Future High Performance Computation
The Megaflop per Dollar Alternative

Lennart Johnsson

Research Report YALEU/DCS/RR-360
January 1985

Work supported by the Office of Naval Research under Contract No. N00014-84-K-0043

Future High Performance Computation

The MegaFlop per Dollar Alternative

Lennart Johnsson
Department of Computer Science
and Electrical Engineering
Yale University

High performance in future computing systems to an ever increasing extent has to come from concurrency in computation. The performance improvements over the last few decades to a large extent have been due to technological developments. Switching speeds have been improved by about 5 orders of magnitude, but clock rates only with about 3 orders of magnitude. Architectural innovations have accounted for 2 - 3 orders of magnitude in increased floating-point capability per clock cycle, yielding a total improvement in floating-point speed of 5-6 orders of magnitude. Silicon technology is expected to offer about one order of magnitude increased switching speed before fundamental limits are reached. Other technologies, such as gallium arsenide, potentially offers a further factor of 5 to 10 in increased switching speeds.

Extreme performance in future systems must come from architectural innovations, in particular through highly concurrent systems. High real performance can also be accomplished through algorithmic innovations, and such innovations are in general necessary to obtain high *real* performance on a system of high *nominal* performance. The experience gained from current vector machine architectures have made the interdependence between architectures and algorithms very clear. Some algorithms that are efficient on sequential machines are inherently limited in their ability to benefit from realistic parallel architectures. Others can be adapted to a variety of such architectures. The adaptation in many instances has the nature of an algorithm or program transformation.

Abstract representations of algorithms that captures the essence of the behavior of the algorithm are urgently needed, as are efficient transformation techniques that generates the appropriate data and control structures for the target architecture. A suitable abstract representation relieves algorithm designers from all the minute details of particular machine architectures. The variety of parallel architectures is indeed very large compared to uniprocessor architectures. In highly concurrent architectures the communication time is of prime importance. Entirely new algorithms that minimize the time of arithmetic operations and communication on classes of architectures are likely to be discovered. High real performance and the success of any architecture depends on effective programming models and systems. Algorithm transformation techniques will be an important part of such systems.

An ideal architecture must be scalable both with respect to the desired performance and the technology. It is also highly desirable that the essential architectural characteristics are stable with respect to these forms of scaling. The technology, the architecture, the algorithms and applications, and the programming models and systems, all have different dynamics. It is highly desirable that the dependence of any one of these components on any others is minimized in order that advances in any one area can be introduced at the rate they are produced.

Ensemble Architectures

Ensemble architectures represent a low cost alterrnative to future high performance systems. An ensemble architecture is composed of large numbers of relatively simple (small), mass produced processors, each equipped with local storage. The processor interconnections form a sparse, regular graph. Control, as well as data, are distributed. There are only a few shared resources, such as storage at some point in the hierarchy, where the bandwidth requirement is low, and certain peripherals.

High performance requires a high rate of instruction execution, given that each instruction has a limited set of operands, and high storage bandwidth. In the ensemble architectures we use as model for algorithm development, high instruction execution rates are achieved through replication of processing units, each executing its own, distinct, instruction stream. In Flynn's [2] classification we consider MIMD (Multiple Instruction streams Multiple Data stream) architectures. High storage bandwidth is achieved through a highly partitioned storage. Each processing unit has its own local storage. A high nominal performance is obtained through replication of mass produced parts implemented in state-of-the-art technology for an excellent cost/performance ratio.

A high performance design should also allow high clock rates in any given technology. Simple processors are smaller than processors with large instruction and register sets. Simple processors can be designed effectively with regular structures without severe area and performance penalties [6], [7], [19], [18], [23]. Pipelining of functional units that significantly adds to the complexity of scheduling operations can be reduced to a minimum. Wire delays are of increasing importance as feature sizes of the technology are reduced. Indeed, wire delays already are of main concern with respect to performance. The difference in effort of designing a fast simple and a fast sophisticated processor will be magnified in the future.

The same argument applies to storage as well. A large on-chip storage based on an array design may actually become slower as the feature sizes are reduced. To maintain or improve the speed of storage with reduced feature sizes it ultimately becomes necessary to structure storage itself [16], which may reduce the density, and cause access time to be nonuniform. Intermingling of storage and processing reduces the average area per processing element, and increases the clock rate in a synchronous (nonpipelined) design. The increased ratio of processing capability per unit of storage, and the increased clock rate both contribute to increasing the maximum size of the state that can change in a single clock cycle, i.e., the rate of computation.

At the extreme, bit-serial, pipelined, logic can be used. Bit-serial architectures may allow clock rates up to an order of magnitude higher than a word parallel design. Bit serial designs also require less area. Taking a very idealistic view, one should be able to fit as many bit serial units as bits in a word in the same area a word organized unit requires. Hence, with this simplistic assumption the same amount of logic fits in a given area. The same amount of computation can take place concurrently. But, the rate of computation is higher in the bit-serial case. The bit-serial design is preferable with respect to nominal performance.

High real performance also requires an appropriate node architecture, providing hardware support for frequent operations. Such an optimization depends upon the target class of applications, and upon what operating system features are provided. Hardware for floating-point operations is a must for compute-intensive applications in scientific computing and advanced signal processing, such as VLSI device and circuit simulation, and the transient sonar problem. For ultimate performance, potential concurrency in primitve operations such as arithmetic operations, should be exploited.

An architecture achieving high nominal performance through replication of parts requires effective communication and synchronization. As the processors and their storage are reduced in size and increased in number, the interconnection problem increases both in complexity and importance. The speed of interprocessor communication relative to the speed of accessing local storage increases with increased locality of interconnection. Currently, interprocessor communication is generally inter-chip, or inter-board communication, which represents different challenges from on-chip communication.

The increased importance of interprocessor communication with reduced processor size also stems from the fact that with reduced granularity of the computations the amount of computation per communication decreases. The sensitivity of the performance of the system to different network configurations increases, as does the sensitivity to data allocations (and reallocations). In a system of course grain size the sequential time dominates the time for communication for most computations, reasonable communication speeds, and data allocations, regardless of the interconnection scheme.

Manufacturability, and scalability of ensemble architectures with respect to performance and reduced feature sizes of the evolving technology, are assured by interconnecting the processing elements sparsely and regularly. The richness of the interconnect is a point of trade-off between performance, fault tolerance, and design and manufacturing concerns.

High real performance and scalability are accomplished by enforcing the concept of locality wherever possible. Control and storage are effectively distributed among the processing elements, eliminating potential bottlenecks and sources of limited scalability. High real performance is achieved by devising algorithms that minimizes the time devoted to arithmetic <u>and</u> communication. Global communication is accomplished via a succession of local communications. Algorithms need to be devised with respect to the communication time required by the best possible embedding of the computation graph in the ensemble architecture. In some instances a static, properly embedded, data structure allows the computation to be carried out in minimum total time. Other cases require dynamic data structures (or reallocation of the data structure). The solution of tridiagonal systems of equations by cyclic reduction on ensembles configured as binary trees is an example of the latter kind of computation [9], and is discussed further below.

The need for data reallocations, as well as the ease of finding low cost embeddings of computation graphs, depends on the topology of the ensemble, and the form of interconnect. A topology is said to be more powerful than another if any algorithm, or a sufficiently large class of algorithms, that run in a given time on the latter can be made to run on the former with an increase in running time by only a constant factor, but the reverse is not true. For instance, a shuffle-exchange network is considered more powerful than a binary tree, since any tree algorithm can be made to run on a shuffle-exchange network with the running time increased by a constant factor, but the converse is not true. The computation of the Fast Fourier Transform is a prime example. The running time of an algorithm on a shuffle-exchange network may be decreased by a constant factor if the algorithm is mapped on to a boolean n-cube. Many graphs can be embedded in a boolean cube preserving proximity. A given data structure can support many types of access schemes without communication penalty, reducing the need for data reallocations. Reconfigurability, i.e., programmable interconnections, also allows a fixed data structure to efficiently support a variety of access schemes.

A high real performance for data independent computations is attained by a *compile time* mapping of the computations on to the processors of the ensemble. Whenever it is possible to find a set of rules for the mapping of a computation with data independent control flow on to the ensemble, the mapping can be made part of the compilation process. The user is relieved of the need to have detailed knowledge of the architecture. Where compile-time mapping is not used an effective *run-time* mapping is required. Such a mapping must take processor load as well as communication needs and resources into account.

Interconnection Networks

The communication capabilities are of prime importance in an ensemble architecture. The selected processor interconnection scheme affects scalability, wireability, and performance. Clearly, from a designers point of view it is desirable to minimize the amount of interconnect, whereas from a performance point of view a high bandwidth between arbitrary processors is desirable. The feasibility of different forms of interconnect from a design point of view is related to the granularity of the processors: a large number of fine grain processing elements may fit on a single chip, but there may be pin-out difficulties in going off chip, and further difficulties in interconnecting a large number of processors on chip. We briefly review the communication and design characteristics of a few processor interconnection networks considered in the discussion of ensemble architecture algorithms. A survey of interconnection networks is given by Siegel [24], and a collection of papers on networks contained in [27].

Configuring processors as linear arrays and binary trees requires a total number of interconnections equal to the number of processors. Both configurations scale in an excellent way. With several processors on a single chip, the required bandwidth at the chip boundary only grows at the rate of the clock frequency, regardless of the number of processors per chip and the size of the machine being built, [15]. The tree has an advantage over the linear interconnect since the maximum distance between processors only grows logarithmically in the number of processing elements. Global communication can be accomplished faster than in a linear array.

However, linear arrays and trees do present communication bottlenecks for large classes of computations. Richer interconnection networks are 2-dimensional meshes, shuffle exchange networks, cube connected cycles [20], and boolean n-cubes. The number of interconnections per processor, the fanout, is 3 for almost all processors in a shuffle-exchange network (half the processors in a binary tree also have fanout 3), 4 for the square 2-dimensional mesh, 6 for the hexagonal mesh, and $\log_2 N$ for a boolean cube. The diameter of the network topology defines a lower bound for the speed of computation, [4]. The diameter of a binary tree of 2^n-1 nodes is 2(n-1). For a the shuffle network of 2^n nodes it is 2n-1, and for a boolean n-cube (2^n nodes) it is n. A 2-dimensional square mesh has a diameter of $2\sqrt{N}$ without end-around connections, $\sqrt{N}$ with end-around connections. Binary trees and shuffle networks offer a small diameter for few interconnections. The shuffle networks are more powerful than the binary tree, but the tree is a recursive structure, and has efficient planar layouts, $O(N)$ [17] for the H-tree, or $O(N\log_2 N)$ [1] for all leaf nodes on the boundary. The layout of shuffle exchange networks require a larger area, $O(N^2/\log_2^2 N)$ [14]. The layout of a boolean n-cube requires area $O(N^2)$, [15], and the layout of a cube connected cycles network requires area $O(N^2/\log_2^2 N)$, [20].

The number of interchip (or interboard) connections grows at a lower rate for the mesh than for shuffle networks, that in turn grows at a lower rate than the number of interchip interconnections for a boolean n-cube. The n-cube is the most difficult to wire of the configurations discussed here, but it offers higher bandwidth between arbitrary processors than any of the other alternatives. Another definite advantage of the boolean cube over shuffle networks is that it is recursive. Extensions of a cube does not require a complete rewiring, but for an easy extension it needs to be anticipated at design time so that a sufficient number of ports are available.

An alternative to direct links between processors is some form of switch network, such as an Ω-network. The bandwidth of a pipelined Ω-network, like that planned for the proposed NYU Ultracomputer [5], is essentially the same as the bandwidth in a boolean cube, given that the two alternatives have an equal number of processors. However, there is a latency in the switch that grows logarithmically with the number of processing elements. In the boolean cube the interprocessor communication time is nonuniform. The minimum interprocessor communication time amounts to one routing, the maximum number of routing steps is $\log_2 N$. Interprocessor communication in an architceture of the Ultracomputer type always require $2\log_2 N$ routing steps.

For a large cube, or a large switch, the time for interprocessor communication in the cube, and the time between stages of the switch, is likely to grow with the number of processors, due to increased distance of communication. Though there is a qualitative difference in the communication capabilities, the quantitative difference depend on various implementation decisions. Furthermore, whether the nominal difference results in a real performance difference depends on the particular data dependences of the computation, and the mapping of the computations on to the architecture. With simple switching elements, and several elements on a chip, the fanout may become a problem in a switch based architecture at an earlier stage in the evolving technology than in a machine with processor-to-processor connections. One can view such an architecture as one in which the complexity of the switches is increased to yield regular processing nodes.

Ensemble Architecture Algorithms

An ensemble architecture of extreme concurrency is similar to systolic architectures. However, in ensemble architectures data management is an even more predominant factor. In most, but not all, systolic architectures most of the data (input and output) are stored outside the array, and the management of such data generally ignored. In algorithms for ensemble architectures it is generally assumed that initial as well as resulting data are stored within the ensemble. Furthermore, the ensemble is in general of a smaller size than "the maximum possible" as defined by characteristic parameters of the problem. The amount of storage per node is significant. The granularity of computations and processors in ensemble architectures is often larger than in systolic architectures.

Time-space trade-offs are at the core of mapping algorithms onto ensemble architectures. Data and control structures, synchronization and communication are, in general, considerably more complex in ensemble architectures than in systolic designs. The time-space trade-off in ensemble architectures is often made in favor of minimizing data movement. Systolic designs are of fine grain and designs are often such that the communication time is comparable to the time for logic or arithmetic operations. Ensemble architectures are of a courser grain and interprocessor communication is typically slower than the execution of arithmetic and logic operations.

Algorithms are devised both in an ad hoc manner, and systematicly. Though the first approach may lead to entirely new, efficient, algorithms we focus on the systematic approach to developing ensemble architecture algorithms. Such an approach can be supported by algorithm design tools, and provide the necessary insight to develop compilation techniques that transforms abstract representations of algorithms into efficent code for a variety of architectures. In the approach we use, and that is followed in the description of sample algorithms below, a computation graph defining the partial ordering of computations is first created from the definition of the computation in a suitable notation. Then, this computation graph is mapped on to the ensemble. We often carry out this mapping in two stages. First, a mapping is carried out for the case of a "sufficiently" large ensemble. In such an ensemble the mapping of the computation graph can be made such that all nodes of a given level (order) are assigned to distinct processors in the ensemble. The situation in this case is similar to what is typical for systolic designs. In the second step the actual computation graph is mapped on to the ensemble. This step can often be thought of as a folding of the computation graph on to itself, until the reduced graph is of a size that fits the ensemble. However, for optimum utilization of the resources the procedure is quite complex, as will be illustrated further. The optimum mappings may depend upon the relative sizes of the problem to be solved and the ensemble architecture, and whether one or several problems are to be solved.

As an example of the algorithm desgign procedure we use, the solution of tridiagonal systems of equations is considered. Ignoring possible numerical difficulties, a system of N equations can be solved in time $\log_2 N$ by cyclic reduction if only arithmetic operations are considered. The solution of tridiagonal systems on binary trees is interesting not only for the importance of efficient tridiagonal solvers, and the relative simplicity of constructing large tree ensembles, but also from an algorithm design point of view. First, there exist a mapping of the computation graph for cyclic reduction on N equations on to a binary tree of N nodes such that the communication complexity is $3\log_2 N$. A comparable communication complexity is also obtainable on shuffle networks and boolean cubes, [9]. Second, for $N > K$, a dynamic data structure is required for minimum communication complexity. Third, a poly-algorithmic approach yields a communication complexity of minimum order. The third property is also true for the shuffle and cube configured ensembles.

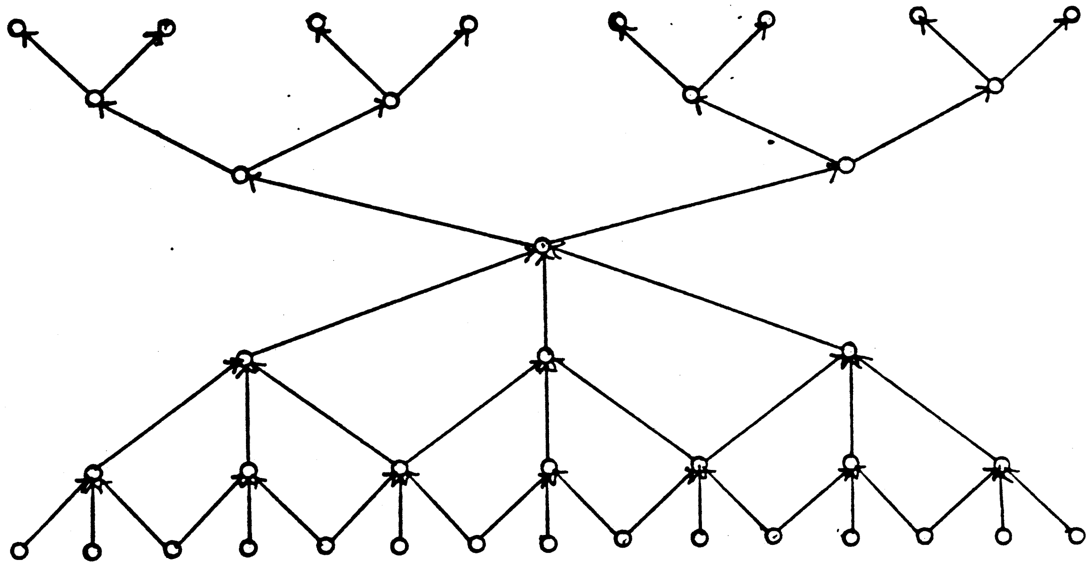

Figure 1: The computation graph for odd-even cyclic reduction

The computation graph of cyclic reduction is shown in Figure 1. For N=K mapping the equations on to nodes in the tree in inorder for every level of the computation graph, yields a map with the desired order of complexity. The first reducton step requires a time proportional to k, the second a time proportional to k-1, etc. But, the reduction steps can be pipelined, and the total time is proportional to k, [12], [3].

For $N=2^n-1 \geq K$ it is necessary to map several equations on to a tree node. One natural approach is to first map the computation graph in inorder on to a logic tree of size N, and then to map this logic tree on to the ensemble tree. This mapping can be done by folding the tree on to itself, or mapping several adjacent levels of the logic tree into one level of the processor tree. However, such an approach does not yield an efficient algorithm in the case of cyclic reduction, neither with respect to the balance of computational load, nor with respect to communication.

Another natural approach to the mapping of the computation graph on to the tree is the formation of a quotient graph from the computation graph by combining a number of successivly indexed equations into a node in the quotient graph. This approach is similar to domain decomposition in the solution of partial differential equations. The quotient graph is then mapped on to the processor tree as in inorder. The number of qoutient nodes with $\lceil N/k \rceil$ equations is $2^{nmodk-1}$, which corresponds to a nmodk level binary tree. The quotient graph approach provides the best possible computational balance.

A critical observation in finding a communication efficient mapping is that the communication between some quotient nodes alternates in direction every reduction step. The efficiency of the inorder map relies on the fact that the communication is unidirectional, and can be pipelined. A communication (and arithmetically) efficient map in the case of cyclic reduction on a "large" set of equations on a "small" tree ensemble is obtained by a proximity preserving map of the quotient graph on to the tree, for the first several steps until there is one "active" node per quotient node, followed by an inorder map for the last k reduction steps. The remapping can be made in k-3 routing steps [9]. Figure 2 illustrates a proximity preserving embedding of a path in a binary tree. The embedding is a minor variation of the embedding by Rosenberg and Snyder, [22].

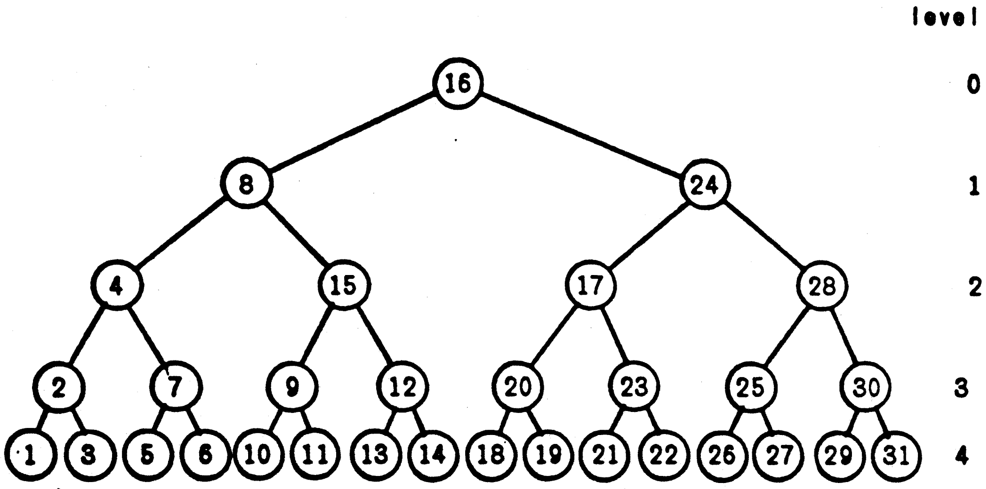

Figure 2: A proximity preserving path embedding in a binary tree

Hence, in the case of cyclic reduction on a binary tree ensemble, and $N > K$, a dynamic data structure yields a communication complexity of the same order as the arithmetic complexity on a parallel processor with unbounded parallelism and shared storage, i.e., $O(\log_2 N)$, whereas for $N=K$ a static inorder map yields the same result.

However, cyclic reduction with the dynamic data structure described above does not yield a communication complexity of the same order as the lower bound, which is $O(\log_2 K)$ [4]. A tridiagonal solver having communication complexity of minimum order, without increased arithmetic complexity, can be obtained by combining Gaussian elimination and cyclic reduction, even though Gaussian elimination essentially is a sequential method. Possibly, the elimination can progress concurrently from the upper left and the lower right corners towards the middle, and back substitution in the opposite directions. For $N > K$, Gaussian elimination can be used concurrently in all processors, reducing the set of equations forming a node in the quotient graph to a single node. Fill-in is created, but the arithmetic complexity is approximately the same as for cyclic reduction, [26]. In this reduction only one interprocessor communication is needed, compared to $\log_2(N/K)$ for cyclic reduction on a binary tree. The reduced system is then solved by cyclic reduction using an inorder map. The total complexity is of order $O(N/K+\log_2 K)$ [9].

The combined algorithm is of minimum order of complexity both with respect to communication and arithmetic. The communication time is proportional to the diameter of the ensemble. This minimum order algorithm is of the poly-algorithmic type. However, the pure cyclic reduction algorithm may have advantages in the case of truncated cyclic reduction. But, Reiter and Rodrigue [21] have showed that under certain conditions the fill-in elements decrease in magnitude, and a truncation of the reduction process may be possible also for the poly-algorithmic approach. The reduction in computational complexity rendered by truncating the reduction process is relatively much more significant in a highly concurrent system than in a single processor system. For N=K the running time is proportional to the reduction steps executed, while on a single processor architecture half of the total (untruncated) execution time is spent in the first reduction step, a quarter in the second, etc.

On a boolean cube a static data structure yields an implementation of cyclic reduction of communication complexity of order $O(\log_2 N)$. Combining Gaussian elimination and cyclic reduction yields a total complexity of minimum order with quotient sets mapped to nodes according to a Gray code. The binary reflected Gray code also allows for simple, distributed control. Each processor can determine from its address and the reduction step being executed [9] with which neighboring processor to communicate, and what information shall be transmitted/received. Successive levels of the computation graph are mapped into subcubes of successively reduced dimensionality. For N=K all processors participate in the first reduction step, about half of which only perform communication. The equations participating in the second reduction step are moved to one half of the cube, and the process is repeated recursively. Figure 3 illustrates a few steps in the reduction process for N=K=8.

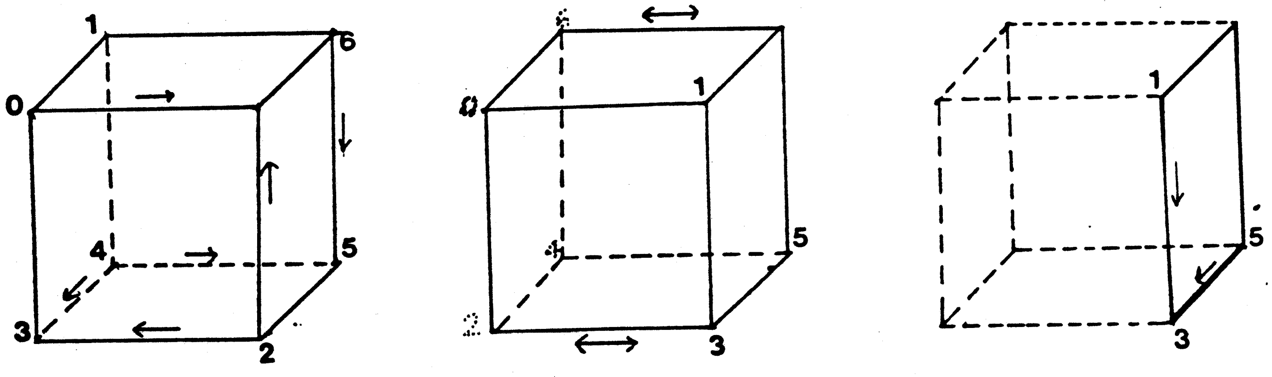

Figure 3: Cyclic reduction on a boolean cube

The optimum ensemble architecture algorithm may also depend on the number of problems to be solved. We again use the solution of tridiagonal systems as an example. The solution of multiple tridiagonal systems is part of efficient Poisson solvers such as FACR, and the ADI method. Using cyclic reduction one processor is active during all $\log_2 N$ reduction steps. In the event of multiple problems and all problems mapped in the same way the time per problem is $O(\log_2 N)$. Even with different problems mapped differently to balance the computations perfectly, the minimum time for N problems on an N processor ensemble is $O(N+\log_2 N)$, whereas it is $O(N)$ for Gaussian elimination. The constant multiplying N is smaller in the latter case. If the ensemble instead is partitioned such that there is one problem per partition, Gaussian elimination nevertheless becomes more efficient at some point.

The solution of banded systems with dense bands is another problem class where a poly-algorithmic approach may be the most effective for certain combinations of ensemble size, matrix bandwidth, and matrix order. In [10] we devise an algorithm that solves a banded system of bandwidth $2m+1$ in time $O(m\log_2 N/m)$. The algorithm assumes that the ensemble is partitioned into clusters, with intracluster connections in the form of boolean cubes or toruses, and the intercluster connections in the form of binary trees, shuffle networks or boolean cubes. In this particular case it is desirable to maximize the cluster size.

The conclusions of these simple examples with respect to algorithm design are that

- computation graphs are suitable representations from which suitable mappings can be found.
- the optimum mapping may include several different strategies (such as proximity preserving and inorder for cyclic reduction on a binary tree), and the strategy may depend on the problem size relative to the size of the ensemble.
- a combination of algorithms may yield the lowest communication, arithmetic or total solution time

Data movement is a key issue in ensemble and systolic architectures. In an ensemble architecture dedicated to one, or possibly a few, "high-level" operations, (e.g. matrix multiplication) the topology may be choosen to reflect the ideal data structure. However, in most cases the computations to be performed by the ensemble is composed of several "high-level" operations, each with its own ideal data structure. It becomes necessary for optimum performance to perform data reallocations. As in the case of cyclic reduction on a binary tree it may also be necessary to perform such reallocations as part of a single "high-level" operation.

Reconfigurability obtained through programmable switches, as in the CHiP architecture [25], or through sufficently many interconnections that several proximity preserving embeddings can share the same data structure, significantly reduces the need for data reallocations. It is important to consider both space and time, or the dynamic behavior of the algorithm, in the embedding. For instance, a binary tree of 2^k-1 nodes cannot be embedded in a cube of 2^k processors preserving proximity, but a binary tree computation like the summation of 2^k elements can nevetheless be performed in time k.

There is a latency, or propagation time, associated with many concurrent computations. The minimum latency may depend on the ensemble configuration, the algorithm, or the size of the problem to be solved relative to the size of the ensemble. For instance, consider forming the transpose of a N by N ($N=2^n$) matrix on a boolean cube. If the cube has 2^{2n} processors, then the transpose can be formed recursively in time $\log_2 N$. What's the time for forming the transpose on a K^2 cube; $(N/K)^2 \log_2 K$ or $(N/K)^2 + \log_2 K$? The latter applies if pipelined communication can be used. For a large cube the difference is significant. The answer to the question in the event of matrices with rows and columns embedded according to their binary encoding, or embedded according to a binary reflected Gray code, is that the lower time bound applies in the event processors can serve all their ports concurrently. Communication paths need only intersect at nodes, [11].

Exploiting data dependences, as well as, where appropriate, history dependencies sometimes yields very efficient algorithms. Exploiting data dependencies takes advantage of particular problem characteristics. Exploiting the history dependence allows a further improvement whenever repeated computations take place, and those computations are correlated. For data independent computations it is in principle possible to map the computations on to the nodes in the multiprocessor system at "compile time". For simple problems mappings that are optimal with respect to some criteria, like time, can be found at a small or moderate expense. However, finding optimal mappings for most problems is, in general, an NP-complete problem. For large problems such a computational complexity is infeasible, even in highly concurrent systems. It is often necessary to resort to heuristic mappings.

For data dependent computations good strategies for run time mappings of computations on to processors are needed. The algorithms for run time mapping are only allowed to rely on local information, given our computational model. Global information can only be gathered through a sequence of local communications over time. Our primary approach is to find heuristic algorithms that efficiently make use of the processors. Clearly, for a good balancing it is desirable to base decisions not only on "snapshots" of the load situation, but to include estimates of expected load changes, and communication. It is important to decide what data should be moved with a computation if a task is generated in one processor, but assigned to another. It is important to account for what other computations, present and expected future, will use the same data. The problem of load balancing is indeed a control problem.

To summarize we advocate the following approach to the systematic development of ensemble architecture algorithms whenever a compile-time mapping is feasible.

- Specify the desired function (computation) in common mathematical notation, i.e., in a space-time *independent* way.

- Generate a computation graph explicitly defining the data dependences and a partial order of computations

- Derive an ensemble architecture algorithm based on the computation graph, an architectural description, data constraints, and some optimization criteria. Algorithm/program transformation techniques are applied in this process.

- Generate code

This approach forms a basis for the development of effective compilers for ensemble architectures.

The use of algorithm transformation techniques may lead to the discovery of new, for ensemble architectures more efficient algorithms. Though the FFT and cyclic reduction algorithms are no new discoveries, we produced a FFT algorithm from a DFT algorithm by algorithm transformation techniques in the process of mapping DFT computations on to linear arrays, [13]. We also rediscovered cyclic reduction in the process of finding a maximally concurrent equation solver based on elimination. A computation graph corresponding for Gaussian elimination was first created, then a maximally concurrent algorithm sought, which turned out to be equivalent to cyclic reduction in the case of tridiagonal systems, [8].

References

[1] Brent R.P., Kung H.T.
On the Area of Binary Tree Layouts.
Information Processing Letters 11(1):44-46, 1980.

[2] Flynn M.J.
Very High-Speed Computing Systems.
Proc. of the IEEE 12:1901-1909, 1966.

[3] Gannon D., Van Rosendale J.
On the Impact of Communication Compleity in the Design of Parallel Numerical Algorithms.
Technical Report ICASE Report 84-41, NASA Langley Research Center, August, 1984.

[4] Gentleman W.M.
Some Complexity Results for Matrix Computations on Parallel Processors.
J. ACM 25(1):112-115, January, 1978.

[5] Gottlieb A., Grishman R., Kruskal C.P., McAuliffe K.P., Rudolph L., Snir M.
The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer.
IEEE Trans. Computers C-32(2):175-189, 1983.

[6] Hennessey J.L., Jouppi N, Baskett F., Gill J.
MIPS: A VLSI Processor Architecture.
In *VLSI Systems an Computations*, pages 337-346. Computer Sciences Press, 1981.

[7] Hennessey J.L., Jouppi N, Przybylski S., Rowen C.
Design of a High Performance VLSI Processor.
In *Proc. of the Third Caltech Conference on VLSI*, pages 33-54. Computer Sciences Press, 1983.

[8] Johnsson, S.L.
Gaussian Elimination on Sparse Matrices and Concurrency.
Technical Report 4087:TR:80, Caltech Computer Science Department, December, 1980.

[9] Johnsson S.L.
Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution Tridiagonal Systems of Equations.
Technical Report YALE/CSD/RR-339, Department of Computer Science, Yale University, October, 1984.

[10] Johnsson S.L.
Solving Banded Sytsems on Ensemble Architectures.
Technical Report YALE/CSD/RR-, Department of Computer Science, Yale University, December, 1984.

[11] Johnsson S.L.
Transposing Matrices on a Boolean cube.
December, 1984
unpublished note.

[12] Johnsson,S.L.
Cyclic Reduction on a Binary Tree.
Computer Physics Communications ():, 1985.

[13] Johnsson S.L., Cohen D.
Mathematical Approach to Computational Networks for the Discrete Fourier Transform.
1984
revised DRAFT of technical report.

[14] Leighton F.T.
Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.
MIT Press, 1983.

[15] Leiserson, C.E.
Area-Efficient VLSI Computation.
MIT Press, 1982.

[16] Mead C.A., Conway L.
Introduction to VLSI Systems.
Addison-Wesley, 1980.

[17] Mead C.A., Rem M.
Cost and Performance of of VLSI Computing Structures.
IEEE J. of Solid State Circuits SC-14(2):455-462, April, 1979.

[18] Patterson D.A.
Reduced Instruction Set Computers.
Communications of the ACM 28(1):8-21, 1985.

[19] Patterson D.A., Sequin C.H.
A VLSI RISC.
Computer 15(9):8-22, 1982.

[20] Preparata F.P., Vuillemin J.E.
The Cube Connected Cycles: A Versatile Network for Parallel Computation.
In *Proc. Twentieth Annual IEEE Symposium on Foundations of Computer Science*, pages 140-147. 1979.

[21] Reiter E., Rodrigue G.
An Incomplete Cholesky Factorization By a Matrix Partitioning Algorithm.
In *Elliptic Problem Solvers II*, pages 161-174. 1983.

[22] Rosenberg A.L., Snyder L.
Bounds on the Costs of Data Encodings.
Mathematical Systems Theory 12:9-39, 1978.

[23] Lutz C., Rabin S., Seitz C.L., Speck D.
Design of the Mosaic Element.
In *Proceedings, Conf. on Advanced research in VLSI*, pages 1-10. Artech House, 1984.

[24] Siegel H.J.
Interconnection Networks for SIMD Machines.
Computer :32-48, 1979.

[25] Snyder L.
Introduction to the Configurable Highly Parallel Computer.
Computer 15(1):47-56, 1982.

[26] Wang H.H.
A Parallel Method for Tridiagonal Equations.
ACM Trans. Math. Software 7(2):170-183, June, 1981.

[27] Wu C.-L., Feng T.-Y. (editors).
Interrconnection Networks for Parallel and Distributed Computing.
IEEE Computer Society, 1984.
Tutorial.