PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

Theory and Practice of Concurrent Systems

Geoffrey Fox

California Institute of Technology

Pasadena, CA 91125

August 17, 1988

What are Fundamental Problems in Constructing and Using Concurrent Systems?

We can consider the issues in three categories: (i) Hardware, (ii) Systems Software and Programming Environment, (iii) Application Software.

(i) It appears to me that basic principles of the hardware architecture and construction are understood. One has a collection of nodes connected by some sort of switch or router to a collection of memories. Such machines can be built although there are several important questions.

   (a) Are the nodes powerful or wimpy? Is the Teraflop processor 20,000 twenty nanosec nodes or 1000 one nanosec nodes?

   (b) What is the nature of switch and correct tradeoff between latency, bandwidth and dependence on routing distance?

   (c) Should one build "smart" memories or switches?

   (d) Should control be SIMD or MIMD?

In each case, there will be an inevitable favoring of local data references and varying penalties for access to shared or non-local memories.

There are other architectures to be explored and discovered (using neural networks, dataflow, etc.) but currently known general principles seem to be sufficient to build high performance general purpose concurrent computers.

(iii) Experience such as that gained at Caltech [Angus 89, Fox 87f, Fox 88a, Fox 88b, Fox 88c,] have shown that a broad range of applications - including the majority of those running on current sequential supercomputers - perform well on concurrent computers with speed ups that are typically at least 80% of the number of nodes.

(ii) However, this experience was gained with unusually talented users who rethought problems and rewrote codes from scratch. Further, the resultant programs were quite small (500-5000 lines). In principle, this experience is directly applicable to large industrial and government (defense, national laboratories) applications which use "essentially" the same algorithms. However, these applications involve much larger codes (100,000 lines or more) and it is often impractical to rewrite code - especially for today's concurrent computers where software productivity is low. We need to develop the programming environment to allow semi-automatic (user + compiler) parallelization of existing code and to make it easier to develop new code. This will require both better tools and perhaps new languages. I see little consensus as to the appropriate approach to this issue and not convincing evidence that a good solution will exist in the near term. Thus, I consider that the development of a productive programming environment as the key problem in concurrent computing.

What are fundamental differences between sequential and concurrent systems? Should differences be exposed and hidden and at what level should they be addressed?

Parallel algorithms are usually quite natural e.g. one is often simulating the physical world - a well known parallel system. So in this sense, concurrent computers are natural and are not fundamentally different. However, current programming environments - especially languages such as C and FORTRAN - do not naturally support parallelism. In this sense, there is a fundamental difference between today's sequential systems (hardware + environment) and concurrent systems.

At Caltech, essentially all hypercube applications have addressed concurrency at the application program level. Users decompose data, and write the programs to control the different parts of the decomposed domain. This was the correct approach to quickly show that "parallel processing works" but it is certainly rather tedious and not very portable between concurrent machines of different architectures. Currently, I view the operating system as not getting involved at the level of concurrency within an application program. It should typically view a given application - consisting of many processes on many nodes - as a single entity. Concurrency should be handled at the level of system utilities (e.g. for load balancing [Fox 86f, Fox 88e]), compiler (e.g. for parallelization and vectorizing FORTRAN code) and novel languages. I see several approaches to concurrency at this level for example [Arvind 87, Allen 87, Callahan 88, Chen 88, Fox 85d, Kuck 86, Mirchandaney 88, Rose 87, Taylor 88, Wilson 87, Zima 88,] but I do not yet have a good feeling as to the status and promise of these very different methods.

Will there be a transition in mainstream computing from sequential to concurrent computing?

I certainly hope that such a transition occurs but I fear that, rather, we will evolve from sequential to concurrent computing. Consider Fig. 1 which plots computer performance as a function of time. Messina and I have adapted a plot due to Buzbee [Buzbee 87] to separate sequential and concurrent performance. Current "conventional supercomputers" are already parallel machines with up to 8 heads (ETA-10, CRAY-YMP) and several pipes (functional units) per head. In three years, we can expect this technology to lead to 64 processor machines. We have plotted current distributed memory MIMD machines scaling the number of nodes and hence performance to a large machine with a price tag around $20M - we can define a supercomputer as what you can do for this price. Even with this scaling, the parallel machines are not of significantly higher performance than their commercial "sequential" competition. Thus, we expect the "conventional" (IBM, ETA, CRAY in the U.S.A.) approach to dominate high performance computers in the near term. This approach offers competitive cost-performance and typically better software environment than the "massively-parallel" machines. This assertion assumes that conventional supercomputers use their different heads to run different jobs and do not multitask within a job. Parallelism is achieved by the compiler by using the several pipes on a given head. This conventional approach will lead to concurrent computing with 64 nodes in the early 1990's and 1024 in the later part of the decade. Thus, we have a natural commercial evolution to concurrent computing with the conventional approach eventually needing the programming environment advances discussed above to decompose over 64-1024 heads. A sharp transition to parallel processing, rather than an evolution is possible but it needs development of "massively-parallel" hardware that is clearly more cost-effective than the conventional competition. It also needs the productive software environment already discussed. These are challenges to the parallel computing industry and the computer science research community.

Is there a difference between parallel and distributed systems?

This is partly semantics! Let us interpret parallel to mean tightly coupled cooperating processes such as those involved in domain decomposition of a scientific computation. Let us define distributed to mean more loosely coupled processes such as those occurring in a functional decomposition - say of servers for an operating system or controllers of different parts of an automobile. Then these systems can clearly be approached from a common point of view but this may have limited value. In my terminology [Fox 88a, Fox 88b] parallel systems require substantial interprocess communication in a loosely synchronous fashion; distributed systems often need less communication and are naturally asynchronous. I suspect that it may be crucial to build these characteristics into the support environment to achieve an efficient productive system. For instance, distributed systems are perhaps naturally approached in an object-oriented fashion; parallel systems by a language like C* or CPC [Felten 88a, Rose 87].

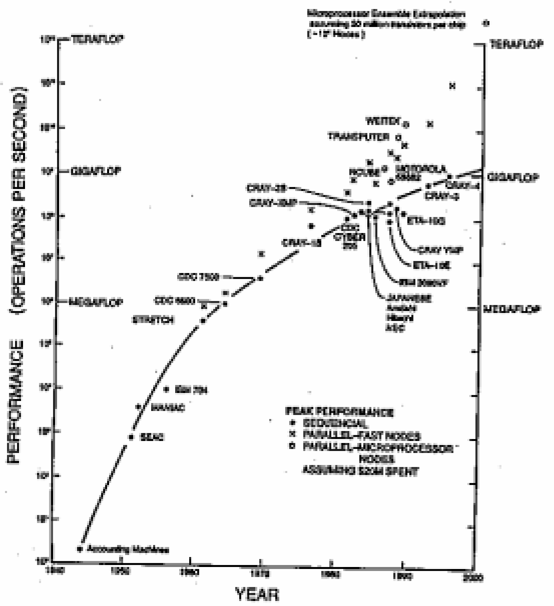
What is the structure of future general purpose concurrent computer hardware?

As discussed above, several of today's architectures naturally scale to large machines and are essentially general purpose e.g., can tackle all large scientific computations [Fox 85c, Fox 88b, Gustafson 88]. Examples are the hypercube, transputer arrays and their relations; Butterfly, RP3, Ultracomputer and the SIMD Connection Machine. The latter can naturally address about 50% of major computations on today's supercomputers [Fox 88b].

It is important to improve the cost performance of future machines to compete more favorably with conventional computers.

Is concurrency an artificial nuisance inflicted on us by the deficiences of VLSI techniques?

No, computation is the modelling of one complex system (the problem) by another (the computer). The essential task is to map these systems on to each other. Both systems are naturally parallel and in neither case is the parallelism artificial. Today's nuisance is caused by the available tools and is not intrinsic.



References

[Allen 87] Allen R., Callahan D., Kennedy K., "Automatic Decomposition of Scientific Programs for Parallel Execution", in proceedings of "14th ACM Symposium on the Principles of Programming Languages", Jan. 87, ACM, New York, N. Y.

[Angus 89] Angus, I., Fox, G., Kim, J. and Walker, D. "Solving Problems on Concurrent Processors - Software Supplement," to be published by Prentice Hall 1989.

[Arvind 87] Arvind and Nikhil, R. S., "Executing a program on the MIT tagged-token dataflow architecture," PARLE Conference, in *Lecture Notes in Computer Science, 259* edited by G. Goos and J. Hartmanis, Springer-Verlag, New York (1987), 1.

[Buzbee 87] Buzbee, B., "Supercomputers: values and trends," *Int. Journal of Supercomputer Applications 1* (1987) 100.

[Callahan 88] Callahan, D., Kennedy, D. 1988 "Compiling Programs for Distributed-Memory Multiprocessors," in proceedings of "1988 Workshop on Programming Languages and Compilers for Parallel Computing," Cornell, August 2-5, 1988.

[Chen 88] Chen, M., Choo, Y., Li, J., "Crystal: From Functional Description to efficient Parallel Code" in proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, edited by G. C. Fox, published by ACM, New York, N. Y., [Fox 88c]

[Felten 88a] Felten, E. and Otto, S. W. 1988 "Coherent Parallel C," in proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, edited by G. C. Fox, published by ACM, New York, N.Y., [Fox 88c], Caltech report C^3P-527.

[Fox 85c] Fox, G., "The performance of the Caltech hypercube in scientific calculations: A preliminary analysis" in *Supercomputers-Algorithms, Architectures, and Scientific Computation*, edited by F. A. Matsen and T. Tajima, University of Texas Press (1987), Caltech report C^3P-161.

[Fox 85d] Fox, G. "Use of the Caltech Hypercube": IEEE Software, Vol. 2, p. 73 (July 1985), Caltech report C^3P-162.

[Fox 86f] Fox, G. C., "A Review of Automatic Load Balancing and Decomposition Methods for the Hypercube," November 1986. The Proceedings for the Workshop on Numerical Algorithms for Modern Parallel Computer Architectures, held at the IMA in November 1985, published as Volume 13 in the IMA Volumes in Mathematics and Its Applications, *Numerical Algorithms for Modern Parallel Computer Architectures*, (Springer-Verlag), New York, Caltech report C^3P-385.

[Fox 87f] Fox, G. and Frey, A. 1987 "High Performance Parallel Supercomputing Application, Hardware, and Software Issues for a Teraflop Computer," Caltech report C^3P-451b.

[Fox 88a] Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. 1988 "Solving Problems on Concurrent Processors," published by Prentice Hall 1988.

[Fox 88b] Fox, G. C. 1988 "What Have We Learnt from Using Real Parallel Machines to Solve Real Problems?" Invited talk at the Third Conference on Hypercube Concurrent Computers and Applications, sponsored by the Jet Propulsion Laboratory, Pasadena, CA, Jan. 19-20, 1988, in proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, edited by G. C. Fox, published by ACM, New York, N.Y., Caltech report C^3P-522.

[Fox 88c] Proceedings of Third Conference on Hypercube Concurrent Computers and Applications, edited by G. C. Fox, published by ACM, New York, N.Y.

[Fox 88e] Fox, G. C, and Furmanski, W. 1988 "Load Balancing Loosely Synchronous Problems with a Neural Network," in proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, edited by G. C. Fox, published by ACM, New York, N.Y., [Fox 88c], Caltech report C^3P-363b.

[Gustafson 88] Gustafson, J. L., Montry, G. R., Benner, R. E. 1988 "Development of Parallel Methods for a 1024-Processor Hypercube," SIAM journal on Scientific and Statistical Computing.

[Kuck 86] Kuck, D. J., Davidson, E. S., Lawrie, D. H., Sameh, A. H., "Parallel supercomputing today and the Cedar approach." *Science 231*, (1986), 967.

[Mirchandaney 88]	Mirchandaney R., Saltz J. H., Smith R. M., Nicol D. M., Crowley K., "Principles of Runtime Support for Parallel Processors", in proceedings of "1988 International Conference on Supercomputing", St. Malo, July, 1988, published by ACM, New York, N. Y.
[Rose 87]	Rose, J., Steele G. 1987 "C*: An extended C Language for Data Parallel Programming", Thinking Machines Corporation.
[Taylor 88]	Taylor, S., Shapiro, R. and Shapiro, E. 1988 "FCP: A Summary of Performance Results," in proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, edited by G. C. Fox, published by ACM, New York, N.Y., [Fox 88c].
[Wilson 87]	K. Wilson, "The Gibbs Project" in Supercomputers - Algorithms, Architectures and Scientific Computation", edited by F.A. Matsen and T. Tajima, University of Texas Press (1987).
[Zima 88]	Zima, H. P., Bast, H-J., Gerndt, M., "SUPERB: A tool for semi-automatic MIMD/SIMD parallelization," Parallel Computing 6, 1 (1988).