Multimicroprocessor systems

Daniel Prener, in his introduction to the second issue devoted largely to multimicroprocessors, considers the four articles presented here and gives an overall view of the multimicroprocessor articles in this and the previous issue, for which he was guest editor.

Aspinall and Dagless, in 'Overview of a multimicroprocessor development environment' described the CYBA-M multimicroprocessor. The logical and physical architecture are explained in terms of both design motivation and the resulting system. The balance between the requirements of an adequate vehicle for research and the limitations imposed upon the implementation by cost, time and the availability of hardware and software is particularly instructive. The CYBA-M system will enhance the state of the art not only by its existence, but also through its use.

In 'Software for CYBA-M' Dowsing presents the complex of software developed for the CYBA-M multimicroprocessor system. The software provides suitable means of controlling concurrency and parallelism in application programs. It allows development and testing before the hardware is available, and is sufficiently portable so as to be easily bootstrapped onto the object system and remain transportable to future systems.

Pugh offers a description of an application that was implemented on the CYBA-M multimicroprocessor in his 'Colour television graphics simulation'. This case study is doubly interesting in that it not only gives an example that makes good use of parallelism, but it provides a model of good design methodology. Designers of multiprocessor systems who are fortunate enough to have access to a system such as CYBA-M would do well to use Pugh's approach to simulate systems as part of the design process, for certainly one of the most natural and effective applications of a multiprocessor is the simulation of another multiprocessor.

In 'Processor-processor communication', Gallacher presents techniques for industrial systems. In that setting, he has devised hard and soft communications protocols for serial communications among loosely coupled processors, without getting mired in unnecessary generality. Particularly worthy of note is his use of 'timeouts' to avoid deadlocks.

These two issues have offered many views of multimicroprocessor systems. This variety is a suggestion of both the diversity and the timeliness of such systems. The designer should avail himself equally of the examples of today's implementations and the theory and imagination that lay the groundwork for tomorrow's systems.

There is a sharp dichotomy between dedicated and general purpose systems. The dedicated systems are with us now. Multiple microprocessors are economically alternative to a larger processor in meeting real-time requirements. They also lead to simple modular design, for example as device controllers. Most of the technology necessary for the development of dedicated systems is readily available. A notable and essential weak point is still, of course, the testing and trouble shooting of asynchronous processes in both hardware and software. Several of the articles in these issues have addressed this problem, some with design methodologies and some with development systems. Needless to say, a working and easily used multimicroprocessor system is the most powerful tool for investigating parallel asynchronous processes. It is far more effective for software than simulation on a single processor, and can, for hardware purposes, monitor more lines than a logic analyser. Because of the value of existing multimicroprocessor systems in developing other such systems, we can expect progress in this area to accelerate rapidly.

The progress toward general purpose multimicroprocessor systems is less advanced, largely because of the broad scope of such systems. The hardware is in a state comparable to that for dedicated systems: it is mostly available, with much more to be desired for testing and trouble shooting. Software technology is not yet sufficient for these systems, and, consequently, is a field in which new and exciting developments can be expected. Various linguistic constructs are available to specify and control parallelism. As they are used more widely they will evolve, as have sequential languages, into languages that are both powerful and easy to use. Presently available software techniques and languages are suitable for system programming, and other programming done by programmers experienced in concurrent programming and willing to contend with occasional hardware dependence. We await a major breakthrough to make application programming for general purpose multimicroprocessor systems possible in a realistic context: the language processor, and not the programmer, must ultimately bear the burden of recognizing parallelism inherent in an algorithm and of controlling it. This will lead to syntactically minor but philosophically profound changes in programming languages. The research that can lead to such developments is widely carried out and so, although progress in general purpose systems will be slow in the immediate future, a vast proliferation of such systems lies just over the horizon.

Daniel Prener