# Some Thoughts on Parallel Processing

LYNN D. YARBROUGH

*North American Aviation Inc., Inglewood, Calif.*

In the past two years or so I have seen a number of papers and heard a number of talks describing the characteristics of (and the wonders inherent in) certain computers like Gamma 60, LARC, H-800, STRETCH, and others. Of course, all these machines share a capacity for parallel asynchronous multiple processing. Now this is a truly marvelous property, especially from the point of view of the common variety of "my-job's-on-the-machine-keep-your-cotton-pickin'-hands-off" programmer.

However, it appears to me that the application of the multi-program technique, in any manner which approaches the true capacity of the hardware, creates some headaches of imposing magnitude. The most obvious problem is how to prevent one code from mutilating not only itself but any of the other half-dozen or so programs currently sharing the main memory.

Although much has been said and written about the solution of this and other problems (e.g., accounting and scheduling) associated with such multi-processors, I have yet to hear of anyone who has found a reasonable solution to the problem of mutual code-mutilation, or even has much faith that a solution actually exists on a particular computer.

Furthermore, I do not believe that the policy of restricting multi-processing activity to "production" programs is a wholly satisfactory solution. Being an experienced programmer, I know how hard it is to get *all* the bugs out of a large program, and I know of very few large programs that will not run wild when particular (incorrect) data sets are submitted to them. So I say there always exists the spectre, however faint, of even "production" programs killing each other.

Therefore I offer the following hardware scheme, for what it is worth. It does not solve all the problems, but I believe it does move the solutions to the biggest ones within the reach of the programmer.

Assume a system of $n$ processors, operating in parallel and sharing a common fast-access memory with binary address logic.

Allow one processor, called the Monitor (M), access to all of storage. Then allocate storage to the remaining $n-1$ processors (A, B, C, $\cdots$) according to the following scheme: Assign the leftmost $n-1$ bits of the address as allocation selectors so that if the $i$th bit is a 1, the $i$th processor has access to that cell and if that bit is a 0, that cell is not available to the $i$th processor.

For example, a 32-K memory would be used by four processors in the following fashion:

| 4-K Module | Processors using this module |
|---|---|
| 000 | M |
| 001 | M, A |
| 010 | M, B |
| 011 | M, A, B |
| 100 | M, C |
| 101 | M, A, C |
| 110 | M, B, C |
| 111 | M, A, B, C |

Consequences of this scheme would be:

1. One module of storage would be unassailable by processors A, B, C. Similarly, one module accessible to A could not be ruined by programs on processors B and C; and so forth.
2. Each of A, B, C would have access to half of storage.
3. Any of the processors could communicate with any others through the appropriate module.
4. Any priority method (that I have been able to dream up) can be handled easily within this scheme. (But of course I don't have a really objective point of view.)
5. The scheme would, I believe, allow parallel execution of programs which use dynamic storage allocation (e.g., list logic) without resorting to the use of the Monitor as referee, and with a minimum of headaches.

A variant of this scheme would be to allow free use of memory for readout, and use the memory allocation scheme to restrict writing only. This would keep the safety features intact and allow freer communication between processors, but would complicate to some degree the task of storage assignment programs such as Compilers and Assemblers.

There are problems in this scheme (such as address computation involving complements) which I have not even attempted to solve. My hope is that the presentation of this scheme, complete with fallacies, will jar someone into finding a really good solution.