PARALLEL PROCESSING AS A LANGUAGE DESIGN PROBLEM

Jim Savage

Myrias Research Corporation

A discussion of approaches towards the design of parallel processors is followed by a description of the language designed by Myrias Research Corporation for parallel processing, Myrias Parallel Fortran (MPF), and of the processor designed to support MPF, the Myrias 4000 system.

Introduction

The major goal behind the drive for parallel processors is to remove the upper bound on processing speeds which is implicit in serial processing. High processing speeds imply a high memory bandwidth. In addition, most of the problems for which high processing speeds are required, such as physical modeling and signal processing, involve large amounts of data, so a large memory with high bandwidth must be provided.

Since software development costs are typically much larger than processor costs for most major installations, it is imperative that a parallel processor be easy to program, have a simple performance model and require minimal effort for conversion of existing serial programs.

To date, there have been two major, distinct approaches towards designing general purpose parallel processors. The first approach, followed by the vast majority of commercial and experimental parallel processor designers, consists of a finite number of relatively fast (usually energy inefficient) processors with global memory semantics. The main advantage of this approach is that it does not require a radically different software architecture. The major design effort is directed towards the hardware architecture, which involves relatively well understood problems.

The first approach has a number of disadvantages, however. Since the design is hardware oriented, users are often handed a piece of hardware for which little or no high level software support is provided, requiring the user to gain intimate knowledge of the architecture in order to achieve good performance. The global memory semantics result in complicated and/or restrictive synchronization semantics such as vector operations, producer consumer variables, Dijkstra semaphores, etc. The amount of parallelism is restricted more than is necessary and the hardware tends to be suited to a small number of classes of algorithms, resulting in a wide performance spread. For example, the Cray XMP1 has peak rates of about 200 Mflops but its harmonic mean for the first fourteen Livermore loops is about 19 Mflops[1], while many important applications run at scalar speeds (3–5 Mflops). In short, hardware restrictions are passed into user software, restricting algorithm development and use because of difficulties in programming or performance. The restriction on uses of different algorithms can be very costly since many more operations may be used to compute a result than are actually required.

The second approach, which Myrias has followed, is to design an expansible architecture, allowing an unbounded number of energy efficient processors to be devoted to one problem. One obvious advantage is that the processing speed is determined by the resources which are allocated to a problem. Also, the opportunity exists to make a simple performance model with a small number of variables, such as the number of processors, and to design a simple programming model without complicated synchronization semantics. Global memory semantics are clearly incompatible with an expansible architecture because of the non–linear growth in cost of the processor–memory interconnection.

The major difficulty behind the second approach is that it requires a radical shift not only in the architecture design but in the design methodology. The traditional design method is to pass a hardware design to system programmers who implement an operating system and then to compiler writers who implement languages as best as possible, given the hardware restrictions, and then to the users. The design methodology used by Myrias consisted of doing a detailed analysis of the target applications, investigating their parallel structures. A simple but expressive language construct was designed to exploit the inherent parallelism of the target applications. The language then drove the system architecture design, with the goal of an efficient, cost–effective implementation of a parallel processor which supports that language construct.

Parallel Language Design → MPF

Any parallel programming language should meet a number of design criteria. The design criteria used by Myrias are the following:

0149-7111/85/0000/0221$01.00 © 1985 IEEE

- The language must imply a simple, intuitive programming model which is close to the cultural expectations of present programmers, scientists, engineers and other users. There should be a simple physical model for the data flow implied by the language.

- The programming model must be independent of the number of processors.

- The language must be very expressive, allowing a natural expression of algorithms used in physical modeling, signal processing, combinatorial problems and other cycle–intensive computing tasks.

- The conversion costs of present serial programs and algorithms should be minimized.

- The parallel construct should be easily grafted onto present serial languages which are used for cycle–intensive problems, such as Fortran and C.

- The language construct should allow an efficient implementation with a low performance spread.

- An implementation of a high level failure recovery mechanism must be enabled by the language. Otherwise the large number of components in a large configuration would limit the MTBF too severely.

The result of the language design done by Myrias is the PAR DO construct and its associated memory semantics. Eliminating global memory semantics eliminates many problems, both in the programming model (synchronization semantics) and in the implementation architecture.

For economic reasons, Fortran is the first language onto which Myrias is grafting the PAR DO construct.

Myrias Parallel Fortran (MPF) was designed to give the user easy access to parallel processing. There are no restrictions on the number or heterogeneity of the parallel processes, and there are no explicit synchronization requirements. Furthermore, by using standard Fortran 77 with some slight extensions, the difficulties of program and algorithm modification encountered with vector machines are avoided.

The following is a short description of the user–level model of MPF. It is not meant to be a precise language definition, nor is any attempt made to demonstrate how the implementation avoids unnecessary work which might be implied by this model.

The principal means of achieving parallelism in MPF is with a language extension, the parallel DO. When the calculations within a DO loop are independent they can be done in parallel by changing the DO keyword to PAR DO. This changes the memory semantics slightly. Each "iteration" now sees the machine state as it was at the beginning of the PAR DO instead of as it was at the end of the previous "iteration". Conceptually, this initial machine state is a parent to many child tasks or loop "iterations". The child tasks may be completely heterogeneous and, conceptually, are done in parallel. Of course, the amount of actual parallelism is restricted by the number of processors available. Scheduling is done by the operating system, not the user.

At the end of a PAR DO, all child tasks are merged into one machine state using the following rules:

- If no task assigns to a variable (or memory location), then the variable is unchanged.

- If one task assigns to a variable, the variable is changed to the assigned value.

- If more than one task assigns to a variable, but the values assigned are identical, then the variable is changed to the assigned value.

- Otherwise, the value of the variable is unpredictable. If several tasks assign different values to a variable, there is no natural way to choose which value it should have after merging.

Note that there is no communication between sibling tasks.

A Eureka! jump occurs when a GO TO inside a PAR DO jumps outside the range of the PAR DO. The program behaves as if the task in which the GO TO is executed is the only task which was executed. There is no merging of memory spaces.

PAR DOs can be combined with recursion. For example, a dot product of two vectors can be done by dividing the two vectors in half, recursing, and summing the resultant dot products. This reduces round–off errors since an operand is involved in only O(log n) additions instead of O(n). Recursion is also the most convenient method for handling combinatorial problems. Parallel recursion enables a limited simulation of nondeterministic calculations to be performed.

There are no restrictions on the number of "iterations" in a PAR DO, nor on the depth of nesting of PAR DOs and recursive subroutines. There is no need to worry about parallel tasks having different amounts of work to perform or different memory requirements. Causal restrictions are handled without requiring any complicated synchronization semantics. The recursive parallel method (RPM) of programming made possible by MPF subsumes all vector, parallel and tree–machine architectures.

Several other features facilitate programming with MPF. Dynamic array allocation eliminates programmed size restrictions and wasted memory. Signed and unsigned infinities as well as control of rounding are provided. Library packages provide the usual vector and matrix operations based on both integer and real arithmetic.

MPF gives the user access to parallel processing in a form which is friendly, intuitive and easy to use. We at MRC are not aware of any calculations which require large numbers of operations and which cannot be programmed in a natural way with Myrias Parallel Fortran. MPF extensions to FORTRAN can be adapted to other conventional computer languages.

The Myrias 4000 System, an MPF Implementation

The firmware to support MPF must meet a number of design criteria, including:

- The architecture must efficiently schedule tasks, support the MPF memory semantics, minimize the data motion costs, and recover from hard and soft failures.
- The implementation must result in a simple performance model for programmers, depending only on a small number of variables such as the number of processors.
- The user must be able to specify the number of processors he wishes to use for his problem, based on the expected performance.
- A common, usable operating system such as UNIX must be supported.
- A multiuser environment must be supported.

The design process resulted in the design of data and management structures required to schedule tasks and support the MPF memory model, taking advantage of the locality of reference in programs in the same manner as other virtual memory machines. These data structures required a hierarchical communication system. The hardware architecture was then designed with the additional constraint that the components used be common, inexpensive and reliable.

The hardware design methodology reflects the general methodology used by Myrias. An array logic language was designed and implemented, allowing the use of software development techniques to design the hardware and giving the advantage of very quick turn-around time through the use of PALs.

The M4000 operating system is totally distributed to eliminate performance bottlenecks. Virtual memory management, process management and resource management are all distributed via a kernel which resides in every processing element of a configuration. Communication is done through messages and page transfers. Pages are cached at different levels of the hardware hierarchy.

The communications system firmware of the multi-level architecture provides support for performance measurement, system failure reporting and basic support for the message and paging systems. The M4000 resource manager collects performance information and adjusts system tuning parameters. The M4000 recovery subsystem collects processor and memory damage reports and initiates appropriate recovery actions.

The hierarchical hardware structure takes advantage of the locality of data references within programs. The basic processing element consists of a Motorola 68000 microprocessor, 128 Kbytes memory and a high speed interface to the board-level bus. Eight processing elements and a service processor are combined on one board. Sixteen boards, a printed circuit backplane (no wirewrap), a service module and a communications board are combined into a cage. Each cage has its own power supply. The communication board has 4 full-duplex 20 Mbaud fiber optic terminations which are used to interconnect the cages.

The Myrias 4000 system supports UNIX, providing a multiuser environment.

The minimal configuration of the Myrias 4000 system consists of 4000 processing elements resulting in a memory of 512 Mbytes and a usable memory bandwidth of 20000 Mbytes/sec. The minimal configuration can be expanded to 64000 processing elements.

The performance model of the Myrias 4000 system requires that a user be conscious of the locality of reference in his program, the degree of parallelism (numbers of parallel tasks), the memory requirements (numbers of processors required because of memory) and the ratio of calculations per processor required for efficient performance.

Because the Myrias 4000 is a virtual memory machine with a distributed cache system. it has locality of reference requirements which are similar to those of other virtual memory machines with caches. If a task has to reference large numbers of variables which have little locality, then system performance can be degraded because of the data motion costs. However, performance is not degraded if there is good locality of reference among related tasks.

The locality of physical processes can be advantageous for parallel processing if the processing elements are capable of large heterogeneous calculations. Thus, in contrast to vector machines, the parallelism of programs should be introduced at the outer contour levels, as well as the innermost contour levels. In other words, the PAR DO construct should be used particularly for the outer reaches of a program.

Putting the PAR DOs on the outermost reaches of a program has a number of benefits. The ratio of calculation to data motion cost is maximized. It is usually very easy to modify present programs. Programs written with the PAR DOs on the outermost reaches are very intuitive and easy to understand.

The memory semantics of MPF often eliminate extra arrays that are used for copying, making programs more natural. However, the size of memory required for a problem does not necessarily decrease since the memory semantics imply that both the old and new values of an array which is being updated must be present until the end of a PAR DO.

A general guide to choosing the number of processors to use for a problem is ((number of calculations per task * number of parallel tasks) / number of processors) $\geq$ 20 single–precision floating point operations. In addition, the memory requirement is (memory required in bytes / (128 Kbytes * number of processors)) $\leq$ 4. The latter requirement may vary somewhat due to differences in memory usage patterns between programs.

References

[1] J. Worlton, "Understanding Supercomputer Benchmarks," *DATAMATION*, pp. 121–130, Sept. 1, 1984.

Historical Footnote

The PAR DO memory model was first presented by Colin Broughton to a Myrias working group which included Chris Thomson, Dan Wilson and the author in November 1982. The working group later formed the core of the Myrias 4000 implementation team.