The Cedar System - A Retrospective*

A. Veidenbaum, P.-C. Yew†, D. J. Kuck**, C. D. Polychronopoulos*, D. H. Padua*, K. Gallivan††*

* Authors are with CSRD of the University of Illinois at Urbana-Champaign (** Emeritus)

+ Author is with the Dept. of Computer Science at the University of Minnesota, Minneapolis

++ Author is with the Dept. of Computer Science at Florida State University

1. Projects Goals

The Cedar project was officially started in 1984 following many years of research by our group in parallelizing compilers, parallel algorithms, and vector and multiprocessor architecture. At the time, multiprocessing was not universally accepted as a way to speed up the execution of a single program. The primary goal of the Cedar project was to "demonstrate that the supercomputers of the future can exhibit the general-purpose behavior and be easy to use" [1]. We felt that major advances in the state of hardware technology, architecture, compilers, and parallel algorithms made such a demonstration possible. A two-phase approach was advocated: the construction of a 32-processor prototype followed by a production system with thousands of processors. We stated that "the prototype design must include the details of scaling the prototype up to a larger, faster production system". Both "architectural and technological upward" scalability was required. Another goal was to have the prototype "achieve Cray-1 speeds for programs written in high-level languages and automatically restructured" by a compiler. Finally, "an integral part of the design... was to allow multiprogramming".

1. The quotes are from the [GKLS83] reference in the 1993 ISCA-20 paper.

* This research was supported by the Department of Energy under Grant No. DE-FG02-85ER25001 and by the National Science Foundation under Grants No. US NSF-MIP-8410110 and NSF-MIP-89-20891, an NSF PYI grant NSF-CCR-89-57310, and support from several industrial affiliates including Alliant, IBM, AT&T, DEC, and CDC.

2. General Programming Model

Cedar was to be a scalable shared memory multiprocessor to achieve programmability. To avoid problems with control of a parallel computation and high synchronization overhead that had proved fatal in some of the previous systems, "a hierarchy of control" was to be used and a macro-dataflow model of computation was defined. A program was to be decomposed into a set of tasks, a dataflow graph of the tasks built and directly executed. A ready task in the macro-dataflow graph could be scheduled on one or more clusters.

The architecture was to be constructed as a hierarchy as well and consist of "processor clusters", each with a local memories, switch, and "synchronization unit". The clusters were connected to shared memory and had an ability to overlap cluster computation with shared to local memory moves. The "processor cluster" was to be a basic schedulable unit. The memory was to be organized as a hierarchy and a compiler was to assist in managing it. In particular, a form of software-managed caching of shared memory data in cluster local memory was to be provided.

3. Original Architecture Requirements

The architecture definition of the Cedar system was driven by the above considerations. The system architecture requirements were distilled to the now familiar list below.

- *Shared memory.* To achieve high bandwidth it was to use:
    - (a) an "interleaved" design, separate from cluster memory and with its own network,

- (b) high-bandwidth, low latency interconnect. A multistage design was to be used to avoid latency problems of the earlier, mesh-connected computers.

- Support for multiple levels of program parallelism
- *Efficient synchronization and scheduling support* via a processor in memory
- *Memory hierarchy,* with software-controlled data 'caching' in cluster memory
- Data and code prefetching
- *.Scalability* to a larger number for processors

4. Implementation

Some original implementation ideas were modified in the course of the project in light of schedule and/or hardware constraints. For example, hardware control of macro-dataflow scheduling was shifted to software with efficient hardware synchronization support. Some of the more complex memory-based synchronization primitives were not implemented. Finally, the degree of memory interleaving was reduced by a factor of 2. Even so the resulting board size, packaging complexity, and the need for new surface-mount packaging pushed the limits of technology and delayed project completion.

4.1. Architecture/Hardware

The emergence in 1983-84 of small, high performance multiprocessors, such as Alliant' and Elxsi, allowed us to avoid building our own "clusters" but also limited some of our architectural options. The Cedar team designed and built the Omega networks, shared memory/synchronization processor boards, network interface boards, and performance monitoring hardware. Our hardware linked four Alliant "clusters" into the shared-memory multiprocessor. The resulting system led the introduction of many new architecture and software ideas now commonly found in scalable MPs and even small symmetric MP systems. These are described next, followed by our retrospective view on improving them.

- *Shared memory*. Word-interleaved, UMA shared memory was part of the virtual address space and directly accessible via processor instructions. It was not cached and was physically separate from cached "cluster" memory. Shared memory is now implemented in three out of four U.S.-made scalable MP systems, with message passing architectures in retreat. If allowed to change one thing about our implementation it would be to provide a direct path between shared and cluster memories, which was difficult due to our use of Alliant clusters.
- *Shared memory-based synchronization*. Read-modify-write indivisible operations commonly used in bus-based system are extremely inefficient in large-scale systems due to interconnect latency and contention. Cedar implemented Test-And-Set and other Fetch-and-Op primitives via a fast processor in memory. A 32-bit Fetch-And-Add operation cost two extra clocks over a regular read.
- *Omega network interconnect*. The network used a buffered, self-routing design. Measurements on the actual hardware helped to better understand the effect of multi-word requests, such as vectors or cache lines, and of a processor's limit on the number of outstanding requests on network performance. Simple traffic throttling at the network interface would have been the most valuable addition. This type of network is now used in IBM's SP2 systems.
- *Per-processor strided block prefetch unit and a tagged storage buffer*. This unit, operating independently of its processor could fetch and re-order up to 8KBytes of data. It was limited by lack of address translation. Prefetching is now widely supported by commercial scalable MPs. The most valuable change would have been to prefetch into cluster memory as originally planned and to use multiple prefetch units/buffers.
- *Weak shared memory consistency*. To improve performance, Cedar allowed buffering of reads, writes and prefetches and had out of order memory request completion due to the interconnect. The interface hardware "blocked" the issue of synchroni-

zation operations until all previously issued shared memory requests completed. Relaxed consistency models are now widely used in commercial systems.

- *Use of clusters*. Alliant was chosen for its basic processor performance and similarity in programming model, the latter due to Alliant's founders having been influenced by interaction with our group. An 8-processor Alliant "cluster" efficiently supported vector/parallel computations and we extended parallelism by another level. As a result, Cedar could support three levels of parallelism, memory, and synchronization hierarchy in hardware.

4.2. Software

The Cedar system software supported an explicit parallel API which included nested DOALLs, task management, data placement as private/shared, and synchronization. Many of the API features have now been incorporated in OpenMP, a portable parallel programming standard. Standardization of the interface will undoubtedly help to speed up the acceptance of parallel programming.

A single system image for parallel programs was provided by the Xylem OS kernel running on top of a cluster's Unix system. Xylem implemented global, dynamic task creation and scheduling assigning tasks to available clusters. A single shared memory virtual address space was supported, across all clusters, while private memory in each cluster was individually managed. File systems in each cluster were made available to the user as a single abstraction. The single system image OS for parallel computers is still a research topic and is another area where standardization would help make parallel processing more usable.

Cedar automatic compilation goals were very ambitious and were only partially met during the project. Many of the ideas have since been implemented in other compilers for both uni- and multiprocessors. In particular, automatic detection of parallelism and management of the memory hierarchy made great strides during and since the project. The concept of compiler-based coherence was widely explored and program restructuring to optimize for a cache hierarchy is now widely used.

Macrodataflow ideas of the Cedar project led to many novel scheduling algorithms for arbitrarily nested loops and for general task-graph program models. A more lasting impact came from some of our research work on the efficient and transparent combination of parallel processing and multiprogramming (or equivalently, multithreading and multiprogramming). Results of that work were incorporated in a number of commercial compilers and operating systems, with the latest being the SGI Irix 6.5.

In addition, parallel algorithm design during the project made great strides in understanding and optimizing for a complex parallel/memory hierarchy, with some of the ideas now appearing in compilers and numerical libraries. For example, the BLAS3 were first implemented for the Cedar linear algebra applications.

5. Performance evaluation

At the start of the project, Livermore loops were a standard benchmark used by the community. An important effort spearheaded by the Cedar staff was the creation of a user group to define a standard set of application benchmarks. This led to the "Perfect Club" benchmark suite. The group was later merged with the SPEC consortium and the codes became the first version of the SPEC-HPG benchmarks. Cedar performance was carefully evaluated using these and other benchmark applications/algorithms. It led to better understanding of architecture, compilation, and application programming through hardware and software performance monitoring. The original performance target of the project, Cray-1 performance on a wide range of high-level language programs, was demonstrated. In many cases performance scalability matched that of the later Cray model, 8-processor YMP.

Finally, during the project a set of practical parallelism tests was defined to quantify how well a parallel system performed. In particular, they stressed delivered performance, scalability, and programmability. These tests are still very relevant today and having today's multiprocessors pass the tests remains a great challenge.

6. Conclusion

The multiprocessor design space is very large and has not yet been sufficiently explored. We believe there is a continuing need to study scalable MP systems, especially in light of current and future hardware and software capabilities. Architectural ideas cannot be studied in isolation and require an interdisciplinary team of hardware, architecture, system and application software experts to collaborate. Such a team cannot settle for simulation studies; it needs to develop and experiment with real systems. Testing ideas via implementation, especially their interdisciplinary verification is a key to developing successful architectures. In particular, this avoids the benchmarking syndrome of replacing all application/algorithm experience with simple codes. It also allows the actual concerns, modes of operation, and preferences of one group of experts to be communicated to the others concretely. Finally, 'overlapped' development and research nurture each other and greatly accelerate progress. Most of the above happened on the Cedar project and was both a major achievement and a key to our success.

The potential to speed up program execution by increasing the number of processors continues to hold great promise. This has now been demonstrated by several large production systems on a number of applications. It is also becoming common-place as a cost-effective means of enhancing desk-top performance via parallelism. However, widespread use of parallel programs is still being hampered by programming and performance tuning complexity, architectural bottlenecks, and by their high cost additionally limiting availability of scalable systems. Overcoming these obstacles remains a grand challenge.