This research project was to explore and develop the technologies necessary to build high performance computers from low-cost VLSI-based microprocessors. The project embraced a wide set of technologies for achieving these goals;  
- Novel parallel computer architectures and associated parallel software  
- High performance microprocessor design  
- High speed CMOS and BiCMOS design  
- Development of key supporting technologies such as testing and CAD technologies.
Microsupercomputers:
Design and Implementation

Final Technical Report

Defense Advanced Research Projects Agency

Contract Number: N00014-87-K-0828

For the period of August 1987 - August 1991

Stanford University
Computer Systems Laboratory
Final Technical Report

Principal Investigator
Dr. John L. Hennessy

Associate Investigator
Dr. Mark A. Horowitz

August 1987 - August 1991

Contract No. N00014-87-K-0828

Order No. 1133, R & T Project Code: 4331685

This work is supported under the auspices of the Defense Advanced Research Projects Agency and the Office of Naval Research.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.
Overview
The goal of this research project was to explore and develop the technologies necessary to build high performance computers from low-cost VLSI-based microprocessors. The project embraced a wide set of technologies for achieving these goals, including:

• Novel parallel computer architectures and associated parallel software.

• High performance microprocessor design, including architecture, high speed component design (such as floating point units and caches), and novel compiler technologies.

• High speed CMOS and BiCMOS design.

• Development of key supporting technologies for designing high performance computers such as testing and CAD technologies.
Parallel Architecture

DASH-Directory Architecture for Shared Memory
The Stanford DASH multiprocessor advances the state of parallel computing by combining the programmability of shared-memory machines with the scalability of distributed-memory machines. The key idea on which DASH is built is that of distributed directory-based cache coherence; caches of the processors are kept coherent by sending point-to-point messages between processors on a scalable interconnection network.

We designed and developed the DASH prototype and successfully demonstrated a 16 processor version using MIPS R3000/R3010 processors delivering up to 400 MIPS of processing power. The interconnection network used consists of a pair of meshes, each with 16-bit wide channels using wormhole routing and derived from the Caltech routing chip.

DASH successfully demonstrates that scalable cache coherent shared memory architectures are possible. A large number of papers have been written about DASH and are cited at the end of this report. There is preliminary interest in industry to commercialize the DASH ideas.

Basic Parallel Architecture Studies
We completed several studies of invalidation patterns in multiprocessors. Our studies show that the best invalidation behavior is achieved when the cache line matches the size of the data objects being shared. Both line sizes that are too small and line sizes that are too large can drive up the average number of invalidations per shared write. We note, however, that this effect is not very strong. Consequently, it appears that high performance multiprocessors should use a large cache line size (32 - 64 bytes) to benefit from pre-fetching effects and to hide network latency encountered in scalable shared-memory multiprocessors.

Simulation Tools for Parallel Machines
We completed two novel systems for simulating multiprocessors. Tango is a software tracing and simulation system that provides data to aid in evaluating parallel programs and multiprocessor systems. The system provides a simulated multiprocessor environment by multiplexing application processes onto a single processor. Tango allows the user to trace the shared memory and synchronization behavior of parallel programs. The system is efficient, and can be used with a wide range of machine and programming models. The Tango system currently runs on the MIPS M120 and on the DECStation 3100. It is about 100-1000 times faster than older trace generators, and thus qualitatively changes the kinds of studies that we can do. In general, accurate tracing is difficult since parallel programs are typically non-deterministic; the execution path through the program depends on the real time behavior of the hardware system. Tango offers accurate tracing by allowing the user to optionally integrate his shared memory and synchronization timing simulations into the tracing environment. We currently have a detailed simulator of the DASH prototype integrated with Tango. This system is being extensively used for studying architectural tradeoffs and also for verifying our directory-based coherence protocol. A new system, called TangoLite, which is significantly faster was subsequently developed. Both Tango and TangoLite, as well as large amounts of program data were distributed to the parallel processing research community.
Parallel Software

The DASH Operating System
We developed an operating system for DASH and successfully booted it on the 16 processor DASH prototype. The OS provides users with the basic abstractions of UNIX System V with added functionality to make use of the new resources available in DASH. Our additions include the ability to attach processes to CPUs and clusters and the mechanisms to access the various novel locking and prefetching primitives provided by DASH. We have also extended the I/O subsystem to allow concurrent disk I/O.

Performance Debuggers for Parallel Programs
MTOOL is a tool for performance debugging of parallel programs. Case studies show that the information provided by MTOOL is extremely helpful: each of the scientific computing group's programs we have examined were sped up by 50-400% after one to two hours of work with MTOOL. A version of MTOOL was distributed for Silicon Graphics systems. A version of MTOOL was also used by Sun to develop a performance debugger for their machines.

Compiler Technology
We made major contributions to the development of compiler algorithm for loop-level optimizations to improve data locality in programs. These optimizations include loop interchange, reversal, skewing, and subblocking. Our technique is unique in that these optimizations are unified into a general transformation, thus we can find the best optimizations without trying every transformation sequence. This new approach to loop transformations can also be used in vectorizing and concurrentizing compilers, and industry has adopted our analysis technique.

We also developed the first compiler system to integrate both high-level parallelizing transformations and scalar optimizations in a common framework. Existing parallelizing compilers use two separate programs: a source-to-source compiler performs high-level code restructuring, and a conventional compiler that translates source code into object code. The integration of parallelization and scalar optimizations is important because scalar information can be used for parallelization and vice versa. Many of the standard scalar optimizations such as constant propagation and forward substitution are useful for parallelism detection. Conversely, if we want to exploit instruction level parallelism, this high-level data dependence information must be made available to the machine code scheduler. Moreover, a clean, simple internal representation is more conducive to high level code transformations than the source level representation.

One of the key issues in integrating the parallelizing and scalar optimizations is the intermediate code representation. The former set of optimizations needs higher level information and the latter needs lower level information. The new intermediate format, called SUIF, captures this ability. We have gained some experience in using the intermediate format by building several rapid prototypes of code transformations.
Parallel Programming Languages

To compensate for the weakness of parallelizing compilers in extracting coarse-grain parallelism, we developed a parallel programming language called Jade. Starting with a sequential program, a programmer simply augments those sections of code to be parallelized with side effect information. The Jade system finds not just static parallelism but also parallelism that can only be derived at run time. Using Jade can significantly reduce the time and effort required to develop a parallel version of an imperative application with serial semantics. We have implemented a prototype Jade system, ported it to DASH, and programmed several applications using this language.
Uniprocessor Architecture

MIPS-X: Second Generation RISC
A major accomplishment during this contract was the completion and evaluation of MIPS-X, our second generation RISC architecture. In addition to reaching new performance levels, MIPS-X incorporates several novel architectural ideas:

- An architectural mechanism that allows integration of high performance coprocessors. Several commercial RISC designs have recently incorporated this approach.

- A novel mechanism for fast branching in pipelined machines, called squashing or canceling branches. Several companies have adopted this idea as well.

- A new technique for combining caches and TLBs to reduce cache access time.

The MIPS-X design was fully documented in a book; this book served as a “design manual” for a number of companies building RISC processors. In addition, the database was licensed to a number of companies that have used the processor core for application-specific processors.

Super-Scalar Design
We did extensive work on techniques for extracting instruction level parallelism in hardware, as well as in integrated hardware/software approaches. Our initial work on the non-scientific applications we are interested in, showed that it is possible to execute a program in about one-half the number of cycles needed by a conventional RISC machine. But, to achieve this type of speedup in hardware required branch prediction, a four wide instruction decoder, out-of-order execution, and register renaming; a non-trivial amount of hardware.

Our integrated approach used some new hardware structures together with sophisticated compiler scheduling technology. Our approach is to change the hardware slightly to allow the software to move code through branches, giving the compiler more opportunities to optimize the code. At present, we have the basic compiler up and running and have a system to estimate the performance advantage of moving code through branches. We also have a proposal for the basic machine organization.

Multi-level Caches
The presence of a second-level cache can decrease the optimum size and cycle time of the first-level cache, and significantly improve performance beyond the best attainable with a single level of caching. The optimal characteristics of the second level-cache depend on the miss ratio of the first-level cache, but in general, an optimal second level cache will be significantly larger and more likely to be associative than if it were alone in the system. This is because the presence of the first-level cache reduces the number of accesses to the second level without significantly reducing the number of misses in that cache. This shifts the tradeoff away from short cycle times and towards low miss ratios.

We performed novel new studies for the design of second-level caches which have been used by industry as they begin to incorporate second-level caches.
High-Speed Arithmetic
We have shown how to build dense, fast multipliers by building a partial multiplier tree, and pipelining the tree after every two carry-save adders. By clocking the structure quickly (100s of MHz) one can complete a multiply only slightly slower than a full tree. We also showed how to perform IEEE rounding in this type of structure. It turns out there is a way of starting the final carry propagate add before the carry-in is known. This result is very important since in iterative multipliers, the carry-in is only known 1 to 2 cycles after the rest of the result has been generated. Using this rounding method allows one to perform IEEE compatible multiplication nearly as fast as producing a truncated result. The overhead in hardware is also not too large, less than 25% added hardware.
High-speed VLSI Design, CAD, and Technology

BiCMOS Memory
During this period, we designed a 64K sRAM in TI's 0.8u BiCMOS technology. The sRAM was designed to demonstrate it is possible to build a large, high-speed (under 4ns) BiCMOS sRAM, while maintaining a reasonable power dissipation (1.5W). It uses the CSEA cell, with a bipolar transistor in each memory cell that we reported earlier. The design uses an innovative address decoder that combines the high-speed of a standard diode decoder, while greatly reducing the power that the decoder requires. The power reduction is accomplished by replacing the resistor load in a diode decoder with a pMOS device, and then using the gate of the pMOS to control its resistance. This technique allowed us to reduce the decoder power by a factor of 4, which reduced the power of the part by about a factor of 3. The sRAM also used a novel write-path. Instead of having a set of CMOS decoders for writes, the RAM uses the read decoders and provides an ECL-CMOS converter per wordline. Sharing the decoder increases the write speed, while reducing the complexity of the part.

The RAM is functional with an access time of 4ns. This BiCMOS memory cell design was used by a commercial company, Aspen, in their 3ns 4K BiCMOS RAM.

We also designed a new BiCMOS CAM cell and built a TLB with a 4ns (pad to pad) translation delay. The 64-entry, fully-associative TLB was simply a demonstration vehicle for the CAM cell which uses small ECL-like swings to achieve new performance levels.

BiCMOS CAD
The last major part of our BiCMOS effort is in creating CAD tools to support BiCMOS designs. We are using Magic for layout, but simulation is a much more difficult problem. Trying to fill the current gap in simulation tools for BiCMOS circuits, we are working on Bisim; a Rsim like simulator for Bipolar and BiCMOS circuits. Like Rsim, Bisim is a switch level simulator. But unlike Rsim, Bisim understands that signals are voltages rather than simple Boolean values. This extra flexibility allows Bisim to handle a wider class of circuits than Rsim can handle. In particular it should be able to correctly simulate a wide class of sense circuits -- circuits that often occur in BiCMOS. At this point, we model both MOS and bipolar devices by piecewise linear devices and have written code that will find the final values for networks of these devices. These algorithms have been incorporated in Bisim, and we are now evaluating their performance by trying to simulate the BiCMOS sRAM we designed. The initial results look promising, but we have a significant amount of work still to do. Although we have simple timing models in this version of the simulator, it seems that these models might need some improving.

Low-Cost Testers
We developed a novel single chip tester. The chip, called Testarossa, contains a dRAM for the test vector storage, a decompressor to increase the effective vector size and the pin electronics for 16 DUT pins. The chip was fabricated in a 1.6m CMOS technology and used to construct a 256 pin, 25 MHz tester.
High Level Synthesis
We had a major breakthrough in optimal logic synthesis of digital synchronous sequential circuits. We have developed algorithms for minimizing the area of synchronous combinational and/or sequential circuits under cycle time constraints and the cycle time under area constraints. Previous approaches attacked this problem by separating the combinational logic from the registers and by applying circuit transformations to the combinational component only. We have shown instead how to optimize concurrently the circuit equations and the register position. This method is novel and achieves results that are at least as good as those obtained by previous methods. A computer implementation of the algorithms in program MINERVA has been accomplished and experimental results have supported the theory. This tool has been widely distributed to industry.

We also developed two tools for high level synthesis: Hercules and Hebe. The former performs behavioral level synthesis into an intermediate form that can be easily simulated. The latter performs user-directed structural synthesis and it embeds the implementation of the algorithms described before. The Hercules/Hebe tools have been used for two chip designs: a digital audio interface chip (CD or DAT to PC) and a discriminator of a multi-anode photodetector for the space telescope.

Protocol Verification
Formal verification of hardware is increasing in importance. We developed and applied new protocol verification techniques to the cache coherence protocol for DASH. We used a reduced description of the protocol which has been sufficient to find some cases that were not covered in the documentation of the protocol. We are continuing to enhance the DASH description and specification, with the goal of either proving it correct or accelerating the debugging of the DASH prototype.

Parallel CAD Tools
Parallel simulation has been proposed to explore the potentials of the DASH multiprocessor machine for CAD applications. Integration of existing simulators (THOR, IRSIM, and SPICE) for mixed-mode circuit and system simulation has been developed to provide a parallel multi-level simulation environment. Parallelism is obtained within each simulator by decomposing its simulation into smaller blocks and among the simulators by providing a communication and synchronization interface between them.

We have also implemented the prototype on a network of DEC workstations and on an Intel iPSC/860 message-passing machine. Initial results show reasonable speed-ups for up to 16 processors.

We also developed a novel parallel place-and-route system called LocusRoute and successfully achieved speed-ups of up to 8.
Power and Noise Analysis
We completed and distributed Ariel, an analysis tool that allows the designer to calculate the noise on the power supply lines in the integrated circuit. This program first extracts the resistance of the supply lines, then estimates the supply current and finally uses this information to find the voltage drops.
3. Publications and Reports


82. Lam, M., "The Software Pipelining Algorithm and Experimental Results." Transactions on Parallel Languages and Systems. 1990.


143. Singh, J.P. and Hennessy, J.L. "Data Locality and Memory System Performance in the Parallel Simulation of Ocean Eddy Currents" in *Second Symposium on High Performance


169. Weise, D. and Seligman, S., *GeneralCompiledElectricalSimulation*. 1989,


