes. This should permit reasoning about parallel programs to be carried out without knowledge of or reference to details concerning scheduling. There also is a research focus on intensional properties concerning runtime and efficient use of resources. Technically, the project aims to develop fully abstract semantics, with respect to a variety of important notions of program behavior. A semantics is fully abstract if it gives the same meaning to two program phrases precisely when they induce identical behavior in all program contexts. This provides a rigorous criterion for judging the utility of a semantics in supporting compositional or modular reasoning about program behavior. The long term goal is the development of a unifying framework suitable for establishing and exploiting relationships between languages, models, and proof methods for reasoning about programs. In particular, the research seeks to establish a mathematically tractable theory and use it to develop practical techniques for modular design and analysis of parallel programs.

University of California – Santa Cruz; Allen Van Gelder; *Efficient Extensions of Well-Founded Semantics for Functions, Aggregation and Collection Types*; (CCR-9505036 A000); $80,433; 36 months.

Recent research has demonstrated that the well-founded semantics and its equivalent, the alternating fixpoint semantics, can be implemented efficiently on deductive databases (relational databases with additional function free logical rules, which may introduce intertwined negation and recursion). Logic programming is a natural candidate for a database programming language because it shares the relational paradigm. This project is studying extensions to traditional logical languages that are likely to be useful for database programming, including nested relations and aggregation. Deductive rules involving aggregation as a primitive operation permit a natural and concise expression of several well-known optimization problems, such as shortest paths in a graph. However, implementation of such primitive operations is complicated by the fact they they occur inside recursion; that is, the relation being aggregated within a deductive rule is also being defined in part by that same rule. Greater complications are added if the rules may contain negation. This project is developing algorithms to implement aggregation efficiently within recursion and negation. Collection types are a valuable adjunct to deductive database languages, and are closely related to aggregation, nested relations, and complex objects. A collection type is any data type suitable for representing a finite set. With collection types, a language can manipulate relations (and nested relations) directly. This research is designing a language to integrate collection types with deductive database rules, with sufficient restrictions on the collection-typed objects to make efficient implementation possible. For example, the language excludes unification among partially instantiated sets. Integration of the functional programming paradigm and the relational paradigm of logic programming is also being studied. Data typing plays a central role. In logic programming functional terms are normally uninterpreted, and specific data types are not declared. The main idea is to establish contexts in which functional terms are interpreted as user-defined functions. The objective is to provide a high level language that can manipulate sets and relations in a convenient and understandable way, with sufficient power to solve practical applications.

University of Pennsylvania; Carl Gunter; *Relating Static and Dynamic Semantics of Programs*; (CCR-9415443 A000); $95,599; 12 months. (Joint support with the Software Engineering Program)

A key problem in programming languages is the design and implementation of a balance between static (compile-time) and dynamic (run-time) phases of computation. A good balance allows the generation of good code without an undue burden on programmers or compilers. Finding this balance relies critically on intuition and experimentation, but these methods are not enough by themselves. As type systems and optimizing transformations have become more sophisticated, it has become essential to provide rigorous means to express static/dynamic relationships at suitable levels of abstraction. Programmers and compiler writers rely on these relationships, so a precise but approachable description of them is important. Using intuition and experimentation alone may allow errors or valuable opportunities for generalization to go unnoticed. The goal of this project is to develop and apply techniques for rigorously expressing and proving the relationships between static and dynamic semantics of programming languages. Primary objectives include (i) finding ways to specify meanings at levels of abstraction suitable for reasoning about sharing, copying, and liveness; (ii) developing ways to represent and manipulate semantic specifications in a formal manner; (iii) aiding the development of new interface definition languages by formulating techniques for describing the properties that compatibility of modules guarantees for systems that integrate them.

University of Massachusetts – Amherst; Ernest G Manes; *Modular Semantics with Monads*; (CCR-9406793 A000); $75,000; 36 months.

In principle, if an algorithm is given generalized input (e.g. a probability distribution, a set of values, a list with no repeats, or values coupled with CPU times)it should produce correspondingly general output. The "modular semantics" studied in this project provides formal machinery for a transparent implementation. For example, given code to produce random outcomes in {x1,...,xn}, a simple declaration that the ambient environment is to be probabilistic prior to calling the code is all that is needed to modify the output from a random single output to an expression {{p1,x1},...,{pn,xn}} wherein pi is the probability of the outcome xi. The underlying programming system must be altered so as to replace standard function evaluation with Kleisli evaluation with respect to a finitary monad in the category of sets. Universal algebra allows for a uniform implementation using tree processing. The primary research goal is to understand the process by which code is transparently interpreted relative to a monad. While existing examples demonstrate the viability of the project, most of the work remains to be done. A secondary research goal is to develop creative applications of monad-driven algorithms.

Tulane University; Michael W Mislove; ***The 11th Conference on Mathematical Foundations of Programming Semantics, Tulane University, New Orleans Louisiana*****; (CCR-9503096 A000); $3,400; 6 months.**

This award supported the eleventh meeting in a series concerned with the Mathematical Foundations of Programming Semantics. Research in this area ranges from the design to the implementation of programming languages, focusing on the semantics of such languages, and includes those areas of mathematics that underlie these research areas. Topics ranged from type theory, polymorphism, and models of the lambda calculus to specification, program logics, and proof theories. Previous meetings have been distinguished by the breadth of interests of the participants and by the high degree of interaction that has taken place between researchers representing these diverse areas. The purpose of this award was to assist students and to create wider opportunities for participation in this research community, in particular for women and members of other underrepresented groups.

Johns Hopkins University; Scott F Smith; ***Semantic Tools for Realistic Programming Languages*****; (CCR-9301340 A001); $56,313; 12 months.**

This project addresses the problem of giving full and formal semantics to programming languages that combine a number of complex features, including memories, object-orientedness, rich notions of type, exceptions, and concurrency. An operational approach is taken to semantics since it is the only method with immediate potential for applicability to such rich languages. The resulting operational theory also includes a rich theory of equivalence on programs to aid compiler writers and language transformers. This research focuses on: (1) lifting concepts from domain theory, such as finiteness and omega-algebraicity, into full-featured languages, (2) giving semantic type descriptions of full-featured languages, in particular the difficult case of object-oriented languages, and (3) studying full-featured languages as one piece to discover interactions and conflicts, rather than addressing each feature in a separate study.

State University of New York – Stony Brook; Eugene Stark; ***Dataflow Networks: Calculi, Axiomatics, and Algebraic Semantics*****; (CCR-9320846 A001); $29,996; 12 months.**

Dataflow networks are a paradigm for concurrent computation in which a collection of concurrently and asynchronously executing processes communicate by sending data values over FIFO communication channels. "Determinate" dataflow networks, which are composed of processes with functional input/output behavior, are a well-understood special case, whereas "indeterminate" networks, which can contain nonfunctional processes, still present some puzzles. In particular, the goal of discovering a useful and powerful logic for reasoning about indeterminate networks, has not yet been fully achieved. A step in the direction of such a logic is a formal theory called "dataflow calculus". The syntax of the calculus is designed to reflect in a natural way the mathematical structure that has seemed most important in previous semantic studies of dataflow networks. In previous research a soundness and completeness theorem has been proved for a set of equational axioms for dataflow networks, which shows that these axioms characterize an interesting notion of equivalence for dataflow networks. This research project is concerned with continuing the study of dataflow calculus, and its uses, as a concrete foundation on which to build a more powerful logic of dataflow networks, and as a tool for understanding better the relationship between the various abstract semantic models for dataflow networks.

University of Pennsylvania; Val Tannen; ***Presidential Young Investigator (PYI) Award: Type Disciplines in Programming Language Design*****; (CCR-9057570 A006); $37,500; 12 months.**

The semantics of programs and reasoning about them are strongly influenced by the adoption of rich type disciplines. This research concerns the relationships between type systems on the one hand, and operational semantics, specification logics, correctness proof systems, and denotational semantics on the other hand. Particular attention is given to programming with limited forms of recursion enhanced by rich type disciplines, and to automated reasoning (especially rewriting techniques) about such programs relative to data type specifications. Attention is also given to the problem of matching complex data models, in particular object-oriented database models, with appropriate type systems for data manipulation languages. The integration under a statically enforceable type discipline, and in a uniform and perspicuous semantic framework, of class abstraction, inheritance, polymorphism, set manipulation, and persistence helps to increase the reliability and maintainability of database software.

University of Pennsylvania; Val Tannen and O. Peter Buneman; *Collection Types in Programming Languages and Databases*; (CCR-9216122 A001); $70,000; 12 months. (Joint support with the Database & Expert Systems Program)

Structural recursion provides a unifying paradigm for the forms of manipulation of collection types that are common in programming languages and databases. As evidence for this claim a general characterization is developed for the semantics of programs that exploit structural recursion on lists, bags, and sets. An intimate connection is identified between structural recursion and a range of existing languages that manipulate complex objects. This project conducts an investigation into the general principles of collection types, including heterogeneous collections, the semantics of collections in the presence of divergence, the parallel semantics of collection languages, the equational theories of collection languages as a basis for discovering and validating optimizations, and extended pattern-matching with collection types. It also involves the design and prototype implementation of a language demonstrating most of these ideas.

Northeastern University; Mitchell Wand; *Semantics of Computation*; (CCR-9304144 A002); $115,336; 12 months.

This project conducts research in the semantics of computation, with a focus on the development and verification of semantics-based compilers. The goal of the research is to extend semantics-based compiler technology to encompass more of the workings of real compilers, by showing how conventional run-time language structures are formalized as representations of the data manipulated by formal language specifications, and by showing how compiler optimizations are justified by flow analyses. In addition, the project explores issues in type theory, including its relation to flow analysis and questions of complexity in type inference.

Compilation and Optimization

Utah State University; Vicki H Allan; *Research Planning Grant (RPG): Register Allocation with Instruction Level Parallelism*; (CCR-9424384 A000); $17,995; 12 months.

In this Research Planning Grant (RPG) proposal, the PI will evaluate various approaches to instruction level parallelism in modern architectures. More specifically, the project will investigate variations to existing register allocation approaches to optimize the performance of loops. The project will also evaluate register allocation approaches with software pipelining.

Loyola College in Maryland; David Binkley; *Research at Undergraduate Institutions (RUI): The Feedback Compiler - Useful Information from the Backend of a Compiler*; (CCR-9411861 A001); $4,375; 0 months.

This project develops and evaluates a feedback compiler. Most of the traditional compilers provide feedback to the user only from the front-end of the compiler. The feedback compiler provides feedback both from the front-end and back-end of the compiler. The feedback compiler assists both students and practicing programmers. The student programmers gain a clear understanding about the role of a programmer and the role of a compiler. The practicing programmer benefits from the feedback on the machine specific optimizations performed by the compiler. The feedback also aids in the debugging of complex programs. The feedback compiler helps the programmers learn to write efficient programs on a new computer system.

Michigan Technological University; Steven M Carr; *Research Initiation Award (RIA): Improving Cache Performance in Scientific Applications*; (CCR-9409341 A001, A002); $35,540; 12 months.

This project explores approaches for automatically restructuring code to improve the performance of cache memories for scientific applications. Specifically, the project examines experimentally the following techniques: (1) loop optimization guided by a new performance metric that includes the effects of cache performance and floating-point computation on execution speed, (2) the enhancement of cache line replacement algorithms by using compiler generated knowledge of future reference patterns, and (3) the use of compiler generated cache hit and miss information by the instruction scheduler to hide latency on architectures having non-blocking caches. Each of these methods is implemented in an experimental compiler and evaluated on advanced microprocessors. The success of the project represents a significant step towards achieving fully automatic memory hierarchy management.

University of California – San Diego; Larry Carter and Jeanne Ferrante; *Hierarchical Tiling: A Framework for Multi-level*

***Parallelism and Locality*; (CCR-9504150 A000); $331,276; 36 months. (Joint support with the New Technologies Program Program)**

To achieve maximum benefit from tiling, compiler optimization must exercise more control over data movement and storage assignments than is commonly done. Hierarchical tiling takes responsibility for several phases of compilation and code improvements that are traditionally done separately, such as scalar replacement, register allocation, generating message passing calls, and storage mapping. It uses the mechanisms of explicitly naming and copying data to control the movement of data up and down the memory hierarchy and to exploit all levels of parallelism. Its effectiveness as a systematic framework for hand-crafting highly optimized code has been tested on scientific applications for IBM SP1 system. This project will extend the research in compiler optimizations by investigating the following: (1) Develop a parameterized machine model that captures the architectural information needed to guide hierarchical tiling. (2) Study interactions between tiling at various levels of granularity. (3) Incorporate hierarchical tiling into the SUIF toolset. (4) Extend the work to disk storage, explicit I/O and implicit paging virtual memories. (5) Validate the approach using various programs on different parallel machines.

Syracuse University; Alok Choudhary; *NSF Young Investigator (NYI) Award: Compiler and Runtime Optimization Techniques for Parallel Programming on Distributed Memory Machines*; (CCR-9357840 A002); $62,500; 12 months. (Joint support with the Operating Systems and Systems Software Program)

The objectives of this research are to develop compiler and runtime optimization techniques for parallel programming on homogeneous and heterogeneous high-performance parallel computers. Specifically, this research addresses the following issues: (1) Scalable Parallel I/O-Scalable massively parallel computers must include scalable I/O systems to provide a balance between computing power and I/O. The thrust of this research is to investigate compiler transformation for parallelizing I/O operations, stripmining and prefetching data, overlapping computation and I/O, generate efficient I/O access schedules, develop runtime support that can recognize and use access patterns to perform efficient parallel I/O, and develop language primitives that allow users to specify parallel I/O operations; (2) Automatic Data Distribution for Homogeneous and Heterogeneous Systems-The goals are to develop algorithms to automatically generate data distributions that incorporate global analysis, use realistic cost models, and that allow dynamic alignments, distributions and data replication; and (3) Runtime Support for Dynamic Data Redistribution-Dynamic data redistribution may be necessary due to explicit program constructs, compiler generated distributions, across procedure boundaries, or adaptive nature of a problem. The goals are to develop parallel algorithms (including communication scheduling, index conversion, and space optimization) for runtime support for dynamic data redistribution.

Washington University; Ron K Cytron; *SPAR: Synergistic Program Analysis and Restructuring*; (CCR-9402883 A001, A002); $68,894; 12 months.

This project investigates the introduction of ambitious program analysis and program restructuring techniques into automatic program optimization systems, with the goal of significantly improving the quality of code produced by compilers for modern computer architectures. Although a variety of compiler optimization techniques exist, they are not used to their full extent in production compilers because, the analysis and transformations are computationally intensive. In addition, the information obtained by analyses must be maintained and updated across the various compiler passes and program transformations. The project relies on the previous work on "sparse" data flow frameworks and explores three methods for overcoming the current limitations to using ambitious program analyses in production compilers: (1) decomposing a dense data flow framework into an interconnected set of sparse data flow frameworks, (2) selectively computing and using costly information on behalf of a particular program optimization. and (3) integrating program analysis and transformation.

University of Virginia ; Jack W Davidson; *Flexible, Seamless Optimization for High-Performance Processors*; (CCR-9214904 A002); $73,142; 12 months.

This project investigates the design and implementation of algorithms for generating highly optimized code for current and future processors. Of particular interest are processors used in embedded applications. To fully exploit the power of this type of processor requires compilers that use sophisticated algorithms for performing register allocation, instruction scheduling, global data flow analysis, data dependency analysis, and processor-specific optimizations. This research investigates how these optimizations and others interact, and use this knowledge to develop and implement new optimization algorithms that operate seamlessly and cooperate to produce very high-quality code. The starting point for the research is an existing retargetable optimizer that has been used to build compilers for a wide array of architectures, including both general-purpose architectures and specialized architectures used in signal-processing and real-time systems. Its organization is particularly well-suited to exploring the implementation and evaluation of cooperative optimization algorithms for high-performance processors.

University of Pittsburgh; Rajiv Gupta; *Presidential Young Investigator (PYI) Award: Loop Transformations and Scheduling Strategies for Parallelization Software*; (CCR-9157371 A005); $88,634; 12 months.

A framework for studying the interactions among loop transformation and scheduling strategies and a parallelization tool that allows experimentation with alternative parallelization strategies is developed. The tool accepts architectural information to assist in the application of transformations and selection of schedules. The results obtained from studying these interactions are applied in the development of an integrated approach to loop transformation and scheduling. In this approach the loop parallelization process is composed of three major phases. The first phase, the parallelism detection phase, performs dependency analysis to identify loops that can be executed in parallel. In addition, it performs transformations, such as renaming, to convert sequential loops into parallel loops. The second phase, the parallelism scheduling phase, selects an appropriate scheduling strategy. The goal is to choose a strategy that involves minimal run-time scheduling overhead and results in schedules that provide good load balancing and high processor utilization. The final phase, the parallelism enhancement phase, transforms the loops to increase the efficiency with which parallelism is exploited. The transformations that exploit architectural characteristics are applied to reduce synchronization and communication overhead. Any decisions to utilize special hardware features are made in this phase. Decisions of activities performed in the later phases are based upon the decisions made in the earlier phases. An integrated strategy allows convenient application of transformations after schedules have been generated. It also ensures that the scheduling strategy does not undo the effects of transformations that are applied.

Stanford University; Monica S Lam; *NSF Young Investigator (NYI) Award: Language and Compiler Technology for Parallel Machine Organizations*; (CCR-9257105 A003); $62,500; 12 months.

Topics pertinent to parallel system development are examined. Strategies for optimizing interprocessor communication (such as grouping computations which use common data to improve data locality or reduce remote access by the allocation of data to local processor memory) are investigated. Also, tools for simplifying the programming of applications for distributed memory machines are developed.

University of Rochester; Wei Li; *Research Initiation Award (RIA): A Parallelizing Compiler for Distributed Shared Memory Machines*; (CCR-9409120 A001); $28,258; 12 months.

This project develops a parallelizing compiler for distributed shared memory systems, and incorporates a vertically integrated system to manage data and thread locality in parallel programs. The compiler locates threads and data statically whenever sufficient compile time information is available to make near-optimal decisions. When such information is not available, the compiler provides the underlying system with whatever information is available, and lets the underlying system utilize the information. The project investigates the following compile time issues as related to distributed memory systems: (1) automatic parallelism for relaxed models of memory consistency, (2) elimination of false sharing, (3) adaptive coherence protocols, and (4) communication optimizations. Solutions to the issues are implemented within a compiler known as Rochester Optimizing Compiler (ROC). These solutions will be evaluated using standard benchmark programs such as LINPACK, SPEC, Perfect Club, SPLASH and the HPF benchmark suite of Syracuse.

University of Massachusetts – Boston; Joan D Lukas; *Research Initiation Award (RIA): Data Locality Optimization for Data Parallel Extensions to C*; (CCR-9409360 A001); $23,847; 12 months.

In this project, compiler algorithms for the enhancement of data locality on parallel computer systems are developed. These algorithms are implemented in a compiler for a data parallel extension to C targeted to MIMD systems. Enhancements to data locality improve the run time performance of programs by reducing the need for communication and/or synchronization. This work extends compiler design and implementation work on data parallel extensions to FORTRAN (e.g., FORTRAN 90 with extensions) targeted to SIMD systems. Both language extensions and compiler optimizations are investigated. The language extensions are aimed at improving the expressiveness of data parallelism. Generalizations of data locality representations and algorithms to a wider class of architectures and languages are designed and implemented. Automatic techniques for localizing data are investigated with programmer directives.

University of Missouri – Rolla; Lenore M Mullin; *Presidential Young Investigator (PYI) Award: Intermediate Languages for Enhanced Parallel Performance*; (CCR-9350258 A002); $0; 12 months.

The Psi Calculus is taught and used as a mathematical framework for architecture independent enhanced parallel performance for array expressions. Results are applicable in actual compilers for data parallel languages such as High Performance Fortran. Real world practicum trains computer science and engineering students in leading edge research and development technologies.

Ohio University; Santosh Pande; ***Research Initiation Award (RIA): Unified Data and Code Based Program Partitioning on Distributed Memory Systems*****; (CCR-9412407 A000); $84,979; 36 months.**

The objective of this effort is to explore the concept of dominant edges and dominant tasks for scheduling parallel programs on a distributed memory processing system. A dominant edge will play a crucial role in generating a task schedule because of the associated communication cost. A dominant task is a coarse grained task with low run time overhead such that they can be rescheduled on a different processor to achieve a load balance. Thus all remaining (non-dominant) tasks will be allocated statically to processors. This project will build on the earlier work on Threshold Scheduling, where the degree of parallelism is treaded off for the schedule length. In this project, the goal is to investigate the threshold scheduling to enhance unified data and code partitioning for a family of distributed memory architectures including Touchstone Delta and Paragon. This will involve the formalization of feasibility conditions and optimality conditions. A feasibility condition guarantees that the schedule delay of a task from its earliest start time is below the threshold and an optimality condition used a merit function to decide the best task-processor match for a set of tasks competing for a given processor.

Cornell University ; Keshav Pingali; ***Principles and Applications of Restructuring Compiler Technology*****; (CCR-9503199 A000); $240,000; 36 months.**

This project covers the following four activities in the area of restructuring compilers and the application of the technology to applications. (1) The previous work of the PI on data alignment will be extended to include data distribution. This will permit data and computations to be mapped to physical processors of a MIMD machine in such a way that parallelism is exploited while minimizing communication overhead. (2) Loop transformation technology for imperfectly nested loops will be developed. Current technology can handle only "perfectly nested loops" where assignments occur only in the innermost loops. However, many practical applications have imperfectly nested loops. In this project techniques for performing linear loop transformations and tiling imperfectly nested loops will be developed. (3) The project will develop methods for producing parallel code for applications involving sparse matrices. (4) Above techniques will be applied to logic simulation applications.

University of Arizona; Todd A Proebsting; ***CAREER: Efficient Detection of Pipeline Hazards in Superscalar Architectures*****; (CCR-9502397 A000); $117,498; 36 months.**

In this project, a fast simulation environment to aid code generation for pipelined and superscalar machine will be developed. The project will develop a specification language capable of describing superscalar architectures, and extend current mechanisms to permit the detection of data and control hazards, and heuristics for efficient scheduling of instructions on modern superscalar machines. The architectural specifications will be compiled into fast table-driven automata that accurately simulates values moving through pipelines, thus providing adequate information to compilers to correctly predict (and avoid) structural, data and control hazards that adversely affect the performance of the code generated. As a part of the education goals, the PI will develop a compiler laboratory that will emphasize not only the fundamentals of compiler design, but also the importance of the resource utilization to program performance. Interactive tools for learning and better understanding of automatic parsing mechanisms will be developed and adopted for teaching undergraduate level compiler techniques.

University of Maryland – College Park; William Pugh; ***Presidential Young Investigator (PYI) Award: Compilation of Scientific Matrix Programs for Parallel Machines*****; (CCR-9157384 A005); $62,500; 12 months.**

This Software Capitalization award supports the development and implementation of algorithms for manipulating sets of linear constraints over integer variables. The resulting software is used for array data dependence analysis, among other purposes. A preliminary version of these algorithms is used for dependence analysis, and is quite powerful and capable of being used in a production compilers. The software is distributed freely.

Louisiana State University – Baton Rouge; J Ramanujam; ***NSF Young Investigator (NYI) Award: Languages, Compilers, and Runtime Systems for Parallel Architectures*****; (CCR-9457768 A001, A002); $62,500; 12 months.**

The goal of this research is to ease the burden of programming high-performance parallel computers. The project investigates several key issues in this regard: (a) the design of language extensions or new languages for machine-independent parallel programming; (b) algorithms for automatic code and data mapping (both static and dynamic) for scientific programs; (c) code generation, and optimization techniques for locality, inter-processor communication etc.; (d) analysis and optimization of unstructured computations such as those arising in fluid dynamics codes; (e) runtime compilation of irregular sparse matrix codes that exploits both parallelism and data locality; (f) interprocedural analysis and optimization; (g) integration of task and data parallelism; and (h) high-performance input and output.

Michigan Technological University; Philip H Sweany; ***Research Initiation Award (RIA): Global Instruction Scheduling Without Copies*****; (CCR-9308348 A002); $5,000; 0 months.**

Adding instruction-level parallelism to an architecture is a technique becoming popular for increasing power in modern computers. One consequence of this trend is the need to include instruction scheduling in a compiler. "Global Instruction Scheduling Without Copies" investigates a method of global scheduling which, unlike other global scheduling techniques does not suffer from the need for compensation copies. The work's goal is refining a technique for scheduling which will work better than existing methods for ILP architectures with limited parallelism, a characteristic of ILP architectures found today, and expected for superscalar architectures in the future. The study investigates dominator-path scheduling (DPS) in some detail and compare DPS to other global scheduling methods. Since DPS is better suited than other scheduling techniques for the levels of ILP found in current architectures, one outcome of the research is the ability to generate better code for currently popular computers. Additionally, the work quantifies the parallelism available in imperative programs when compensation copies are not allowed. This measurement of such parallelism will influence the design of ILP architectures in the future.

State University of New York – Stony Brook; David S Warren and I. V Ramakrishnan; ***Enhancing the Power and Performance of Tabled Evaluation of Logic Programs*****; (CCR-9404921 A001, A002); $169,660; 12 months. (Joint support with the Knowledge Models and Cognitive Systems Program and the Numeric, Symbolic & Geometric Computation Program)**

This project continues work on the development of XSB system. XSB is a system that integrates logic programming, deductive databases and nonmonotonic reasoning. It uses SLG resolution, a tabled evaluation strategy, to combine Prolog's SLD and deductive databases's bottom up evaluation in a computationally effective way. Current benchmarks show that the XSB system is significantly faster than more traditional approaches to deductive database query evaluation, and shows promise of being very efficient for certain types of nonmonotonic reasoning. In the current project, the XSB implementation will be improved by; (1) improving the underlying WAM-based engine to make it more efficient, and (2) developing program transformation techniques to permit XSB programs to be optimized at the source level.

California State University – Los Angeles; Nancy J Warter-Perez; ***NSF Young Investigator (NYI) Award: Aggressive Scheduling for Instructor-Level Processors*****; (CCR-9457395 A001); $62,500; 12 months.**

In this project, aggressive instruction scheduling techniques for instruction-level parallel (ILP) processors are investigated. ILP processors, such as superscalar and VLIW processors, are used effectively in both general purpose and scientific computer systems. For general purpose systems, the high frequency of branch instructions creates the need for aggressive global scheduling techniques that support speculatively and conditionally executed instructions. For scientific systems, cyclic scheduling techniques such as software pipelining have been developed to aggressively schedule inner loops. The four areas that are explored in this project are: (1) Investigate issues associated with the software pipelining technique and Modulo scheduling. These include scheduling while loops, handling loops with cross- iteration dependencies and removing the uniform initiation interval restriction; (2) Formalize an operation-level intermediate program representation that allows the application of well-known local scheduling techniques to achieve globally scheduled code. This approach has been successfully applied to Modulo Scheduling and is extended to acyclic scheduling; (3) Develop a task granularity adjustment methodology, similar in concept to vectorization, for software pipelining in a superscalar/VLIW based multiple processor system; and (4) Analyze the control architectures for superscalar implementations of existing processors by studying the hardware and compiler design complexities associated with scheduling multiway branching and partial predication architectures.

University of Texas – Austin; Paul R Wilson; ***Research Initiation Award (RIA): Automatic Storage Management for Real-Time, Near-real-time, and Distributed Systems*****; (CCR-9410026 A001); $31,696; 12 months.**

This project explores techniques for automatic storage reclamation, especially for real-time and distributed systems. It compares techniques for incremental garbage collection in terms of efficiency, real-time guarantees, and performance for distributed data. The focus is on techniques that support high-performance on standard hardware, and standard compiler optimizations. The three major phases of the project are; (1) implementation and evaluation of a new real-time uniprocessor garbage collection algorithm which provides better real-time guarantees than incremental copying due to weaker coupling between the application and the garbage collector, (2) extensions of the algorithm for near real-time systems where some real-time guarantees can be sacrificed for higher performance, and (3) application of these incremental techniques as local garbage collectors in a distributed garbage collection system, where they will enable higher performance garbage collection of poorly distributed data.

Pennsylvania State University – University Park; William Winsborough; ***Program Analysis and Compilation Tech-***

***niques for Logic Programming and Concurrent Logic Languages*; (CCR-9210975 A002); $5,000; 0 months.**

Dataflow analysis techniques which lead to better compilation comprise the focus of this project. These are applied to high performance Prolog as well as concurrent logic languages and wide-spectrum languages. First techniques for recognizing dead data structures are verified and prototyped and then compilation methods which utilize them are developed. Then these as well as techniques for detecting single-consumer data structures and deadlock are applied to concurrent logic languages and wide-spectrum languages. A theoretical methodology for power comparison is then applied.

University of Colorado – Boulder; Benjamin G Zorn; *Improving the Performance of Dynamic Storage Allocation Using Behavior Prediction*; (CCR-9404669 A001); $60,141; 12 months.

This project develops optimizations based on object behavior prediction to improve the performance of manual and automatic storage reclamation algorithms. Programs written in languages such as C, C++, or Fortran that use manual garbage collection, and program written in language such as Common Lisp, ML, Smalltalk, or Modula-3 which uses automatic storage reclamation benefit from the techniques developed in this project. The project uses profile-based optimizations to predict properties of allocated objects at the time they are allocated. The properties include object lifetime and object reference behavior. By correctly predicting these properties, the space usage, CPU overhead, and reference locality of programs using either manual or automatic storage reclamation is improved. The project first investigate empirical properties of program in languages using manual and automatic storage reclamation and determines if behavior prediction is feasible. Then the project develops and evaluates alternative algorithms that exploit observed predictable behaviors. Finally, the project prototypes the most promising algorithms and measures their performance.

Automated Program Generation and Verification

University of California – Berkeley; Alexander Aiken; *Program Analysis: Logics, Algorithms, and Application*; (CCR-9416973 A000); $33,492; 12 months. (Joint support with the Software Engineering Program, the Computer Systems Architecture Program, and the Theory of Computing Program)

This project is undertaking a systematic investigation of a general class of structural program analysis problems. The term structural program analysis refers to compile-time program analyses that deal with the types and structure of values computed by programs. This class of program analyses is broad, including within it the classical dataflow analyses of C and FORTRAN compilers, type inference for languages such as ML, and analysis of class membership of values in object-oriented languages. The primary goal of this work is the synthesis of set constraint techniques for structural program analysis in a common framework. This career research pursues an implementation of set constraints suitable for use in a general class of program analysis problems will be developed and distributed for use in research and teaching. Set-based analysis provides a high-level characterization of a class of program analyses in terms of set constraints. Currently, however, there is no general-purpose implementation of set constraints available for experimentation with set-based (or any other class of) analyses. This system is expected to be expressive enough to encompass all set-based analyses and many analyses that are not set-based. Aside from the primary goal of developing an "off-the-shelf" implementation of set constraints that can be used in many settings, the project also pursues three secondary goals. The first goal is to develop a common technical language for structural analyses. The second goal is to describe the structure of the design space for structural program analyses using this common technical framework. The tradeoffs between accuracy and speed in structural analyses are poorly understood. By studying these problems in an abstract setting, it should be possible to make general statements about the relative efficiency and precision of various program analysis techniques that apply across a wide spectrum of applications. The third goal is to design algorithms using the above results. General algorithms are sought that would provide the best precision-speed tradeoffs for various classes of analysis problems.

Carnegie Mellon University; Randal E Bryant; *1995 Daghstuhl Workshop on Binary Decision Diagrams, Germany*; (MIP-9503339 A000); $15,000; 6 months. (Joint support with the Design, Tools and Test Program Program)

Binary Decision Diagrams (BDD's) have found widespread use in synthesis, formal verification and testing of digital circuits. This success in the EDA research area has spawned research efforts on a number of fronts, including theoretical studies of algorithms and complexity, applications to such areas as artificial intelligence and logic programming, and extensions beyond Boolean functions to represent matrices, Markov systems, and multi-variate polynomials. This workshop (February 13-17, 1995) was the first ever held specifically on this important technology.

Cornell University ; Robert L Constable; *Exploring New Constructs in Computational Type Theory*; (CCR-9423687 A000);

$77,500; 12 months.

Computer science has become increasingly concerned with the concept of type. From the 60's onward this notion has demonstrated its value as an effective organizing concept–first in programming and programming languages, then in knowledge representation, in automated reasoning, in formal methods, and now perhaps in computer algebra systems. Previous NSF-supported research has led to a coherent family of constructive type theories and to the implementation of them in the Nuprl system from Cornell and related systems elsewhere. The plan of the investigation is to find the right concepts to explain the use of types in computer algebra systems and in formalizations of abstract computational mathematics (especially algebra). It is conjectured that in some cases the right concepts are generalizations of algebraic data types to higher-order ADT's. In other cases, new type constructors may be necessary, such as the very dependent types. These concepts are expected to clarify both pragmatic issues and foundational questions. Moreover, the method of investigation seeks to formalize the object oriented features of algebra systems and generalize them to other programming environments. The result of this research enables the design of a new generation of computer system that integrates subsystems (such as compilers, concrete and symbolic evaluators, transformation systems, provers, and data base managers) around a rich notion of type used to specify and verify the linking protocols. Investigation of computational type theory also deepens understanding of the foundations of computer science and its links to mathematics and computational science.

Pennsylvania State University – University Park; John J Hannan; *CAREER: Specification and Verification of Compiler Algorithms*; (CCR-9502356 A000, A001); $40,000; 12 months.

The research goal of this CAREER project is the formal specification and verification of algorithms that support the implementation of programming languages. Such formal systems can aid the development of new languages and efficient implementation of these languages. This support arises from both the verification of algorithms, which increases confidence in the correctness of a compiler, and from the improved understanding of compilation techniques allowed by formal specification. Recent work has focused on using logical frameworks and mechanized proof assistants to specify deductive systems representing aspects of compilation and to verify the correctness of these specifications. This research project extends these results by considering advanced, state-of-the-art algorithms found in current compilers, formalizing theories for these algorithms, and verifying the correctness of these algorithms. One specific problem area to be addressed involves closure conversion algorithms and the concept of "safe-for-space complexity." An important aspect of this project is extending the use of mechanized proof assistants to support the verification of complex flow analyses and optimizations of programs. The goal of the education plan is to provide both a general, but rigorous, introduction to the study of programming languages at the undergraduate level and would introduce, at both the undergraduate and graduate level, the recently emerging field of logic and computation which studies the relationship between constructive logics and typed lambda calculi. Although this is typically considered advanced material and not suitable for undergraduates, it can be made accessible to students.

Naval Postgraduate School; L Luqi; *Presidential Young Investigator (PYI) Award: Computer-Aided Prototyping of Real-Time Systems*; (CCR-9058453 A005); $37,500; 6 months.

Rapid prototyping is the process of quickly building and evaluating a series of prototypes of a proposed system. A computer-aided prototyping system is a set of integrated tools supporting an iterative prototyping process characterized by exploratory design and extensive prototype evolution. The goal of this research is to investigate rapid prototyping of hard real-time systems via a computer-aided prototyping system, which is based on a prototyping language supporting the modeling of real-time systems and the reuse of software components. The research focuses on automated methods for retrieving, adapting, and combining reusable components; establishing feasibility of real-time constraints via scheduling algorithms; simulating unavailable components via algebraic specifications; generating translators and real-time schedules; constructing a prototyping project database; providing design completion and error checking facilities; and establishing a graphical interface for design and debugging.