

CEDAR

A Large Scale Multiprocessor

Daniel Gajski, David Kuck, Duncan Lawrie, and Ahmed Sameh

University of Illinois at Urbana-Champaign

The primary goal of the Cedar project is to demonstrate that supercomputers of the future can exhibit general purpose behavior and be easy to use. The Cedar project is based on five key developments which have reached fruition in the past year and taken together offer a comprehensive solution to these problems.

- (1) **The development of VLSI components makes large memories and small, fast processors available at low cost.** Thus, highly parallel (e.g., 1024 processors) systems are not ruled out by cost or physical volume considerations as they have been in the past. Particularly important are the 32-bit one megaflop chips or chip-sets developed in the past year (by HP and TI). Thus, basic hardware building blocks will be available off-the-shelf in the next few years.
- (2) Given the hardware components for a highly parallel system, accessing a parallel shared memory and moving data between memories and processors has been a traditional architectural stumbling block. Many systems have been built that have severe constraints in the memory (e.g., access to columns only) or interconnection network (e.g., nearest neighbors only). **Based on many years of work at Illinois and elsewhere, we now have a shared memory and switch design which will provide high bandwidth over a wide range of computations and applications areas.**
- (3) Compilation for parallel, pipeline, and multiprocessor systems has been another serious traditional problem. **The Parafrase project at Illinois has for more than 10 years been aimed at developing software for restructuring ordinary programs to exploit these supercomputer architectures effectively.** The success of the work has now been demonstrated by achieving high speedups on hypothetical machines as well as on most of the existing commercial supercomputers. It has also been shown that Parafrase can restructure programs to effectively exploit various levels of a memory hierarchy. An important consequence is that a compiler can be used to manage caches in a multiprocessor and thus avoid cache

coherency problems.

- (4) The control of a highly parallel system is another problem of long-standing concern and controversy. It is probably the most controversial of the five topics listed here, mainly because it seems to be the least amenable to rigorous analysis. From an abstract viewpoint, the traditional dataflow approach seems best because control is distributed out to the level of operations on scalar operands. In practice, it seems that dealing with this low level of granularity has many weaknesses. **By using a hierarchy of control, we have found that dataflow principles can be used at a high level (macro-dataflow), thus avoiding some of the problems with traditional dataflow methods.** We have also demonstrated that a compiler can restructure programs written in ordinary programming languages to run well on such a system.
- (5) Algorithms for systems with concurrency have been studied for a number of years. Many successes have been achieved in exploiting the array parallelism of various pipeline and parallel machines. But there have been a number of difficulties as well. It has long been realized that some of these difficulties should be surmountable using a multiprocessor because the parallelism in such a machine is not as rigid as in array-type machines. **Recent work in numerical algorithms seems to indicate great promise in exploiting multiprocessors without the penalty of high synchronization overheads which has proved fatal in some earlier studies.** Furthermore, nonnumerical algorithms have been developed at a rapidly increasing rate in the past few years. These can generally use a multiprocessor more efficiently than a vector machine, particularly in cases where the data is less well structured. Our group has been active in developing both numerical and non-numerical algorithms.

To reach the goal stated in the opening paragraph, we believe that a two phase approach is necessary. The first phase is to demonstrate a working prototype system, complete with software and algorithms. The second phase would include the participation of an industrial partner (one or more) to produce a large scale version of the prototype system called the production system. Thus, the prototype design must include details of scaling the prototype up to a larger, faster production system.

Our goal for the *prototype* is to achieve Cray-1 speeds for programs written in high level languages and automatically restructured via a preprocessing compiler. We would expect to achieve ten to twenty megaflops<sup>1</sup> for a much wider class of computations than can be handled by the Cray-1 or Cyber 205. This assumes a 32-processor prototype where each processor delivers somewhat more than one megaflop.

The *production* system might use processors that deliver 10 megaflops, so a 1024 processor system should realistically deliver (through a compiler) several gigaflops<sup>2</sup> by the late 1980s. Actual speeds might be higher if (as we expect) our ideas scale up to perhaps 4096 processors, if higher speed VLSI processors are available, and if better algorithms and compilers emerge to exploit the system.

---

<sup>1</sup>Million floating point operations per second.

<sup>2</sup>Billion floating point operations per second.

An integral part of the design for the prototype and final system is to allow multiprogramming. Thus, the machine may be subdivided and used to run a number of jobs, with clusters of eight processors, or even a single processor being used for the smallest jobs.

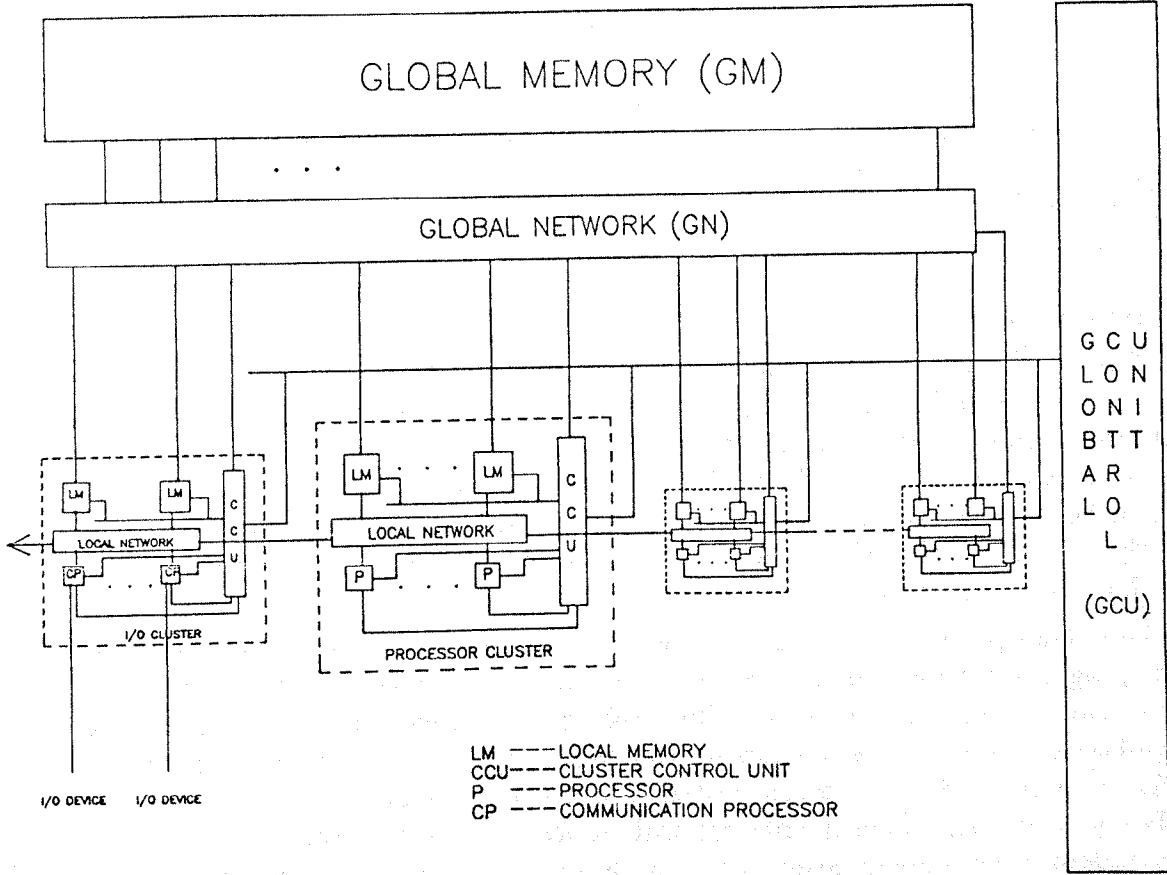


Figure 1. Overall system diagram.

## The Cedar Architecture

### Processor Cluster.

A **Processor Cluster** is the smallest execution unit in the Cedar machine. A chunk of program called a **Compound Function** can be assigned to one or more Processor Clusters. Each processor consists of a floating-point arithmetic unit, integer arithmetic unit, and **Processor Control Unit**, with program memory. The entire Processor Cluster is supervised by the **Cluster Control Unit**, which mostly serves as a synchronization unit that starts all processors when the data is moved from global memory to local memory and signals the Global Control Unit when a Compound Function execution is finished.

### Network Design

Our initial network design is an omega network based on the use of 8x8 switches located on 150 pin boards. Data paths are only six bits wide, and we assume 2.5 ns/gate delay time. Taking into account expected delays due to conflicts, time multiplexing of 120-bit packets, memory access, and return transmission, we estimate an expected delay time of less than 2  $\mu$ s per 1024 words between processor and memory. For a 1024 processor machine, this would be done on 512 boards, or about one-third board per processor. (Using the same techniques but more boards, we can design networks to provide average global memory access in as little as 500 ns, but these designs would require as many as 8 boards per processor.)

### Memory System

The overall memory system has a great deal of structure to it, but the user need not concern himself with anything but the global shared memory. However, the fast local memories present in the design can be used to mask the approximately 2  $\mu$ s access time to global memory. Each cluster of eight processors contains eight local memories (16K of 50 ns access each). A given processor can access its own local memory module directly, or any local memory in its own or its neighboring clusters through the cluster (crossbar switch) networks.

User transparent access to these local memories will be provided in several ways. First, program code can be moved from global to local memories in large blocks by the cluster and global control units. Time required for these transfers will be masked by computation. Second, the optimizing compiler will generate code to cause movement of blocks of certain data between global and local memory. Third, automatic caching hardware (using the local memories) will be available for certain data where the compiler cannot determine *a priori* the details of the access patterns but where freedom from cache coherency problems can be certified by the compiler.

### Global Control Unit.

The Global Control Unit is designed to adapt to the granularity of the data structure. We treat large structures (arrays) as one object. We reduce scheduling overhead

by combining together as many scalar operations as possible, and executing them as one object. In our machine, each Processor Cluster can be considered as an execution unit of a macro-dataflow machine. Each Processor Cluster executes a chunk of the original program called a Compound Function.

From the Global Control Unit point of view, a program is a directed graph called a flow graph. The nodes of this graph are Compound Functions and the arcs define the execution order for the Compound Functions of a program. The graph may have cycles. The nodes in our graph can be divided into two groups: **Computational Functions** and **Control Functions**. All Control Functions are executed in the Global Control Unit, and all the Computational Functions are done by Processor Clusters. All Computational Functions have one predecessor and one successor. Control Functions are used to specify multiple control paths, conditional or unconditional, and can have many predecessors and successors. The compound function graph is executed by the Global Control Unit. When the number of Compound Functions executable in parallel is smaller than the number of Processor Clusters, the Global Control Unit assigns one Compound Function to several Processor Clusters.

### Summary

The Cedar architecture nicely integrates the five key developments sketched in the introduction. We believe that the Cedar system will deliver high performance over a much wider range of applications and algorithms than today's supercomputers can handle. Because the Cedar clock speed is slow relative to such systems, the complexities of building and manufacturing this system are substantially reduced. Due to the ever-decreasing costs of integrated circuits and the relative ease with which the Cedar design can be partitioned, we feel that the monetary cost per megaflop will be much lower than could be achieved by attempting to push today's pipelined supercomputers to higher speeds.