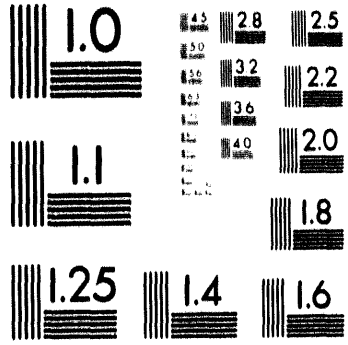
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

1 of 1

Proc. of the 1989 Data Engineering Conference, Los Angeles, California, February 1989 (Invited position paper)

PARALLEL COMPUTING: ONE OPPORTUNITY, FOUR CHALLENGES [1]

Jean-Luc Gaudiot

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California
(213) 743-0249

ABSTRACT: Maturing technologies of integration have provided system designers with an opportunity to increase system performance at little design cost. New approaches are however necessary if one is to take full advantage of this potential.

1 THE OPPORTUNITY:

In the recent decade, we have witnessed an exponential improvement in technology. Both device speed and levels of integration have been growing at a rate which approximately doubles performance every year in terms of complexity and speed. Indeed, Very Large Scale Integration (VLSI), now followed by multiple chips on a chip (Wafer Scale Integration or WSI) are allowing increasingly complex electronics within the same package. In addition, this very capability for high integration has allowed signal paths to become shorter and has therefore resulted in lower communication costs and correspondingly higher basic clock rates. These improvements have not been obtained without cost: first, the high degree of integration renders chips much more sensitive to design defects and have correspondingly reduced fabrication yields. New approaches which includes the addition of redundant circuitry have been necessary in order to raise yields to economically acceptable values. Second, the complexity of each chip has raised design costs and conversely diminished applicability of each new device. Standardization and "silicon" foundry methods [7], however, allow a significant reduction of initial design costs. Nevertheless, this last problem implies that simple structures must be repetitively used within the same chip (so as to lower design costs) or that large numbers of the same chip must be usable within the same system (so as to increase its overall applicability).

Conversely, improvements in speed and level of integration have been recently approaching physical bounds: integrated circuit connections cannot be further reduced without compromising the integrity of the electrical connection or introducing severe metal "migration" problems. Likewise, signals are now traveling within a chip at speeds close to the speed of light.

As a consequence of both the cost and technological constraints, system designers must resort to other methods in order to improve system performance. Therefore, incorporating multiple identical processors within the same system should provide the user with a proportional improvement in throughput at much less cost than would have been if a conventional processor (much more complex) had been designed for the same performance.

[1]This material is based upon work supported in part by the US Department of Energy, Office of Energy Research, under Grant No. DE-FG03-87ER25043

2 THE PROBLEM:

Once our system designer has made the decision of meeting the cost/performance trade-off by a parallel processing approach, more technical commitments must be made. It is indeed necessary to reliably interconnect the processing units. When the processors are located on the same wafer, a "burn-in time" determination of healthy units must be made. This identification of unusable components is needed so as to create a logical structure which will be composed of only properly operating elements. Alternatively, if the system is composed of processors-on-a-chip (or multiple chips per processor), dynamic failure of processing elements should be handled "on the fly," so as to provide the user with uninterrupted operation.

In addition to this reliable operation problem, proper synchronization of the units must be implemented so as to insure deterministic and safe termination of the user program (note that "safe" in this context is defined as the ability to execute a program on a multiprocessor while producing the same results as would a single processor architecture).

In summary, the problem of modern parallel computing is to reliably interconnect and synchronize units in a system. Clearly, technological improvements have added to the dimensions of design constraints while obviously increasing the application potential.

3 THE FOUR CHALLENGES

The field of computer architecture is technology-driven and advances must take into account the limitations imposed by such technology. Further, the requirements imposed by the need to properly interconnected large numbers (perhaps in the thousands) of Processing Elements bring a new dimension to computer system design [5]. We have indeed identified four topics which are the cornerstone of the field of parallel computing.

3.1 Programmability:

This is the first and perhaps the most important problem of the four. As described in the previous section, future systems will comprise large numbers of Processing Elements. No centralized control can be implemented in such conditions since it would

be an inherent bottleneck. Furthermore, the concept of global memory system cannot be easily implemented lest *memory* latencies become prohibitive. We must therefore have recourse to the concepts of distributed memory multiprocessor systems in which Processing Elements are independently controlled and each own a local memory bank. Communication among processing units is made by "message passing" through some kind of packet-switching communication network. When such a system is implemented, the Processing Elements are allowed to proceed asynchronously, without interruption but for synchronization points, semaphore readings, and other parallel scheduling needs. While such explicit parallel approaches to programming can be considered reasonable to program small-scale, shared memory, multiprocessors (*e.g.*, Cray X-MP, Cray-2, etc.), they cannot be considered valid any longer for large-scale distributed memory multiprocessors. Indeed, the programmer cannot be expected to maintain thousands of processes active and properly synchronized among each other.

Finding efficient, architecture-transparent programming paradigms for large-scale multiprocessors is the 1st challenge of parallel computing.

3.2 Communication network design

At the core of our distributed *memory*, message-passing multiprocessor system is the message-passing communication network. As an element in such a pivotal position, it must meet many requirements:

- *diameter*: maximum distance between nodes must be minimized.
- *average bandwidth*: random pairs of nodes must be allowed to communicate with as much bandwidth as possible.
- *connectivity*: each node is usually allowed a limited number of connections with its neighboring processors.
- *number of redundant paths*: for fault-tolerance reasons as well as traffic management purposes, pair of processors must be linked by as many paths as possible.

Clearly, these requirements are inter-related and oftentimes contradictory; for one thing, hardware considerations may limit the connectivity which would reduce network bandwidth and diameter.

A solution to these tradeoffs will answer the 2nd challenge of parallel computing.

3.3 Reliable operation

For a given device technology, increasing the machine size will unavoidably correspondingly increase the failure rate of the whole system. The presence of multiple identical Processing Elements indeed provides at the same time a challenge (increased failure rate over the same system with a single processor) and an opportunity. Since all these Processing Elements are identical to provide increased performance, their redundance can also be used to provide fault-tolerance and continued operation, albeit in a degraded mode, in the presence of multiple failures. However, while message passing multiprocessors comprise several hundreds of independent identical processing units, program and data allocation has rendered them different from each other. Recovering from the failure of one unit means not only routing data and processing around the failed unit, it also means recalculating the data existing in the failed processor at the time of failure as well as re-allocating to healthy processors the original program. This is the 3rd challenge.

3.4 Performance evaluation and benchmarking

Early computers (and even early *microprocessors*) relied upon simple tests for benchmarking purposes. Common examples included gate speed or simple operation (addition) time, etc. This type of benchmark was applicable for simple computer structures with little architectural support for speed-up (pipelining, cache memories, etc.) However, with the advent of such improvements in the architecture, it became necessary to evaluate the behavior of entire programs. This was particularly the case for advanced supercomputers such as CRAYs which are heavily optimized for vector operations. It would still be meaningful but partially informative to evaluate the gate speed or the execution time of a single instruction in such an environment. Instead, more involved program structures such as the Livermore loops were introduced in the 1970s as test programs and comparison points for these new machines.

In short, the history of computing has so far seen an attempt not at doing faster the same things (although this has been accomplished by "natural" technological means) but at doing in similar time larger and larger problems. The machines of the next generation are arriving. The tests and benchmarks of the next generation are not. This is the 4th challenge of parallel computing.

4 THE FUTURE

While the challenges we just examined are serious indeed, research in progress is pointing towards solutions to the problems.

4.1 New models of computation (data-flow, neural nets)

Instead of the explicit approach to parallelism description described above, an implicit method to programming must be allowed. This is permitted by the functional model of execution in which instructions become functions which are executable when their arguments have been evaluated by other functions. A subset of functional programming, namely data-flow languages, has met with some success in this domain and has seen several successful implementations of data-flow machines. Research issues in the domain of structure handling, program partitioning and allocation, high-level language translation, etc. remain nevertheless numerous [1,2,3,4,6].

Another promising approach to fully utilize the power provided by low-cost hardware, is in "non-algorithmic" computing structures such as Artificial Neural Systems. In this method, instead of relying upon an algorithm programmed into the machine, a "training period" (supervised or unsupervised) allows

the network to appropriately configure itself to solve the problem at hand. These systems present advantages similar to that of data flow systems, in that they implement distributed control. Further, individual cells are simple. This can lead to efficient VLSI circuit structures which can be easily replicated.

4.2 New technologies of interconnection (i.e., optical)

One of the major advances brought by new technologies of high integration has been the reduction of computation costs. However, communication among processing units has become correspondingly relatively much higher. These costs have risen not only in terms of implementation but also in terms of performance (recall that communication between two chips is an order of magnitude slower than intra-chip communication). However, while electronic technology may have reached a peak in this domain, new optical interconnection networks offer hope in this regard. Research in optical crossbar connection networks is promising connection times of less than nanoseconds for widely separated components as well as high connectivity. This technology could in fact bring a solution to the problems of communication network design described in a previous section.

4.3 Better understanding of reliable operations (fault-tolerant algorithms)

Efficient, and reliable communication with multiple path routing are an essential component to a fault-tolerant multiprocessor. However, in addition, robust recovery algorithms must be implemented such as described in [8]. Further, reliable operation should be incorporated in both the design and application level of a multiprocessor system. This means that alternate algorithm (so called fault-tolerant algorithms) must be devised for proper completion of a computation in presence of failures. Alternatively, one of the strong advantages of Artificial Neural Networks rests on their inherent robustness and tolerance to stochastic behavior of some elements.

5 CONCLUSIONS

In conclusion, it can be reiterated that current technology enables hardware designers to deliver systems with very high raw computational throughputs provided high communication bandwidth can be insured among processors. However, an efficient utilization of this power will require that the user be given high programmability, as well as a reasonable assurance of safe termination of the computation in the presence of faults. Dataflow execution, Artificial Neural systems, optical technology of interconnection as well as new fault-tolerance techniques will contribute to this goal.

REFERENCES:

[1] Gaudiot, J.L., and Wei, Y.W., "Token relabeling in a tagged token data-flow architecture," in *IEEE Transactions on Computers*, in press.

[2] Gaudiot, J.L., and Lee, L.T., "Occamflow: a methodology for programming multiprocessor systems," in *Journal of Parallel and Distributed Computing*, Academic Press, in press.

[3] Gaudiot, J.L., "Data driven multicomputers in digital signal processing," in *Proceedings of the IEEE, Special Issue on Hardware and Software for Digital Signal Processing*, September 1987.

[4] Gaudiot, J.L., Campbell, M.L., and Pi, J.I., "Program graph allocation in distributed multicomputers," in *Parallel Computing*, Vol. 7, No. 2, North Holland Publishing Co., June 1988.

[5] Hwang, K., and Briggs, F., *Parallel Processing*, Addison Wesley.

[6] McGraw, J.R., and Skedzielewski, S.K., *SISAL: Streams and Iteration in a Single Assignment Language, Language Reference Manual, Version 1.2*, Lawrence Livermore National Laboratory, Technical Report M-146, March 1985.

[7] Mead, C., and Conway, L., *Introduction to VLSI systems design*, Addison Wesley, 1978.

[8] Najjar, W., and Gaudiot, J.L., "Network disconnection in distributed systems," in *Proc. 1988 International Conference on Distributed Computing Systems*, San Jose, California, June 1988.

DATE FILMED

11/19/93

END