PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

The Physical Structure of Concurrent Problems and Concurrent Computers

G. C. Fox and W. Furmanski

Phil. Trans. R. Soc. Lond. A 1988 **326**, 411-444
doi: 10.1098/rsta.1988.0096

Phil. Trans. R. Soc. Lond. A **326**, 411–444 (1988) [411]
Printed in Great Britain

The physical structure of concurrent problems and concurrent computers

By G. C. Fox and W. Furmanski

Caltech Concurrent Computation Program, Mail Code 158–79, *California Institute of Technology, Pasadena, California* 91125, *U.S.A.*

We introduce a physical analogy to describe problems and high-performance concurrent computers on which they are run. We show that the spatial characteristics of problems lead to their parallelism and review the lessons from use of the early hypercubes and a natural particle-process analogy. We generalize this picture to include the temporal structure of problems and show how this allows us to unify distributed, shared and hierarchical memories as well as SIMD (single instruction multiple data) architectures. We also show how neural network methods can be used to analyse a general formalism based on interacting strings and these lead to possible real-time schedulers and decomposers for massively parallel machines.

1. Introduction

In this paper, we shall sketch a theoretical framework that will allow us to discuss the structure of both problems and computers. This will illuminate many issues in concurrent computation where the basic goal can be thought of as finding a suitable map from the problem to the computer. In §2 we shall summarize some of the 'experimental' data that have guided our theoretical development. This represents analysis of the approximately 100 applications developed on the hypercube at Caltech. In §3, we introduce a theory of complex systems and use it to discuss the spatial or data domain aspects of problem structures. These two sections are essentially a review of material covered in Fox (1983, 1984, 1985*a*, 1986*a*, 1988*b*), Fox & Otto (1984, 1986) and Fox *et al.* (1986*a*, *b*, 1988). In the short rather specialized §4, we describe an analytic approach to communication algorithms which will be derived by an automated procedure in §7. Sections 5, 6 and 7 represent rather new material generalizing our earlier work on spatial structure to that associated with temporal degrees of freedom. Conclusions will be found in §8.

2. Basic results and lessons learned from the use of the hypercube

(*a*) *The Caltech-concurrent computation program* (C^3P)

Although most of the paper will address general computer architectures, much of our experience comes from using the hypercube concurrent computer. These were developed at Caltech with the first major hardware, the Cosmic Cube with 64 nodes, being completed in 1983 by Chuck Seitz in the Caltech Computer Science Department. Starting in 1981, we have built up a research group, now called C^3P, whose emphasis is an application-oriented approach to fundamental issues in computation. We built two generations of hypercubes, the Mark II and Mark III, as part of C^3P to provide sufficient parallel computing resources to support

many users. We will finish this phase of our work in summer 1988 when the 128 node Mark IIIfp hypercube will be complete. This computer features a node shown in figure 1 which has two Motorola 68020's, a 68882 floating-point unit, which is enhanced with a secondary board on each node with a WEITEK XL accelerator giving over 10 megaflops (10 million floating point operations per second) performance per board. The total system will have half a gigabyte of memory and a peak performance of over one gigaflop. In figure 2, we show a commercial NCUBE where a board contains 64 single chip CPUs (central processing units), each with 11 communication channels and one half a megabyte of memory. The NCUBE system is architecturally quite similar to the Transputer systems discussed by Hey, May and Wallace (this symposium). Our 512 node NCUBE is now operational as is a 1024 node system installed in SANDIA's new computer science division. These two major machines will be our backbone resource for our research which is centred on the philosophy of 'Solution of real problems on real hardware with real software'.

We are phasing out the in-house hardware component of C^3P as parallel processing technology is being successfully transitioned to industry. Another crucial part of C^3P is the user community; currently some twenty research groups at Caltech's Campus and the Jet Propulsion Laboratory in various fields of computational science and engineering. C^3P provides these users access to parallel computing hardware and software as well as financial support for students and postdoctoral research fellows developing applications and algorithms. The research groups are motivated by both the future promise of parallel computing and the near-term possibility of using existing machines to solve major computational science projects. This production use of parallel computers is possible as we stress powerful machines and our philosophy ensures that we implement full problems and do not get misleading answers from incomplete implementations. The final component of C^3P is a central group to address the computer science and computer engineering issues. Here, for instance, we develop necessary systems software; an effort that is still and perhaps even more necessary with the commercial machines. We also develop fundamental algorithms and research issues such as those described in the rest of this paper. We have a growing interest in neural networks both for the assignment problems discussed in §§3 and 7 as well as for an attractive approach to parallel artificial intelligence.

Finally, we stress that even our theoretical work is closely based on the results of the users group where by now over 100 separate codes have developed. As shown in table 1, these cover a very wide range of algorithms and disciplines. For more general information we refer the reader to our recent C^3P annual reports (Fox 1986*b*), (Messina & Fox, unpublished results, C^3P-487 (1987)) and also the proceedings of the hypercube conferences (Heath 1986, 1987; Fox 1988*a*).

(*b*) *The architecture of concurrent supercomputers*

There are clearly very many (perhaps over 50) different parallel computer architectures being investigated worldwide. However, there are three important classes of machines that we shall investigate in this paper (Fox 1987; Fox *et al.* 1988; Messina & Fox 1987; Messina, unpublished results, C^3P-449 (1987)). These machines differ in their treatment of three important characteristics described below.

Architectural characteristic 1*: memory structure*

A parallel machine consists of many nodes or individual processors. In distributed-memory machines each node has its own local memory; in shared-memory machines a single large

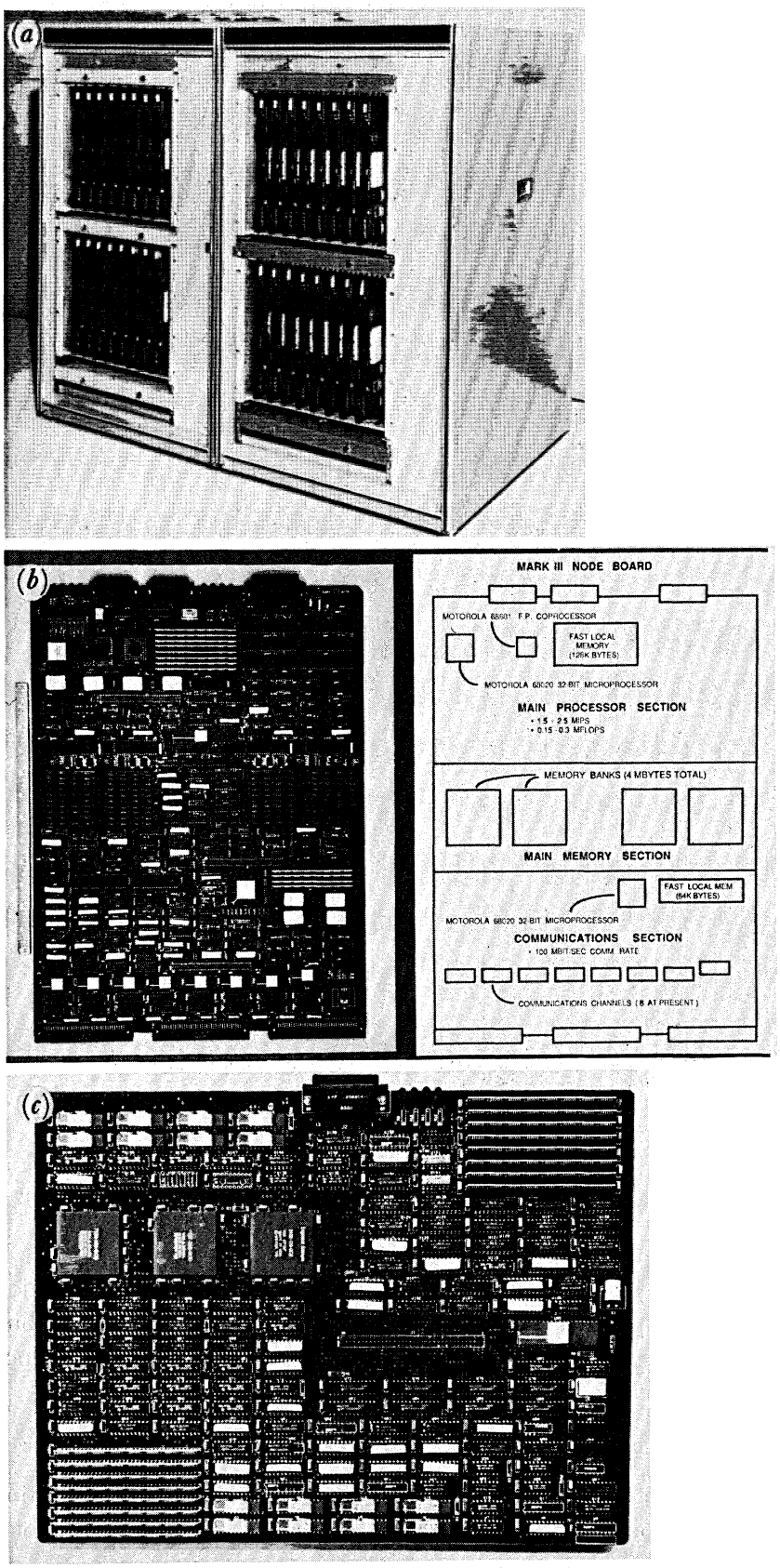


FIGURE 1. The Mark IIIfp hypercube designed and constructed at Caltech's Jet Propulsion Laboratory (JPL). (*a*) The basic 32 node package which can be extended up to 128 modes. (*b*) The dual 68020 based main node board. (*c*) The WEITEK XL chip set based secondary board.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

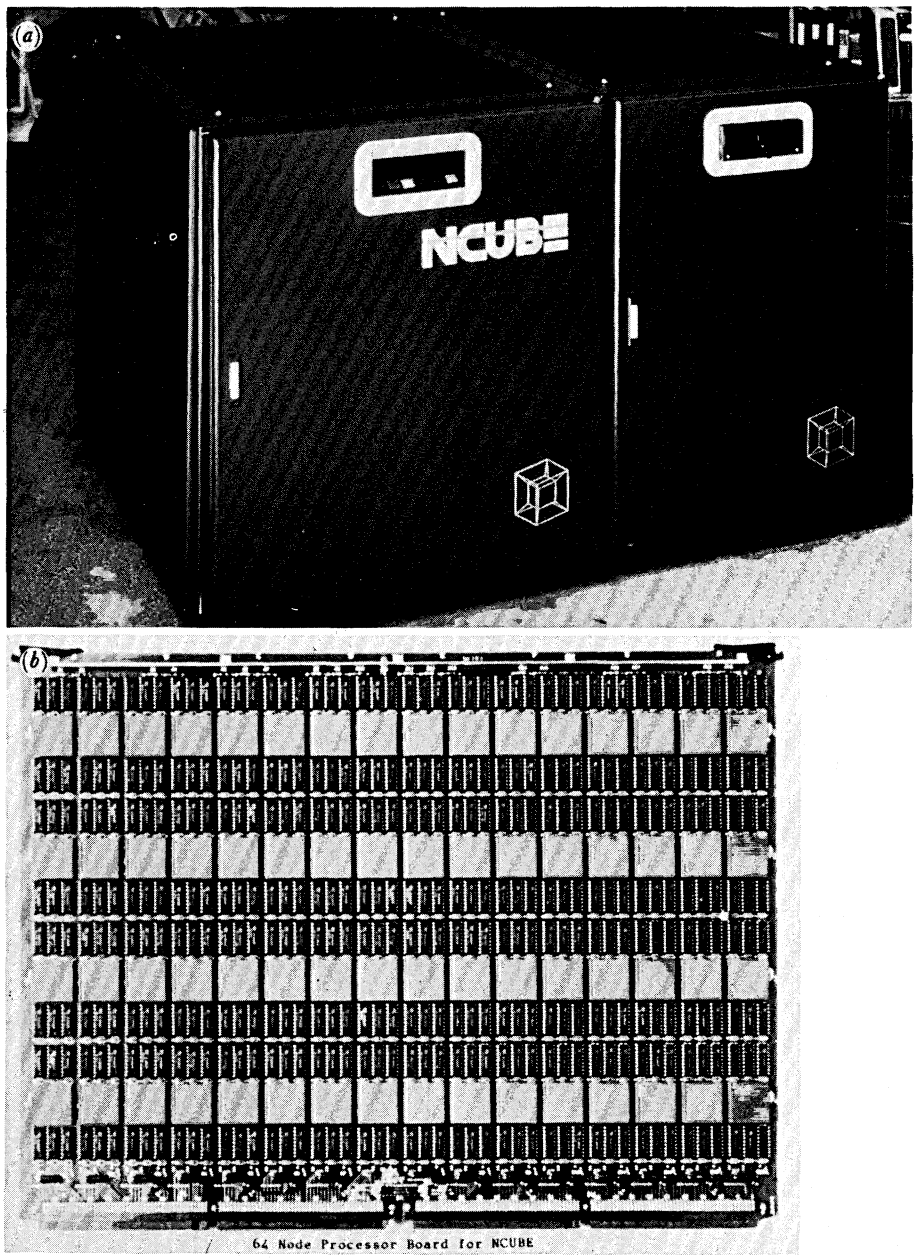


FIGURE 2. The commercial NCUBE/10 hypercube with (*a*) cabinet which can hold up to 1024 nodes configured as a 10 dimensional hypercube; (*b*) basic board containing 64 nodes.

global memory is directly accessible to all nodes. Although memory hierarchy does not directly affect the parallelism, practical high-performance computers usually exhibit a multilevel memory hierarchy with a relatively small fast memory acting as a cache or local memory to buffer data from a larger slower memory. We will discuss this in §5.

Architectural characteristic 2*: grain size*

This reflects the size of the node which is measured in terms of the computational capability of the node and its associated memory. Naturally, there is a continuous choice but we can

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

distinguish two very distinct possibilities. Large-grain size machines have a full function node and substantial memory, currently at least $\frac{1}{2}$ megabyte, capable of holding a complete program. Small-grain size nodes have incomplete nodes, typically single bit, with modest memory, currently less than about 8Kbytes per node.

Architectural characteristic 3*: control*

Here we distinguish SIMD (single instruction multiple data) machines where each node operates synchronously with common instruction stream; this is appropriate for small grain size systems. A second control possibility is MIMD (multiple instruction multiple data) where each node runs asynchronously controlled by a separate program, typically residing in a large-grain-size local memory or cached from a global memory.

These three attributes are currently combined in various ways but three choices dominate as shown below.

Architecture 1*: distributed-memory, small-grain-size SIMD; called hereafter SIMD.* This choice is found in two relatively old, the ICL DAP and Goodyear MPP, and two new machines, the Connection Machine from TMC (Thinking Machines Corporation) and the mini DAP from AMT (Active Memory Technology).

Architecture 2*: distributed-memory, large-grain-size MIMD; called hereafter multicomputer.* This choice is currently exemplified by the hypercube where we have already discussed the in-house Caltech machines and there are currently four commercial offerings, the AMETEK S14, the FPS T Series, the INTEL iPSC, and NCUBE Systems 4, 7 and 10. Another important set of machines of this architecture are built by MEIKO and others around the INMOS Transputer. We will later mention separately so-called hierarchical and homogeneous machines of this class to classify the memory structure on each node (Fox 1988*c*). The machines with high-performance vector units on each node, such as the Mark IIIfp shown in figure 1, naturally need hierarchical memory as described above.

Architecture 3*: shared memory, large-grain-size MIMD; called hereafter shared memory.* This is the dominant commercial architecture; largely because this conservative choice currently allows the easiest development of software and especially the use of existing codes with automatic parallelization (Frey & Fox 1988). Examples include the CRAY-2, CRAY-XMP, and ETA-10; the ALLIANT and its extension CEDAR from Illinois; and systems that are offered with reasonably large numbers of nodes, namely the ENCORE, SEQUENT, BUTTERFLY and RP3 from IBM Yorktown.

These are not the only choices; for instance, the GF11 from IBM Yorktown is large-grain SIMD and distributed memory. Further, there is important research on dataflow and other ideas which may lead to very different architectures from the three discussed above. Again there are hybrid designs combining features of the separate types in the simple classification introduced above.

(*c*) *Domain decomposition and the space-time structure of problems*

All current high-performance concurrent computers with many nodes have obtained their parallelism from what is called 'data parallelism' by Danny Hillis (1985, 1987). We can view any problem as an algorithm applied to a data set and data parallelism corresponds to processing separate parts of the data set at the same time. One usually calls this breakup of the data set as 'domain decomposition' and the parts processed independently can be called grains

TABLE 1. CURRENT PARALLEL ALGORITHMS AND APPLICATIONS WITHIN C^3P

number	status	name	lead personnel	report or HCCA3 abstract
		Biology and computation and neural systems (CNS)		
1	I	Structural simulations of neural networks using a general-purpose neural network simulator	J. Bower (f), M. Nelson, W. Furmanski, U. Bhalla, M. Wilson	(A) (C^3P-404, 405)
2	I	Back propagation algorithms for character recognition and computer games	C. Koch (f), E. Felten, J. Hutchinson, S. Otto	(A261)
3	I	Pattern recognition by neural networks on hypercubes	A. Ho, W. Furmanski	(A207)
4	I	Collective stereopsis	R. Battiti	(A16, C^3P-420)
5	B	Mapping the human genome	G. Fox (f), L. Hood (f), P. Messina	
6	B	Modelling complex neurons	W. Furmanski, C. Koch (f)	
		Chemistry and chemical engineering		
7	I	Integration of coupled sets of ordinary differential equations from chemical reaction dynamics	P. Hipes, T. Mattson, M. Wu, A. Kuppermann (f)	(C^3P-347)
8	B	Polymer simulations	H.-Q. Ding, W. Goddard (f)	(A262)
9	B	Concurrent optimization and dynamical simulation in chemical engineering	A. Skjellum, M. Morari (f)	
10	B	Quantum lattice system for high T_c superconductivity and Monte Carlo simulation	H.-Q. Ding, W. Goddard (f)	
		Engineering		
11	C	Ray tracing on the hypercube	J. Goldsmith (j), J. Salmon	(C^3P-295, 384, 403) (A73)
12	I	Plasma simulations on the Mark III	P. C. Liewer (j), R. W. Gould (f), V. K. Decyk (UCLA), J. D. Dawson (UCLA)	(C^3P-460) (108)
13	I	Vortex dynamics	A. Leonard (f), F. Pepin, K. Chua	(A131)
14	I	Synthetic aperture radar (SAR) analysis on the hypercube	J. Kim, G. Fox (f), G. Aloisio, N. Veneziani (Italy), J. Patterson (j), B. Zimmerman (j), C. Wu (j)	(C^3P-468)
15	I	Flux-corrected transport on the NCUBE	D. Walker, G. Montry (Sandia)	
16	I	Parallel-free-language hydrodynamics	R. Williams	(A189) (C^3P-424, 465)
		Geophysics		
17	I	Finite-element wave propagation	R. Clayton (f) (A264)	(C^3P-408)
18	I	Normal modes of the Earth	T. Tanimoto (f)	(C^3P-408)
19	I	Finite-element flow modelling	R. Gurnis, B. Hager (f), A. Raefsky, G. Lyzenga (j)	(A189) (C^3P-408, 463)
		Physics		
20	I	Lattice gauge theory with fermions on the hypercube	J. Apostolakis, C. Baillie, H.-Q. Ding, J. Flower, S. Otto, G. Fox (f), R. Gupta (Los Alamos)	(A184) (C^3P-450, 411)
21	B	Random lattice calculations	T. W. Chiu	(A183)

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

TABLE 1. (*cont.*)

number	status	name	lead personnel	report or HCCA3 abstract
22	C	Two-dimensional melting	M. Johnson (IBM)	(A47, C^3P-268)
23	I	Non-local path integral Monte Carlo for helium	S. Callahan, M. Cross (f)	(A177)
24	I	$N \log N$ algorithm for astrophysical particle dynamics	J. Salmon, M. Warren	(A164)
25	I	The hypercube for astronomical data analysis	P. Gorham, T. Prince (f)	(A215)
26	I	Multichannel Schrödinger equation	T. Barnes, D. Kotchan (Toronto)	—
		General algorithms and numerical analysis		
27	C	LU-decomposition of banded matrices and the solution of linear systems	D. Walker, T. Aldcroft, A. Cisneros, G. Fox (f), W. Furmanski	(A4) (C^3P-348)
28	C	Optimal matrix algorithms and communication strategies for homogeneous hypercubes	G. Fox (f), W. Furmanski	(C^3P-314, 329, 386) (A265)
29	I	Adaptive multigrid on the Mark III	E. Van de Velde	(C^3P-406, 447) (A159)
30	I	Finite element methods in coherent parallel C	E. Felten, R. Morison, S. Otto	(A56)
31	C	Communication strategies for network simulations	G. Fox (f), W. Furmanski	(C^3P-405)
32	C	A concurrent implementation of the prime factor algorithm	J. Kim, G. Fox (f), G. Aloisio, N. Veneziani (Italy)	(A6) (C^3P-468)
33	C	Concurrent tracking algorithms with Kalman filters	T. Gottschalk, I. Angus	(A186) (C^3P-387, 388, 398) (C^3P-478-481)
34	I	Concurrent α–β search techniques for computer chess	E. Felten, S. Otto	(C^3P-383) (A268)
35	C	Shift-register sequence random number generators on the hypercube	T.-W. Chiu	(A182)
36	B	Transaction analysis on the NCUBE	A. Frey, R. Mosteller (IBM)	—
37	I	Branch and bound algorithms	E. Felten	(A239)

Notes

status	report label	affiliation
I, in progress	(C^3P-XXX) C^3P document number	(f) Caltech Faculty
B, beginning	(AXXX) HCCA3 abstract number	(j) Jet Propulsion Laboratory
C, complete		

C^3P documents may be obtained from C^3P requests, Caltech Concurrent Computation Program, 206-49, California Institute of Technology, Pasadena, California 91125. HCCA3 refers to Third Hypercube Conference (Fox 1988*a*).

in a terminology compatible with that of §2*b*. As shown in figure 3, this form of concurrency corresponds to mapping the problem onto the computer.

$$\text{domain decomposition}: \text{problem} \xrightarrow{\text{map}} \text{computer hardware}. \tag{1.1}$$

This is a key idea because a major thrust of this paper will be to study this map in terms of the structure of the two spaces involved; the target space formed by the computer system and the initial space formed by the problem. In §3, we will formulate this study in terms of a general theory of complex systems.

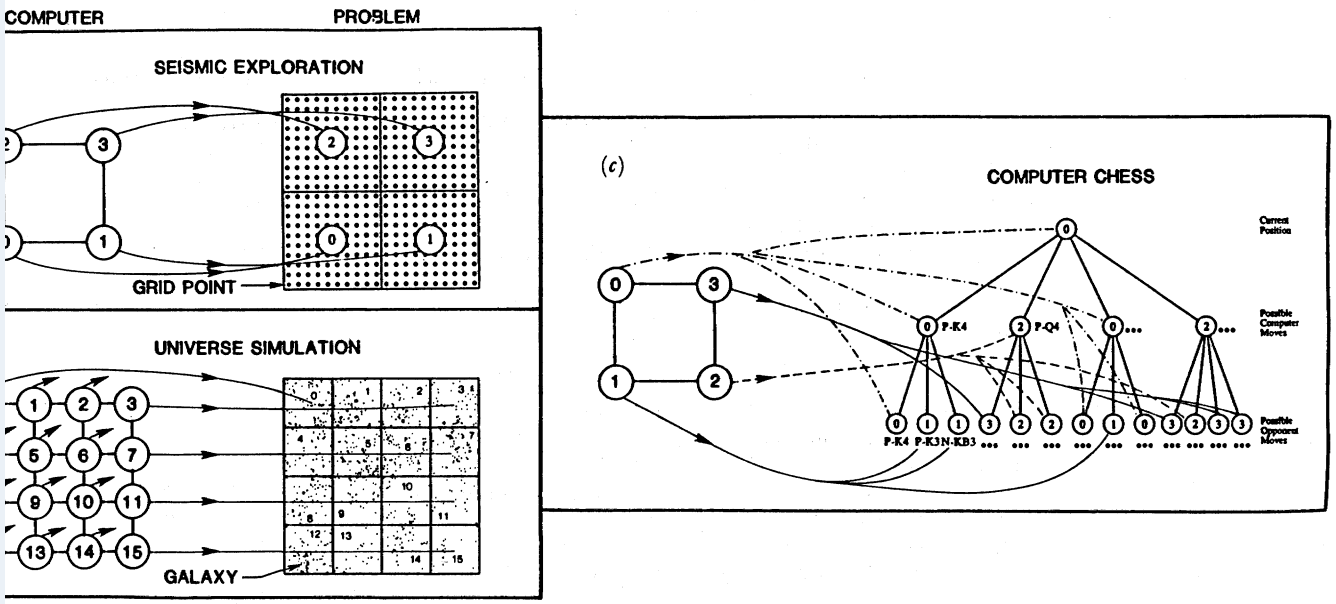


FIGURE 3. Concurrent computing as a mapping problem for (*a*) solution of the wave equation in seismic processing and (*b*) evolution of a collection of galaxies. (*c*) Searching the game tree in computer chess.

Attractive features of data parallelism are its generality and simplicity, and the potential in most problems for large amounts of parallelism; typical data sets have many degrees of freedom, say 10^6 for processing a $100 \times 100 \times 100$ mesh or 10^5 for the number of missiles to be tracked (Gottschalk 1987) by some all-encompassing defence system for the free world. Correspondingly, one can expect potential parallelism (speed-up) of order 10^6 or 10^5 on these two problems.

Note that the three architectures considered in §1*c* typically obtain parallelism from this same source although there are some important differences.

SIMD. Synchronous operation with distributed data and processing.

Shared memory. Only processing distributed with a global data set. We emphasize that most uses of this architecture do correspond to data parallelism even though the data are not necessarily distributed on the computer.

In the above, we have labelled the three architectures by the abbreviations introduced in §2*b*.

We can now introduce a general space-time picture. We will refer to the data domain associated with any problem as its 'space'. Consideration of examples will show that a given computational problem involves some sort of work or computation to be done on each element of the data domain. This work always has some, in practice discrete, label and we will call this label 'time' and refer to the temporal aspect of the problem. For the simulation of the real world, the data set is some part of the normal three-dimensional space and the computation is labelled by true physical time. Thus, our definition of 'space' and 'time' reduce to their conventional meaning in this case. Consider the following examples.

(i) *Full matrix inversion, multiplication, etc.* Here the set of matrix elements in the 'space' and 'time' corresponds to the label of eliminated or multiplied rows.

(ii) *Relaxation methods*. Here we are considering Gauss–Seidel or other iterative approach to

the solution of typical sparse matrix problems. Again the set of non-zero matrix elements is the 'space', and 'time' is just the iteration count.

We will later find in §§5–7 that the different computer architectures of §2*b* treat 'space' in a similar fashion but differ essentially in their treatment of the temporal structure of the problem.

We can also talk about the 'space'–'time' structure of the computer with 'space' corresponding to the distributed nodes of the computer and 'time' to elapsed time used by the computer.

We have used quotes above in defining the concepts of 'space' and 'time' for arbitrary problems and computers but we will drop these quotes hereafter and assume the reader understands the abstraction.

(*d*) *Performance of a homogeneous multicomputer*

Many years ago, we developed a simple performance model that has been shown to describe well the behaviour of our initial applications on the hypercube (Fox 1983, 1984, 1985*a*, *b*, 1986*a*, 1988*b*,*c*; Fox & Otto 1984, 1986; Fox *et al.* 1986*a*, 1988; Hey *et al.* 1988). This is only directly applicable to what we termed homogeneous machines in §2*b* (but its essential features apply to all the architectures in §2*b* and so we describe it here understanding that we will generalize it in §§5 and 6.

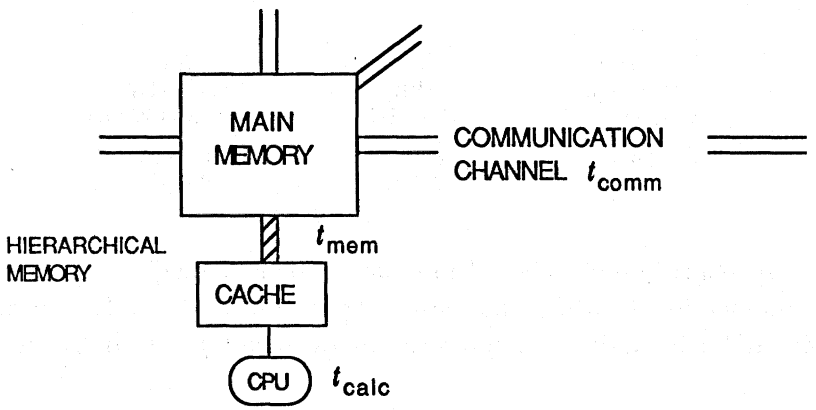


FIGURE 4. Three critical performance parameters $t_{calc}, t_{mem}, t_{comm}$ of a parallel and/or hierarchical memory computer.

In figure 4, we show a generic node of a multicomputer illustrating three parameters where here we will discuss the first two.

t_{calc}, the typical time it takes to perform a 32(64) bit floating point operation;

t_{comm}, the typical time it takes to transmit a single word between two nodes.

Values of these parameters are listed in table 2. As discussed, especially in Fox (1988*c*), Fox *et al.* (1988), (Frey & Fox 1988) and (Hey *et al.* 1988), these parameters are idealized and depend on the given application with the message size (amount of information transmitted at a time) being particularly important. We also define N as the number of nodes and n as the grain size or number of entities stored in each node. Figures 3(*a*) and 5 show a simple case where the space corresponding to the solution of a two-dimensional partial differential equation, say $\nabla^2\phi = -4\pi\rho$, has been split up into square subdomains where each processor

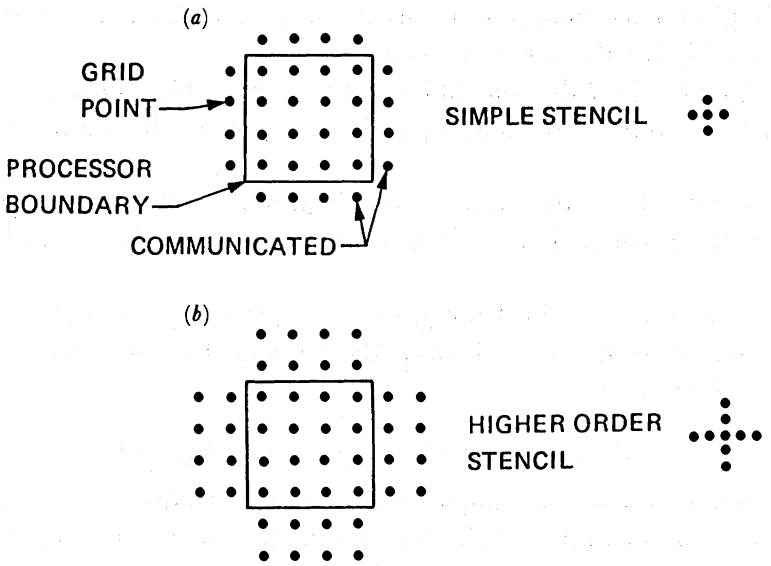


FIGURE 5. An illustration of (1.2) for a simple two-dimensional grid with nearest and next to nearest-neighbour interactions. (*a*) Calculation is $4nt_{\mathrm{calc}}$, communication is $4\sqrt{n}t_{\mathrm{comm}}$, $f_{\mathrm{c}} = (1/\sqrt{n})\, t_{\mathrm{comm}}/t_{\mathrm{calc}}$; (*b*) calculation is $8nt_{\mathrm{calc}}$, communication is $8\sqrt{n}t_{\mathrm{comm}}$, $f_{\mathrm{c}} = (1/\sqrt{n})\, t_{\mathrm{comm}}/t_{\mathrm{calc}}$. n Grid points per processor illustrated for $n = 16$.

TABLE 2. PERFORMANCE PARAMETERS FOR SOME EARLY HYPERCUBES

hypercube parameters	JPL Mark II (×8/5 for Cosmic cube)	JPL Mark III	JPL Mark IIIfp (with WEITEK board)	NCUBE
t_{calc}	25 μs	8 μs	0.08 μs	10 μs
t_{comm}	60 μs	2.5 μs	2.5 μs	13 μs

node has a $\sqrt{n}$ by $\sqrt{n}$ region of ($\sqrt{n} = 4$ in figure 5). For topologies like the hypercube that include a two-dimensional mesh, the basic algorithm only involves nearest-neighbour communication and one finds a communication overhead f_{C} given by the characteristic form

$$f_{\mathrm{C}} = \frac{\text{const.}}{n^{1/d}} \frac{t_{\mathrm{comm}}}{t_{\mathrm{calc}}}, \tag{1.2}$$

where d, in the example of figure 3*a* $d = 2$, is the dimension of the system defined generally in §3*b*. On the early machines where communication and calculation could not be overlapped, one then finds the speedup S is given as

$$S = N\epsilon, \tag{1.3}$$

with efficiency ϵ given by

$$\epsilon = 1/(1+f_{\mathrm{C}}). \tag{1.4}$$

Table 3 shows some measurements from the Cosmic Cube and Mark II hypercubes which verify the model of (1.2)–(1.4) (Fox 1985*b*; Fox *et al.* 1988). Typically, we have found that for reasonable values $t_{\mathrm{comm}}/t_{\mathrm{calc}} \lesssim 3$ (non-overlapped) or 15 (overlapped communication and calculation as in Mark IIIfp), the grain size effect (edge over area: $1/n^{1/d}$) reduces f_{C} to be negligible so that speedups of order 80% N are regularly obtained. In fact, the success of our

model has led to complacency and we no longer tend to perform the analyses that led to the early results of table 3; some recent performance analyses on NCUBE, Mark II and Mark III have been performed (Chen *et al.* 1988) for quantum chromodynamics (Flower 1987; Walker 1988), partial differential equations (Walker & Montry 1988), matrices (Aldcroft *et al.* 1988), (Hipes & Kuppermann 1988) and the non-binary fast Fourier transform (Aloisio *et al.* 1987).

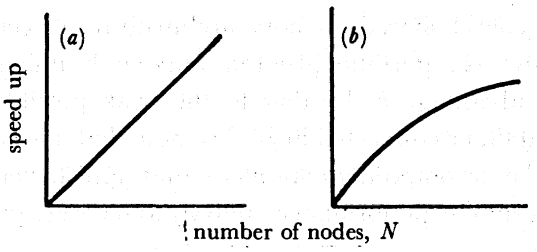


FIGURE 6. A comparison of the speedup for the cases of a fixed grain size (*a*) and a fixed problem size (*b*). In (*a*) the amount of problem in each node is fixed. The problem size is proportional to N; this is an interesting limit as we typically want more powerful computers to address larger problems. In (*b*) n gets small and communication and control dominate. The grain size is inversely proportional to N. Our measurements show speedup of approximately 80% of the number of nodes.

TABLE 3. MEASURED PERFORMANCE OF MARK I AND II HYPERCUBES

(Speed-up $= \epsilon N$; $\epsilon = 1-1/(f(n))$; $t_{\text{comm}}/t_{\text{calc}} = 2.$)

finite element or finite difference	two-dimensional, $\epsilon = 1-1/\sqrt{n}$	
	$\epsilon \sim 90\%$ for 10×10 subregion in each node	
	three-dimensional, $\epsilon = 1-1/n^{\frac{1}{3}}$	
	$\epsilon \sim 90\%$ for 1000 points in each node	
Monte Carlo	2D melting	$\epsilon \sim 1-5/\sqrt{n}$
	lattice gauge theory, quantum chromodynamics	$\epsilon \sim 1-0.1/n^{\frac{1}{3}}$
matrix algorithms	matrix inversion or multiplication	$\epsilon \sim 1-2/\sqrt{n}$
	matrix eigenvalues	$\epsilon \sim 1-4.5/\sqrt{n}$
sorting	$\epsilon \sim 70\%$ for 2000 items per node	

Figure 6 captures one important deduction. The speedup is linear in the number of nodes for fixed local parameters t_{comm}, t_{calc} and n; thus, for reasonable hardware (sensible ratio of $t_{\text{comm}}/t_{\text{calc}}$) one will get good speedup as long as the problem is large enough as measured by its grain size n. This critical grain size is application-dependent but for a given problem, independent of N, the number of nodes.

(*e*) *Software*

The characterization of concurrent computing by the simple map (1.1) is an over-simplification and in particular one should certainly consider the intermediate complex system formed by the software of the implementation (Fox 1988*c*; Frey & Fox 1988).

Concurrent computation can be more accurately abstracted as

$$\text{problem} + \text{algorithm} \xrightarrow{\text{map}} \text{software} \xrightarrow{\text{map}} \text{hardware}. \tag{1.5}$$

Indeed, we believe that software is the pacing issue in the development of parallel computing and much greater attention and more funding should be devoted to software. Only when

parallel machines can offer a good selection of real codes addressing major commercial problems, will there be a large market for our novel architectures.

However, we shall largely ignore the software issues here and our best reason for discussing other issues is to show how much progress has been made outside the software area and so highlight the need for corresponding developments in software!

We have in the above assumed an underlying model of computation involving independent processes communicating via messages. This is natural on distributed-memory machines and supported by the more general shared-memory architecture. A general approach that uses message passing is attractive as a portable programming environment (Fox 1988*a*; Walker & Fox 1988) and has the advantage of leading to the peak possible performance of shared memory machines (Fox 1988*c*) as discussed in §5. We note that one can show (Fox 1988*c*) that software environments, like decomposing compilers, that appear very different from message passing to the user have a similar performance analysis to message passing when implemented to give high performance.

We finish by noting that shared, distributed and hierarchical memories correspond to different trade offs between ease of construction, cost-performance, typical 'easy to get performance' and peak performance. These issues are discussed in detail in (Frey & Fox 1988), where we emphasize that software and its ease of development are crucial in evaluating different architectures.

3. Complex-systems and the particle–process analogy

(*a*) *A complex system*

Complex systems have been discussed in Fox *et al.* (1986*b*, 1988) and Fox & Otto (1986) and, for our purposes, consist of a dynamical collection of fundamental entities defined in the abstract space and time introduced in §1*d*. We have already introduced the important notion of mapping complex systems into each other with (1.1) and (1.5) and §3*c* we describe how temporally regular complex systems can be well described by and mapped into a physical system of particles. Initially we illustrate another aspect of the theory with a general definition of the system dimension in §3*c*.

(*b*) *The system dimension*

In Fox & Otto (1986) and Fox *et al.* (1988) we have introduced a general dimension d which measures the information flow within a complex system. Essentially we invert (1.2) to define d by

$$\left[\begin{array}{c}\text{information flowing}\\ \text{through a grain-boundary}\end{array}\right] \propto \left[\begin{array}{c}\text{information or}\\ \text{calculational}\\ \text{complexity within}\\ \text{grain}\end{array}\right]^{1-1/d_{\text{system}}}. \tag{2.1}$$

This definition ensures that d_{system} is equal to the geometric definition for a local algorithm defined in physical space of dimension d. In general, d is fractional as the work of Mandelbrot (1979) on fractals and we can give examples when d is either larger than or smaller than the geometric definition (Fox *et al.* 1988). Rent's rule for circuits which are typically built in two dimensions shows a dimension $d_{\text{circuit}} \sim 2$–3 (Fox & Otto 1986; Fox *et al.* 1988; Meindl 1987).

The simple $O(N^2_{\text{part}})$ long-range force algorithm, for say a set of N_{part} gravitating particles, can be analysed and shown to have $d_{\text{system}} = 1$ whatever the underlying geometric dimension. The definition (2.1) allows our basic formulae (1.2)–(1.4) for communication overhead to have general applicability and the $d_{\text{long-range force}} = 1$ result illustrates and quantifies in one example our early result that the multicomputer is not just good for local problems but is quite generally applicable. This was already seen in figure 5 which shows that f_{C} takes a similar form for local and next to nearest-neighbour interactions.

In Fox & Otto (1986), we conjectured the constraint

$$d_{\text{computer}} \geqslant d_{\text{problem}} \tag{2.2}$$

to guarantee the validity of the simple form (1.2) with its lack of N dependence and corresponding implication of figure 6 for the scaling of problems to large, in terms of number of nodes N, machines.

(*c*) *The particle-process analogy*

In a series of papers (Fox 1986*c*; Fox *et al.* 1986*b*; Fox & Otto 1986; Flower *et al.* 1988; Furmanski & Fox 1988*a*) we have introduced a very useful analogy between a physical system of articles and the set of concurrently executing processes in a parallel computation. We have also shown how to use simulated annealing and neural network methods in this analogy to find good explicit realizations of the map given in (1.1). We review this here in a context that allow its generalization in §7.

Consider a temporally regular problem; qualitatively this means a system that changes slowly with time and we will quantify this in §7. For such a problem, one can slice the computation at a particular instant of time and define an associated complex system defined by what are called the spatial properties of the problem in §2*c*. The complex system is a collection of entities connected by a computational graph. One can look at the problem at various grain sizes; depending on one's needs, these entities could be large processes each containing many, say $n \approx 1000$, degrees of freedom or alternatively one can focus in and consider the entities as the individual basic degrees of freedom. Figure 7 shows possible computational graphs with that on the left corresponding to the simple problem discussed in figure 5. However, more general cases with irregular connections and entities of different sizes (calculational complexities) can be treated. We now consider the analogy to a physical system

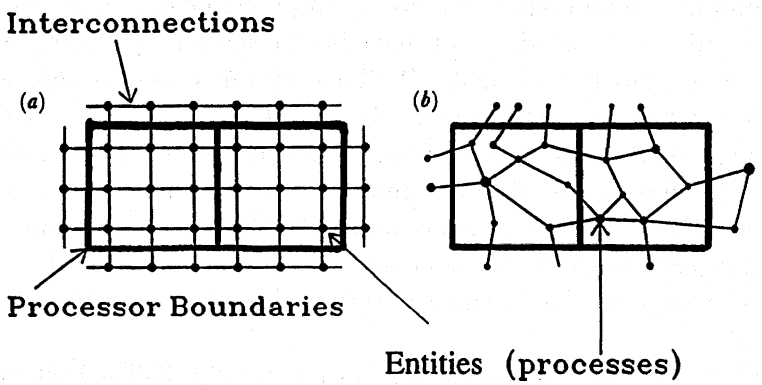


FIGURE 7. Two computational graphs decomposed onto a (fragment of a) parallel computer. (*a*) Local regular; (*b*) irregular.

where the particles correspond to the nodes of the graph, i.e. to the basic computational entities. We want to define an energy function E for this system such that E is a good approximation to the total execution time of the underlying problem on our target concurrent machine. Here we make a crucial but seemingly valid and indeed desirable simplification by defining

$$E \propto \sum_{i=1}^{N} C_i^2 \tag{2.3}$$

by using a least-squares sum rather than the 'true' form

$$E = \max_{i=1}^{N} C_i. \tag{2.4}$$

It is much easier to minimize (2.3) rather than solve the exact mini-max constraint implied by (2.4). In (2.3), C_i is the calculational complexity of particle i and includes the communication cost.

We now have simple physics analogy with particles moving in a discrete space defined by the nodes of the concurrent computer. These particles interact with two distinct two-particle forces.

(i) There is a long-range attractive force between any two particles linked in the computational graph. The associated term of the energy E is equal to the cost of communicating the necessary information between nodes to compute the consequences of this link. The form of this cost is clearly dependent on the particular form of the communication hardware. For the original simple machines Mark I, II and III, this cost is linear in the distance (in the underlying hypercube space) between any two particles linked by the graph.

(ii) The least squares form (2.3) contains a term $C_i C_j$ if and only if particles i and j lie in the same node. This is clearly interpreted as a repulsive hard-core potential between the particles which is zero if particles are in different nodes and positive if they are in the same node.

These ideas are shown in figures 8 through 10 which illustrate the solution of a finite element (FEM) problem by an iterative technique (Flower *et al.* 1988). The problem is defined by the irregular mesh of figure 8 which happened to have been produced by NASTRAN to describe a two-dimensional plate with a crack developing on the symmetry axis; figure 8 only shows the top half of the plate. Now our particles are the nodal points of the FEM with local interactions and figure 9 shows a naive equal area decomposition. This leads to grave load imbalance with many particles in a single node. Simulated annealing (Monte Carlo) methods easily led to the much improved decomposition of figure 10. Now each processor has an approximately equal number of particles and communication costs have been respected by placing connected (and as square as possible) regions in each node. We have also used neural network methods to address this and similar problems with success (Furmanski & Fox 1988*a*; Fox 1986*c*). Note that we are not trying to find the best solution but just a reasonably good one. Neural networks are good for such approximate optimizations and one can show that the time taken to find a decomposition is of order $M \lg M$ for a system of size M. We should also note that either the annealing or neural algorithms can be implemented concurrently and so both the problem and its decomposition can be performed concurrently on the same machine with comparable speedups of order N.

We are currently implementing these ideas (Kolawa *et al.* 1987; Koller 1988) as a dynamical load balancer and decomposer that will run under the multitasking operating MOOSE (Salmon *et al.* 1988) implemented on the INTEL iPSC hybercube. Here the user is responsible

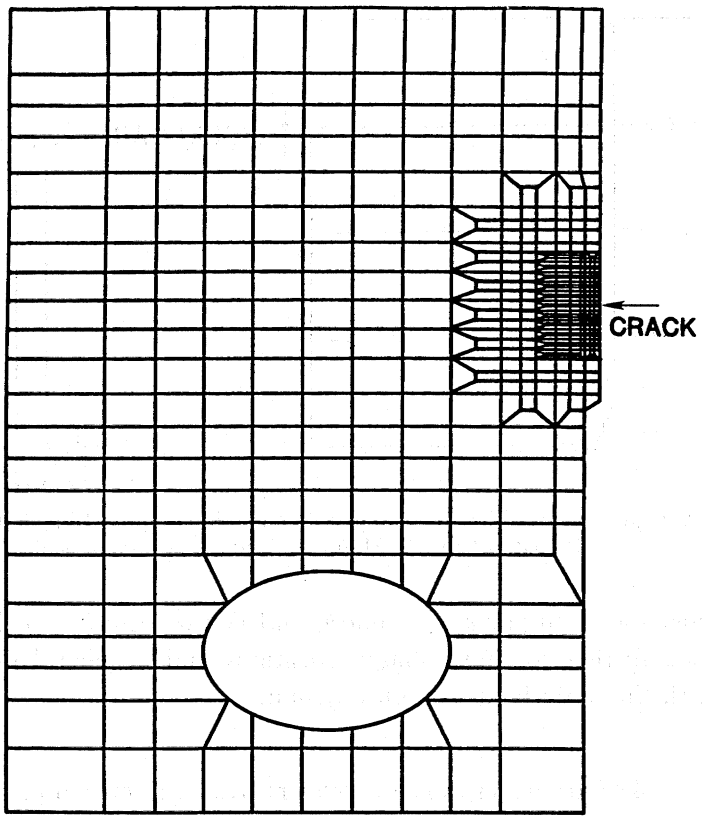


FIGURE 8. A 544 point mesh of nodal points generated by NASTRAN for a sample two-dimensional finite element problem (Flower *et al.* 1988). We intend to solve this by iterative (conjugate gradient) methods.

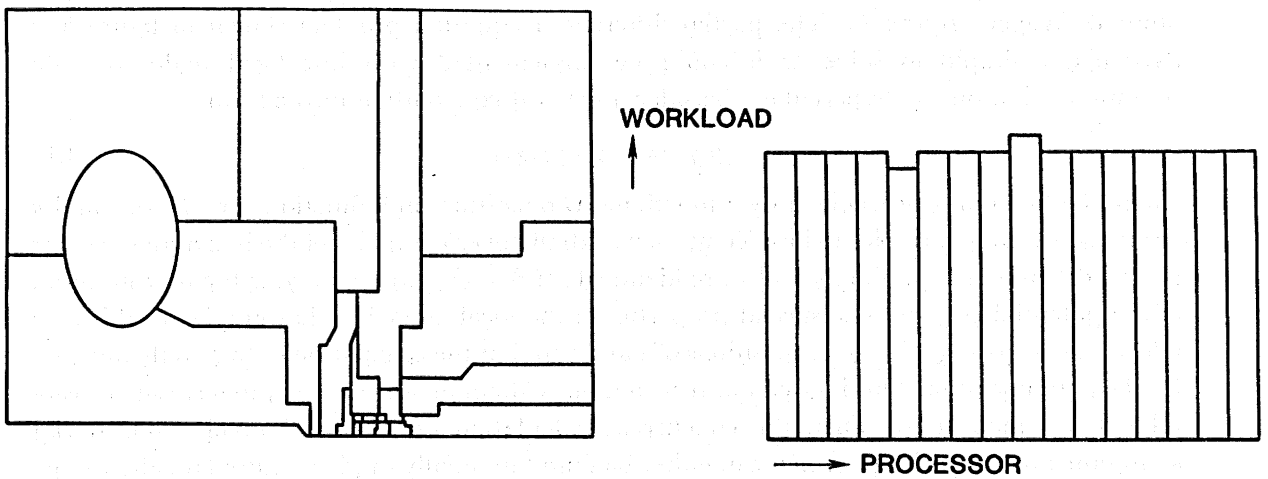


FIGURE 9. A simple two-dimensional equal-area decomposition onto a 16 node hypercube of the problem defined in figure 8. Minimum load, 32; maximum load, 36.

for defining the computational graph and the nodal weights C_i; the load balancer running itself as a concurrent collection of N processes then rebalances the load by moving processes as necessary between nodes. This dynamical implementation needs additional terms in E corresponding to the cost of moving processes but this is a straightforward extension (Kolawa *et al.* 1987).

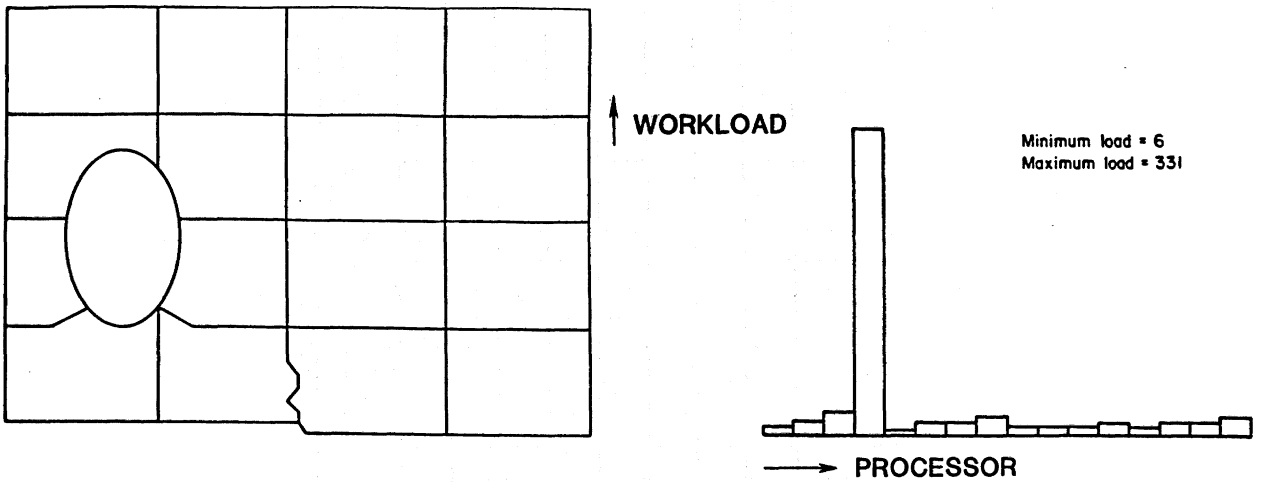


FIGURE 10. A single two-dimensional equal area decomposition onto a 16 node hypercube of the problem defined in figure 8. Minimum load, 6; maximum load, 331.

We have also shown in Fox *et al.* (1986*b*) and Fox & Otto (1986) how this physics analogy can be carried further to show phase transitions and a general definition of temperature associated with the underlying complex system.

4. COMMUNICATION STRATEGIES ON THE HYPERCUBE

This section is somewhat of an aside and is designed to provide background motivation for aspects of §7. It summarizes research that is described in more detail in Fox *et al.* (1988), Furmanski & Fox (1987*a*,*b*, 1988*b*, *c*), Ho & Johnsson (1986*a*, *b*), Johnsson & Ho (1988), Stout & Wager (1987*a*, *b*). The partial differential equation problem shown in figure 5 is particularly simple to solve as it could be implemented with just local node to node communication on the hypercube. Consider a general convolution of the form

$$g(y) = G(x, y; f(x)), \tag{3.1}$$

where the function $g(y)$ is given by a functional G operating on a function $f(x)$. Here x and y are typically both variables to be decomposed spatially over the nodes of the hypercube. In our partial differential equation, y and x are identical; $f(x) = \phi_{\text{old}}(x)$ and $g(y)$ is the new iteration of the potential $\phi_{nw}(y)$. The special properties of the local laplacian that gives G in this case allows one to arrange the decompositions of x and y so that the calculation of (3.1) only involves local communication. This is often not true and one cannot arrange x and y to have such easily related decompositions. Then the concurrent calculation of (3.1) will involve non-trivial communication algorithms which can either be found manually or given automatically by the techniques of §3*c*. A particularly good example of this is the calculation of the FFT in either its binary or especially non-binary form (Fox *et al.* 1988*a*; Aloisio *et al.* 1987). Matrix algorithms also give rise to these complex communication patterns and were the original motivation for our work on this subject (Furmanski & Fox 1986*b*, 1988). We have also shown how the simulation of various (neural) networks need different communication strategies depending on the richness and regularity of the network being simulated (Bower *et al.* 1987; Furmanski & Fox 1987).

A reasonably complete survey of the algorithms will be found in Fox *et al.* (1988*a*) and Furmanski & Fox (1987*a*, 1988*b*), and here we will briefly discuss three particular cases.

The fold algorithm

This is a very important algorithm (Fox unpublished results, C^3P-173; Cisneros unpublished results, C^3P-199), which is central to the optimal implementation of matrix vector multiplication

$$\boldsymbol{v} = M\boldsymbol{x} \tag{3.2}$$

on the hypercube. Here $\boldsymbol{x}$ and $\boldsymbol{v}$ are vectors and M is a full matrix. Another application of *fold* is to the simulation of the piriform cortex; this biological network has an essentially complete interconnect (Bower *et al.* 1988). An even simpler case for *fold* is the concurrent computation of what would be two nested *DO loops* in the equivalent sequential FORTRAN code; namely for large $N1$, $N2$ greater than about the number of nodes N, consider the calculation

```
DO 1 I = 1, N1                    (3.3)
A(I) = 0.
DO 2 J = 1, N2
2 A(I) = A(I)+B(I, J)
1 Continue
```

Here I and J are presumed to be variables that are both uniformly decomposed over the nodes of the hypercube. Problems like that of (3.3) will clearly occur quite often when sequential code is automatically decomposed onto a hypercube or other multicomputer.

The general problem is the calculation of many global sums. It is well known that a single global sum; for instance

```
X = 0.                            (3.4)
DO 1 J = 1, N2
1 X = X+C(J)
```

with J uniformly decomposed over the hypercube, is optimally calculated by forming a binary tree and naturally mapping this onto the cube. *fold* must take several ($N1$) such trees and carefully decompose them on the hypercube to minimize overlap and so minimize the resultant execution time. The best solution is known for arbitrary value of $N1$ and is shown in figure 11 for $N1 = N2 = N = 4$.

(*b*) *The crystal_router algorithm*

This addresses a generalized travelling salesman problem of the type faced every day by London Transport on the underground or the U.S. Military air command as it shuffles servicemen from one base to another. In its hypercube realization, one has a set of messages (passengers) on each node and each passenger has a ticket which specifies the number of one or more distinct destination nodes. This is the general data-shuffling problem and this is, for instance, the underlying communication problem in our concurrent missile tracker (Gottschalk 1987) and non-binary FFT (Aloisio *et al.* 1987) algorithms. The *crystal_router* addresses the case where each passenger has a ticket to a single node and it provides a deterministic algorithm that can be shown to be optimal when information is uniformly spread throughout the nodes. Each passenger (message) is sent on a geodesic or shortest path on the hypercube (Fox *et al.* 1988; Furmanski & Fox 1987*a*, 1988*b*).

(*c*) *The crystal accumulator algorithm*

This algorithm (Fox *et al.* 1988; Furmanski & Fox 1987*b*) addresses problems half way between those of §4 (*a*) and (*c*) and is typified by the solution of (3.2) where M is not full but rather sparse and irregular. Now the passengers are the contributions of $Mx|_i$ to a particular y_i from a given node and their ticket specifies the destination i where the sum y_i is to be accumulated. Now we get an algorithm like the *crystal_router* except that as shown in figure 11 for *fold*, passengers arriving at the same intermediate station (node) with the same ticket i are combined and only a single summed result is transmitted. Note that this is an idea analogous to that of combining switches for shared memory machines (Gottlieb 1987) but one needs more general combining operations than usually envisaged, in particular, floating point addition in the above example.

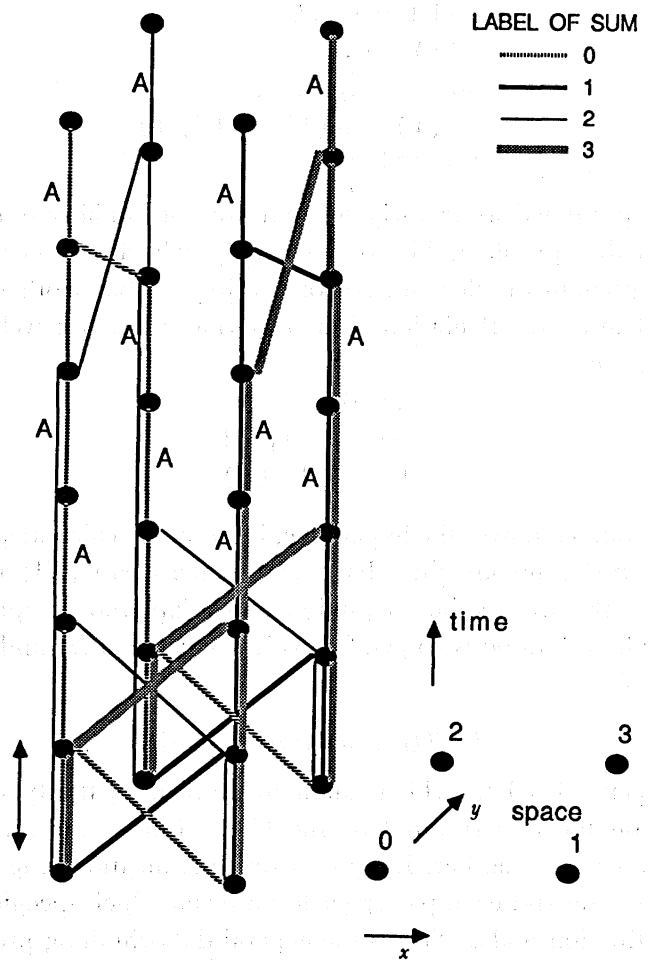


FIGURE 11. An example of the *fold* algorithm for $N1 = N2 = N = 4$. Here $N2$ is the number of entities in each sum. $N2 > 4$ would be illustrated by a similar figure with each node calculating a component of the sum over J in (3.3) local to each node. Then the figure shows how these precalculated components are summed. The double-headed arrow represents one time interval; this is assumed to be the same for calculation and communication. A means addition; otherwise wait for communication.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

(*d*) *Generalizations*

In §6, we will introduce powerful techniques that reproduce these algorithms in the cases we have derived them analytically. However, the new techniques will optimize the algorithms for cases of irregular traffic and for arbitrary architectures of the target concurrent computer.

5. The effects of hierarchical memory

(*a*) *The duality between memory structure of the computer and space-time structure of the problem*

Up to now, we have essentially addressed the spatial properties of problems and their mapping onto the spatial structure of the computer. In the remaining three major sections, we shall discuss some more recent results on the temporal aspects of problems. First, we will review the analysis of Fox (1988*c*) and Frey & Fox (1988) concerning memory hierarchy. We first note that this is very directly related to an analysis of shared memory architectures because high-performance machines of this shared memory class need a fast cache or local memory to buffer data from the slow shared memory. We will use the term 'cache' to refer interchangeably to a true hardware controlled cache or a user or software controlled fast local memory.

We have already decomposed problems into parts designed to minimize communication between them. This was the subject of §2 and is essentially all that is necessary to obtain good performance from machines like the NCUBE or Transputer arrays. We will use the same idea for hierarchical memories and divide the problem into parts (grains) so that each grain fits into the lowest level of the memory hierarchy. This is shown in figures 12 and 13 for shared-memory machines, hierarchical multicomputers and the simpler homogeneous multicomputer. In

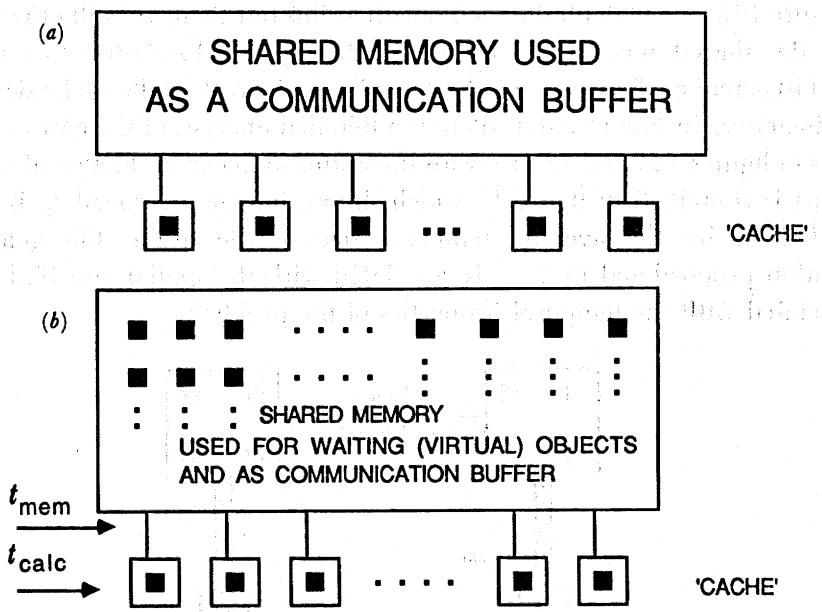


Figure 12. Shared or hierarchical memory computers showing the total problem divided into parts or processes that individually fit into the 'caches'. In (*a*), the total problem fits into the 'caches'; if the total memory needed equals the total memory in the 'caches', then there is one object per 'cache'. (*b*) Shows the general case where the processes are held in the (slow) main memory and need to be cycled through the 'caches'. A sequential vector processor is a special case of this. 'Cache' represents cache or local memory; ■, fundamental unit (process or grain), fits into lowest level of memory hierarchy.

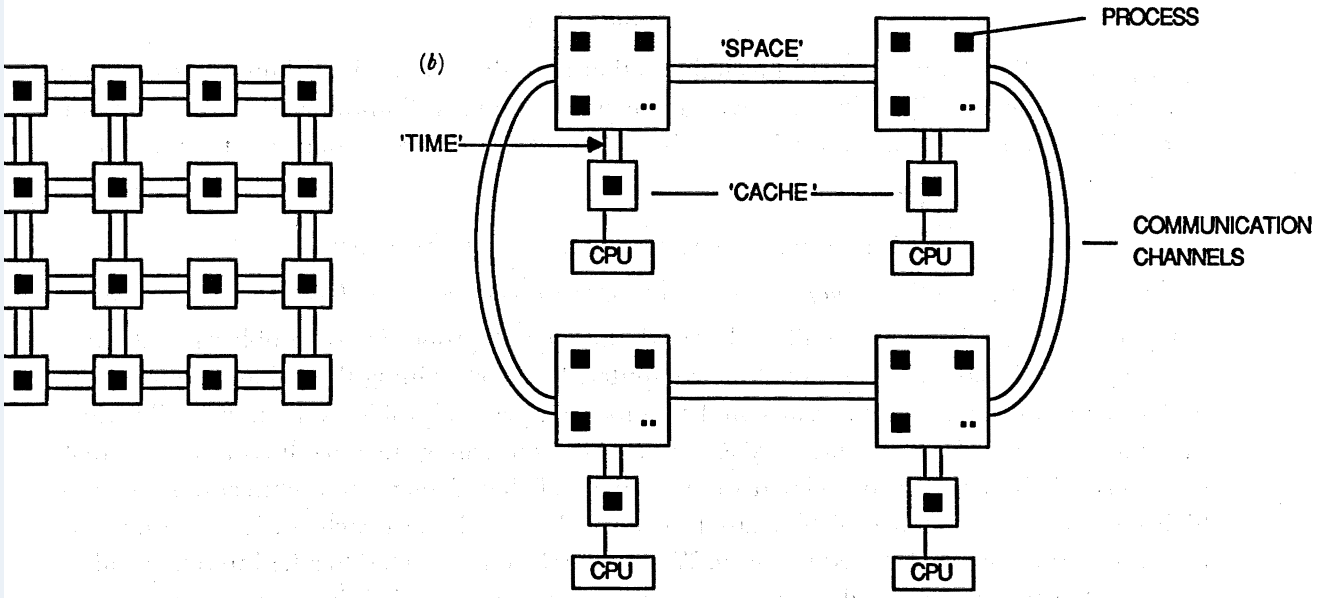


FIGURE 13. An illustration of distributed memory multicomputers. (*a*) Homogeneous multicomputer, purely spatial decomposition; (*b*) hierarchical multicomputer, spatial and temporal decomposition.

figure 12*a*, we see a simple special case where the total problem will fit into the caches when summed over the nodes. Then the global shared memory can just be used as a communication path and one can easily see that the overheads take the same form as (1.2) with t_{mem} replacing t_{comm}. Here t_{mem} was already shown in figure 4 and is defined as the time taken to read and write a word between the two levels of the memory hierarchy. However, in the general case, shown in figure 12*b*, one may fill the caches with grains but there are still other (virtual) grains waiting in the shared memory to be executed. Figure 13*b* shows how this looks for a hierarchical hypercube where we note the grain size is defined by the cache size and not by the total node memory. In Fox (1988*c*), there is a detailed analysis of the extra overhead needed for the cases of figures 12*b* and 13*a* to swap the grains in between the two memory levels. The essential idea is contained in figure 14 which shows that the overhead f_{H} is proportional to $t_{\mathrm{mem}}/t_{\mathrm{calc}}$ divided by the average temporal size of the grain. On general principles, communication proportional to t_{comm} is associated with the spatial and that proportional to t_{mem} is associated with the temporal properties of the problem.

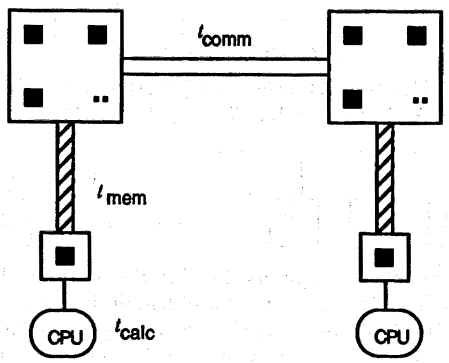


FIGURE 14. A summary of the overheads associated with hierarchical memory; *d* is the system dimension introduced in §3*b*.

(*b*) *A universal decomposition methodology*

These results leads to a universal decomposition methodology which we call the method of space-time blocking. For homogeneous multicomputers one only needs to divide the problem into local spatial blocks. This is a special case of a more general and difficult technique which divides the full space-time structure of the problem into blocks. This idea is shown in figure 15 for a very simple one-dimensional partial differential equation. One can also illustrate this idea with the BLAS-3 project introduced by Dongarra (Dongarra *et al.* 1987; Dongarra 1988). Their well-formed strategy of using matrix–matrix and not matrix–vector suboperations is precisely the implementation of space-time blocking for this problem. Although initially introduced for shared and hierarchical memory machines, the BLAS-3 idea is the correct basis for a universal library of full matrix operations across all the architectures of §2(*b*).

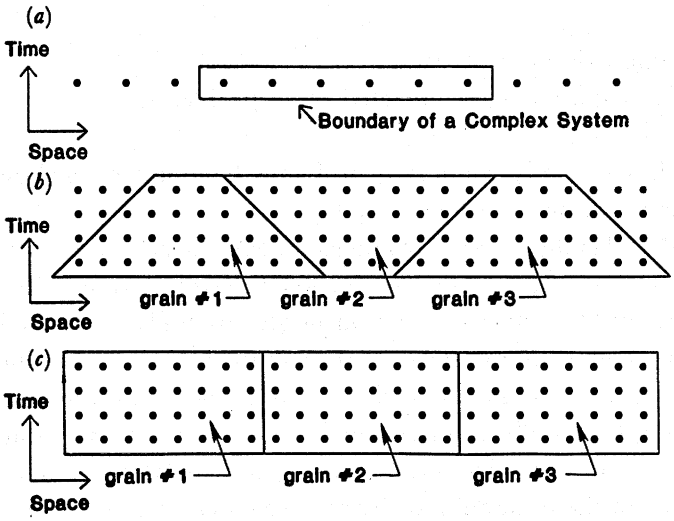


FIGURE 15. Decompositions for the concurrent one-dimensional wave equation. (*a*) A purely spatial blocking; a high edge:area ratio in the time direction. (*b*) Space-time blocking; a better edge:area ratio with modest communication. (*c*) Space-time blocking; a more practical space-time decomposition with more communication.

We see that all high-performance computers appear to need locality to achieve their performance. This is spatial locality for homogeneous hypercubes but more general architectures exploit locality in space and time. We remind the reader that we are using space and time in the general sense defined in §2*c* for arbitrary problems. We are currently investigating at Caltech the possibility of combining these general decomposition ideas with simple portable message passing software to generate a portable environment across many machines. Currently we are targetting a system spanning four machines: hierarchical shared memory ETA-10 and CRAY-2; the homogeneous hypercube, the NCUBE; the hierarchical hypercube and the JPL Mark IIIfp (Walker & Fox 1988).

We should note that we have only explicitly discussed MIMD machines above. The SIMD machines would be analysed similarly to the homogeneous multicomputers with the additional restriction explored in §5 that problems allow synchronous execution. We should also note a possibly controversial deduction from this section. Thus, the homogeneous machines, multicomputers like the NCUBE and SIMD, machines like the CM-2 are the easiest on which

to achieve high performance as one only needs a simple spatial decomposition. This is explored in (Frey & Fox 1988) where we contrast peak performance with 'good but not peak performance that is easy to get' where the hierarchical machines excel. I believe that it may turn out that homogeneous computers may supplant those of hierarchical memory as the future systems of choice.

6. What problems perform well on machines with many nodes?

We have recently realized that there are some simple characteristics of the problems that have performed well in our hypercube experiments and the many other applications on the DAP, Connection Machine, Butterfly and other machines with large numbers of nodes. We want to clarify the problem classes for which the optimistic speedup results sketched in figure 6 will apply. The two problem classes which have currently been shown to run well on MIMD machines are:

(*a*) loosely synchronous large problems;

(*b*) asynchronous but spatially disconnected large problems.

On the other hand:

(*c*) asynchronous but spatially connected problems have not been shown to scale and get good performance or machines with many nodes.

Let us examine the classifications introduced above. Large and spatially disconnected–connected describe the spatial properties of a problem. Large has already been quantified by the grain size analysis of §2*d* and its extension of §5. Spatially connected problems are like those shown in figure 7 with many non-trivial links in the computational graphs. Spatially disconnected systems are sometimes called 'embarrassingly parallel' and are typified by problems coming from data analysis, say in high-energy physics. Here one has a system with often as many as 10^7 entities which are individual scattering events recorded on tape. Parallelism is achieved in the data analysis by processing each event on a separate node. Such problems are asynchronous because each event is different and there is a wide variation of processing times. This prevents synchronization between nodes but this is irrelevant due to lack of connection, i.e. spatial disconnection, between the entities. This could be contrasted with event-driven simulations and some database problems where there are a large collection of asynchronous events which are, however, connected in a complex fashion. A similar situation occurs in our hypercube implementation of computer chess (Felten *et al.* 1986) where the α–β pruning relates the search of different lines of play. It is not clear if event-driven simulations and related problems can achieve large speedups although many ingenious methods are being explored (Jefferson & Sowizral 1985; Chandy & Misra 1987).

Finally, we need to return to the most important category (*a*) where we need to define loose synchronization. This essentially says that there is a single natural temporal label to cover the entire spatial domain. In a physical simulation, one evolves a system from time t to time $t+\delta t$. The different grains are labelled by the same time variable and can separately step in time without a complicated synchronization procedure. Another description of loosely synchronous is that of macroscopic synchronization. The grains synchronize every now and then i.e. at $t_0, t_0+\delta t, t_0+2\delta t,$ etc. but in between these synchronization points are free to calculate independently. Using the generalized definition of time given in §2*c* one will find the vast majority of successful highly parallel implementations are loosely synchronous. We emphasize

that we make no claims that this is a necessary condition, and indeed category (*b*) is a separate scaleable class, but it is a convenient classification for which our successes are intuitively clear.

We can now further strengthen the condition of loose synchronicity to that of microscopic synchronization. This leads to synchronous problems for which each grain can execute in lockstep. This is clearly the subcategory of problems for which the SIMD machines are appropriate. The division of problems into synchronous against loosely synchronous has not been carefully studied. It would appear that the loosely synchronous subcategory is at least for university applications quite a sizeable part (more than about 50%?) of the loosely synchronous category. We can expect that commercial applications to have greater irregularities and a lower percentage of synchronous problems.

7. A THEORY OF INTERACTING STRINGS AND ITS APPLICATION TO COMPUTATION

(*a*) *The string formalism*

In §3 we introduced a particle process analogy which we said rather vaguely was appropriate for temporally regular problems. This is easily defined by illustrating the two classes of problems for which the analogy fails. The first is the general communication algorithms of §4 in which information flows on a fine timescale through the computer network. These are usually loosely synchronous and indeed often synchronous problems but they are microscopically dynamic. The other major category for which the formalism of §3 fails is the very interesting asynchronous spatially connected problems discussed in §6. Event-driven simulations and real-time control of, say, robots fall into this important class. Thus, we see that the results of §3 can be applied to loosely synchronous, microscopically static problems and the goal of this section is to generalize these ideas to the problems with asynchronous or microscopically dynamic temporal properties. As usual, we use the label temporal in the sense of §2*c* to describe the computational label of the problem.

The natural formalism for time dependent graphs seems to be that of 'strings' or 'world lines' rather than the 'particles' used in §3. In the routing case of §4*b* the string formulation is particularly simple: the problem reduces to deriving the set of strings or particle trajectories with fixed end-points and the mini-max length constraint. The detailed solution for a given concurrent architecture should be consistent with the specific hardware characteristics like memory per node capacity, channel bandwidth, etc. More generally, strings may interact by coalescing or branching; they may be created and annihilated.

We can also consider the string language for the *fold* problem of §4*a* which mathematically calculates concurrently

$$y_i = \sum_j v_i^{(j)}, \tag{6.1}$$

where, as in §4, i and j are both distributed over the nodes. We only know the optimal algorithm analytically for special underlying machines topologies, in particular the hypercube, and so a general approach to even this simple problem is of interest for other architectures and irregular distributions of $v_i^{(j)}$. In the string formulation, one could try to abstract the essential elements of the known analytic algorithms for (6.1) in the form of some more general 'string dynamics' by introducing an attractive force between strings $v_i^{(j_1)} v_i^{(j_2)}, j_1 \neq j_2$ (to minimize routing for components of the same sum), 3-particle vertices corresponding to forming partial

sums and the repulsive force between different strings $v_{i_1}^{(j_1)} v_{i_2}^{(j_2)}$, $i_1 \neq i_2$ (to load balance the summation trees).

In general, the interaction formulae in vertices may be much more complex (non-commutative) and time-dependent by itself. For instance, in an event-driven simulation, the update of a given process $P(P \rightarrow P')$ may be accomplished only after receiving some additional information I from some other process P_0, which would correspond to time-dependent vertices $P_0 \rightarrow P_0 + I, P + I \rightarrow P'$, connected by the string I corresponding to the message.

Taken in its full generality, the language of interacting strings is as complete and indeed as intractable as the 'general theory of computation'. Indeed, associating string segments with the processor MOVE or LOAD operations and vertices with all elementary ADD, MULT, SHIFT, etc. operations, one can map any computer program (and hence, any problem) onto the system of interacting strings defined by the registers and memory world lines.

Although the problem of concurrent decomposition, when addressed in this most general 'atomic' form seems to be highly unrealistic, the fundamental aspect of the string language for distributed computation is, nevertheless, worth stressing. We will need to specialize the application or approximate the general formalism to generate a theory of practical value. Our aim in the remainder of this section is to address the general issue of the 'string dynamics' and to start quantifying it for the simplest instances of dynamic computation.

In figure 16, we show how the problem of §3 is viewed in this new formalism. Message strings are passed between two given processes represented by strings i_1 and i_2. However, the microscopic static property ensures that i_1 and i_2 are essentially fixed or rather change slowly with respect to the natural timescale of the computer. We see that one can generally re-interpret the message strings as a force which we can amusingly consider having as its quantum, the basic message packet.

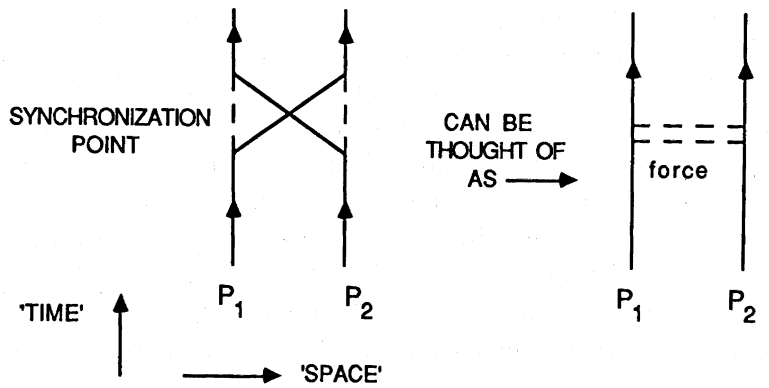


FIGURE 16. The microscopically static strings and their reduction to the particle picture of §3. P_1 and P_2 label strings thought of as particles.

(b) *The strings as dynamical variables*

Our basic strategy is to formulate the computational problem in the string language, to construct an appropriate cost–energy function in string variables and to derive the optimal arrangement by minimizing the string functional. The details depend on both the minimization technique and the string representation.

The combinational optimization typically reduces to the problem of satisfying the set of

constraints imposed either in a rigorous analytic form, like in the simulated annealing approach (Flower *et al.* 1988), or in the neural network (Furmanski & Fox 1988*a*) weak form just as for the simpler problems in §3. In the neural method, the search is performed over much broader phase space and the proper syntax is achieved only at the fixed point of the network. In consequence, many unwanted minima of the cost function are smeared over and the reasonable results can be obtained without the expensive annealing procedure. For the string, this distinction is even more sharp because the string itself is usually defined by a set of constraints. In the neural approach, strings appear as 'fuzzy' objects without definite localization or orientation until the stationary point is reached. In this sense, our neural approach is very similar to the method used by Hopfield & Tank (1986) for the 'travelling salesman' problem.

The advantage of neural net techniques is that one can satisfy simultaneously a large set of constraints by means of a relatively simple local algorithm, admitting both an analogue hardware implementation and concurrent implementation on conventional parallel machines.

In the rigorous strictly digital approach, one would work with explicit string representation in all stages of the algorithms, constructing the usual local moves in the string space by acting with the 'plaquette' operations as illustrated in figure 17. This, when applied with the conventional Metropolis algorithm, may result in very accurate solutions but can be expected to be much slower than the neural network approach.

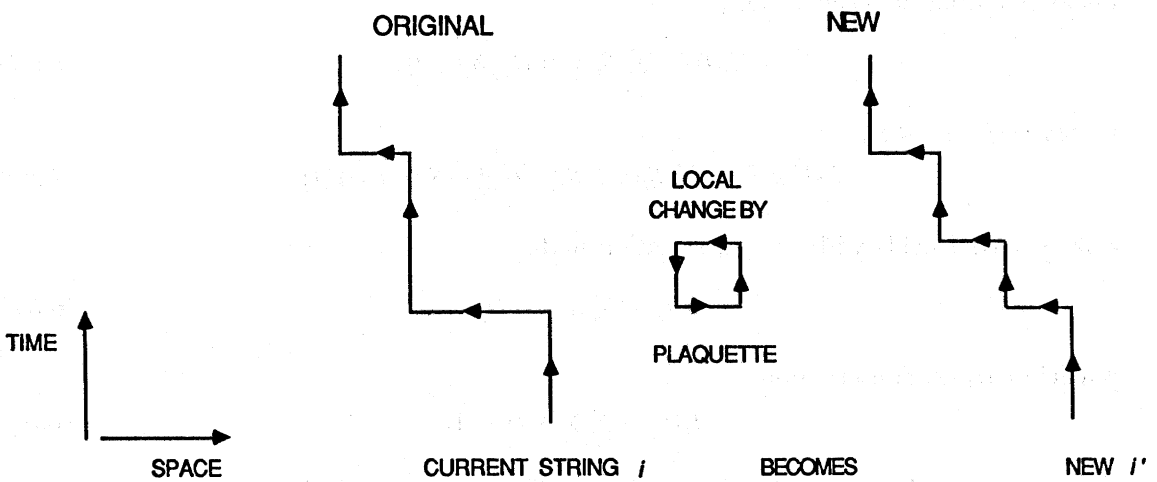


FIGURE 17. The Monte Carlo or Metropolis approach to the computational string dynamics.

Below, we discuss typical energy functions in string variables. It will be convenient to use two complementary representations of a string:

(*a*) global representation, where the string is given by a set of adjacent space-time locations,

$$S_i(x,t); \tag{6.2}$$

(*b*) local representation, where the string is given by a set of moves between adjacent points,

$$m_i(x,t;y,t+1). \tag{6.3}$$

In both cases, i labels a given string or process, x is the node number, t is the discrete time label and $S = 0, 1 (m = 0, 1)$ depending on whether or not the string i is at point x at time t (moves from x to y at time t.)

We will restrict ourselves to a simple multicomputer and denote by A_{xy} the arcs (links) of the machine topology, i.e. $A_{xy} = 1$ if nodes x, y are connected and $A_{xy} = 0$ otherwise (we take $A_{xx} = 1$). Within this topology, we define the metric D_{xy} as the minimal numbers of steps (channels), necessary to move from x to y.

The variables S, m, A are related as follows:

$$\sum_{xy} A_{xy} S_i(x,t)\, m_i(x,t;y,t+1)\, S_i(y,t+1) = 1. \tag{6.4}$$

The above string constraint is automatically satisfied by 'plaquette' moves, whereas in the neural method it enters as a syntax enforcing term to the cost or energy function.

Below, we present a few typical characteristics of a computation, expressed in the string variables, and useful for building the energy functions.

String length,

$$L_i = \sum_{xyt} A_{xy} S_i(x,t)\, S_i(y,t+1); \tag{6.5a}$$

load per node,

$$W(x,t) = \sum_i S_i(x,t); \tag{6.5b}$$

channel congestion,

$$C_{xy}(t) = \sum_i A_{xy}\, m_i(x,t;y,t+1); \tag{6.5c}$$

linear potential between strings,

$$F_{ij}(t) = \sum_{x,y} S_i(x,t)\, D_{xy}\, S_j(y,t); \tag{6.5d}$$

vertex $i+j \to i (i<j)$,

$$MV(x,t) = \sum_{i<j} S_i(x,t)\, S_j(x,t)\, [1 - S_j(x,t+1)]; \tag{6.5e}$$

sink (external field) with x^* = destination node,

$$E_i(t) = \sum_x S(x,t)\, D_{x,x^*}; \tag{6.5f}$$

particle (string) conservation,

$$P_i(t) = \sum_x S_i(x,t) = 1. \tag{6.5g}$$

Thus a possible string action for the routing problem of §3*b* could look as follows:

$$\begin{aligned} E_{\text{routing}} = {} & A \sum L_i \quad \text{(minimal length)} \\ & + B \sum_{xt} W^2(x,t) \quad \text{(load balance)} \\ & + C \sum_{xyt} C_{xy}(t) \quad \text{(channel balance)} \\ & + D \sum_i E_i(t) \quad \text{(destination)} \\ & + \sum (P_i(t) - 1)^2 \quad \text{(conservation)} \\ & + \text{string syntax enforcing terms.} \end{aligned} \tag{6.6}$$

The action for vector sum *fold* of §3(*a*) would contain additional linear force terms F_{ij}, attractive within a given tree, repulsive for different trees and the vertex terms $V(x,t)$. In the latter, the higher index string S_j is annihilated whenever two strings $S_i, S_j\, i<j$ meet in the same

node. For this reason, the particle conservation term should be relaxed, for example, to the following form of 'string continuity'

$$\Delta E = \sum_i (P_i(t) - P_i(t+1))^2, \tag{6.7}$$

which allows for string annihilation but ensures this is not done 'unnecessarily' but penalizes in the energy function changes in the numbers of strings.

(c) *Real-time neural controller*

In the approach discussed in the previous section, strings appear 'as a whole', i.e. the output of the minimization procedure consists of the complete set of all strings in the global time interval. This strategy is useful for dynamical problems with static or slowly varying computational goal, i.e. when the global string layout can be planned independently on the results of intermediate computations. For difficult time-dependent problems such as games, event-driven simulations, etc. such a global strategy is useless because the communication routes have to be continuously adjusted according to the varying environment.

A possible approach for such problems is to design a real-time controller, capable of planning the forthcoming action for next few 'moves' ahead. The neural net technique, with its ability of providing a quick rough guess, seems to be particularly appropriate here.

We will illustrate this method on the example of the routing problem again, but now with possibly time-dependent destination nodes $n^* = n^*(t)$. For definiteness, we will use hypercube topology. It will be convenient to work with the string move variables m_α where $\alpha = 0, 1, \ldots, d-1$ is the communication channel (cube addressing bit) of the d-dimensional hypercube. The limited goal of the following real time algorithm is to predict the correct set of moves m_α one time step ahead, given the current string positions $n(i, t)$ and the currently expected destination nodes $n^*(i, t)$.

One can imagine the complete algorithm as composed of a set of consecutive computation-control time slices in which processes move to the destination nodes but their location might be changed during each computational slice.

The simplest solution is provided by the *crystal_router* algorithm, discussed in §3*b*. The performance of this which is optimal for homogeneous distributions, however, deteriorates substantially in the non-uniform case. Thus in the pipe-dominated configuration (all processes have the common initial and final nodes), the *crystal_router* deteriorates by a factor $\log_2 N$ compared to the best algorithm.

In the general-purpose, heavily used routing algorithm, one would like to achieve a general trade-off between the Hamming distance (for a hypercube) minimization and fan-out–load balancing.

The action for a simple neural router algorithm could look as follows:

$$E_{\text{neural router}} = A \sum_{i,\alpha} \sigma_\alpha(i)\, m_\alpha \sigma^*_\alpha(i) + B \sum_{i,j} \prod_\alpha \{1 - \sigma_\alpha(i)\, \sigma_\alpha(j)\, m_\alpha(i)\, m_\alpha(j)\}, \tag{6.8}$$

where $\sigma_\alpha, \sigma^*_\alpha$ are the hypercube bits of the current and destination nodes $n_\alpha(i)$ and $n^*_\alpha(i)$:

$$n_\alpha(i) = \sum_\alpha 2\alpha \left(\frac{1+\sigma_\alpha(i)}{2}\right), \quad n^*_\alpha(i) = \sum_\alpha 2^\alpha \left(\frac{1+\sigma^*_\alpha(i)}{2}\right). \tag{6.9}$$

The energy is to minimized as a function of the move variables m_α, whereas $\sigma_\alpha, \sigma^*_\alpha$ play the role of time-varying external fields, taken at the current time slice.

The first term in (6.8) attracts to the destination node n^*, the second term constructed in a similar way as in the static algorithm of Furmanski & Fox (1988*a*), takes care of the load balance after the current move.

The evolution equation implied by (6.8) is:

$$m_\alpha^{\text{new}}(i) = T\left\{A\sigma_\alpha(i)\,\sigma_\alpha^*(i) - B\sum_{j(i)}\sigma_\alpha(i)\,\sigma_\alpha(j)\,m_\alpha^{\text{old}}(j)\right\}, \tag{6.10}$$

where the process $j(i)$ shares the current node with the process i. T is the usual nonlinear transfer function in neural networks (typical functional forms are $T\sim\tanh$ or $T\sim\Theta$). The equation $m_\alpha\approx 1(-1)$ indicates if the move is to be done (or not done). As seen from (6.10) the neural algorithm simply reduces to the *crystal_router* for the choice $B=0$. The additional terms for $B=0$ assures the appropriate fan-out for inhomogeneous initial distribution.

In a similar way, we can construct the neural version of the *fold* or, more generally, of the *crystal_accumulator* algorithm. As discussed in §4*a*, *c*, the string structure for this problem is given by the set of trees, distributed over the processors and rooted in appropriate destination nodes.

We will associate the indices i,j with the processes (strings) as in the *neural_router* example above and we will label individual trees by the index τ so that, for example, $\tau(i)$ is the tree label of the process i. All processes i with a given value $\tau(i)$ should be directed to the common destination node n^* and the corresponding strings should be combined, i.e. any two strings i,j such that $\tau(i)=\tau(j)$, arriving at some common intermediate node should form a vertex as in figure 11.

Because the process number is not conserved now, some consistent time-dependent labelling mode should be chosen. The simple scheme is to always annihilate the process with the higher index, say j for $i<j$ and to retain the lower index, here i, using it to label the combined process (see (6.5*e*)).

To enforce the combining action, we can add the attractive force between any string pair i,j belonging to the common tree ($\tau(i)\neq\tau(j)$), while retaining the repulsive, load-balancing force as in (6.10) for strings from different trees ($\tau(i)\neq\tau(j)$). The resulting network evolution equation looks as follows:

$$\begin{aligned} m_\alpha^{\text{new}}(i) = T\Big\{ & A\sigma_\alpha(i)\,\sigma_\alpha^*(i) \quad \text{(sink–destination node)} \\ & -B\sum_{j(i):\tau(i)\neq\tau(j)}\sigma_\alpha(i)\,\sigma_\alpha(j)\,m_\alpha^{\text{old}}(j) \quad \text{(string repulsion–load balance)} \\ & +C\sum_{j(i):\tau(i)=\tau(j)}\sigma_\alpha(i)\,\sigma_\alpha(j)\,m_\alpha^{\text{old}}(j) \quad \text{(string attraction–combine)}\Big\}; \end{aligned} \tag{6.11}$$

as before, $j(i)$ label strings sharing a node with i at the current time.

(*d*) *Numerical results*

We performed some simple numerical experiments with the network formulae in (6.10) and (6.11). The preliminary results, presented in figures 18–21 and discussed below are mainly of an illustrative character; more work is required to rigorously quantify the network performance and its dependence on parameter values.

The *neural_accumulator* network was tested for the 16-node hypercube. The non-zero elements of the vectors to be summed over (combined) were sampled randomly with some fixed overall

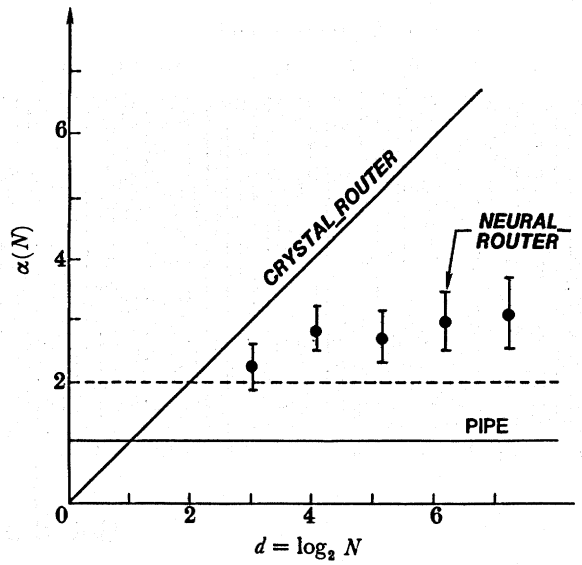


FIGURE 18. The communication time for diagonal transfers as function of the hypercube dimension showing the optimal (pipe), static general purpose (*crystal_router*) and dynamic general purpose (*neural_router*). The variations in the choice of A, B in (6)–(10) are reflected in the error bars as is the variation with M, the number of messages. Communication time is approximately $\alpha(N)\,M$; $\alpha(N) = \log_2 N$(*crystal_router*) ~ 2.5(*neural_router*) $= 1$ (pipe).

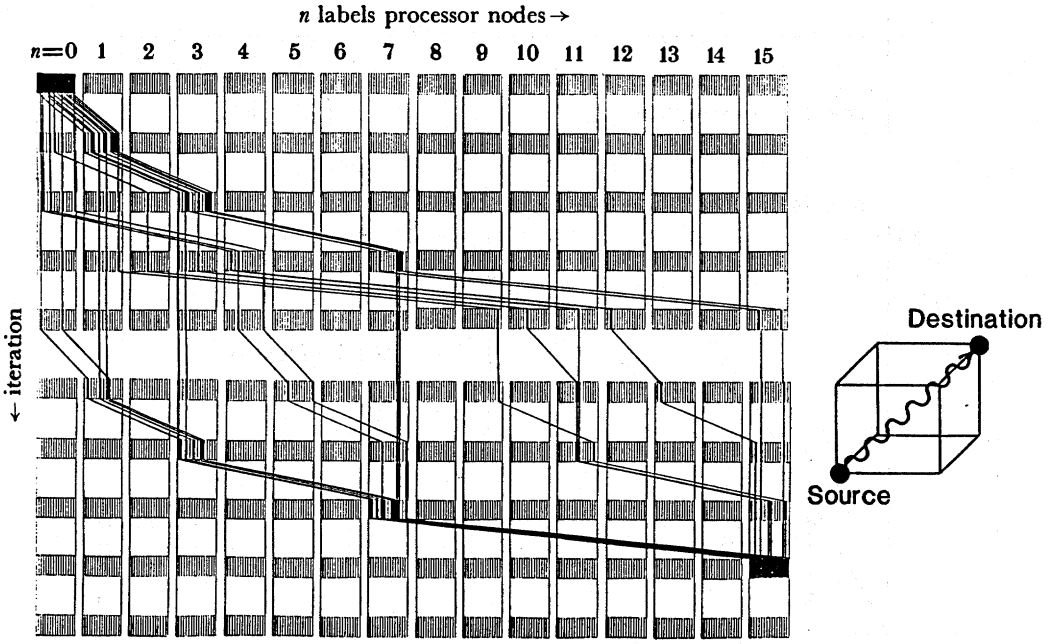


FIGURE 19. The flow of information given by the *neural_router* given for the problem of transmitting $M = 16$ messages from node 0 to node 15 across a longest diagonal of a 16 node (dimension 4) hypercube.

fraction (probability) f of non-zero elements. The resulting performance is plotted against f in figure 21 and compared with the performance of the corresponding analytic algorithms: *fold* and *crystal_router*. An example of the data flow for some particular run (with $f = 0.5$) is presented in figure 20. The network equations (6.11) were applied with the fixed values of the parameters A, B, C.

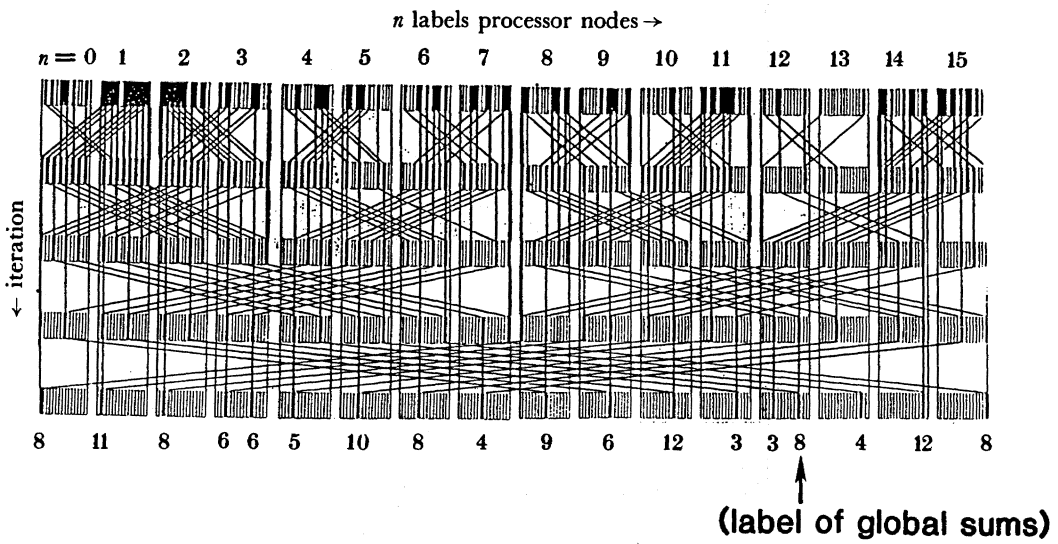


FIGURE 20. The flow of information given by the *neural_accumulator* for a set of 16 global sums on $N = 16$ nodes. Each sum has fN non-zero entries with $f = 0.5$ in this example. The entries are randomly distributed over the nodes.

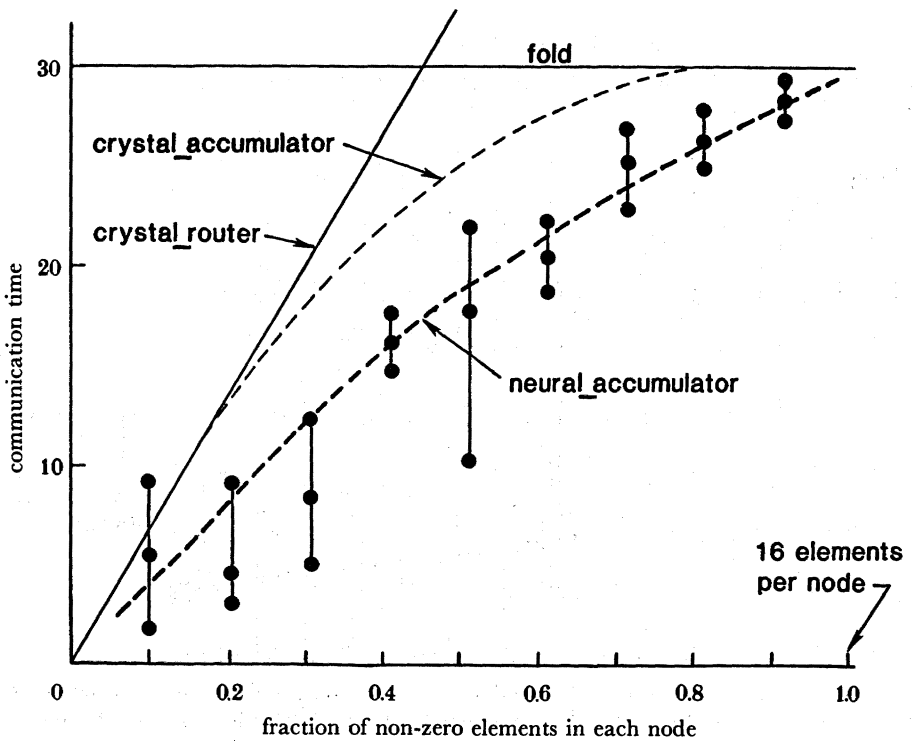


FIGURE 21. The timing of the *neural_accumulator* as a function of f for 16 global sums on a $N = 16$ node hypercube with fN non-zero entries in each sum randomly distributed over the nodes. The error bars reflect an average over different starting configurations as well as variations in basic parameters in (6)–(11) for the *neural_accumulator*.

As seen in figure 20, the network accomplished the requested summation task in the single sweep through all channels up to two mistakes (nodes 3, 12). This type and size of error seems to be typical and difficult to avoid when running the network with fixed values of parameters. The load balancing and the string attraction forces, useful in the first stage of the algorithm, start generating errors in the second, more 'deterministic' stage when the statistical–neural methods become too crude. Clearly, these 'errors' can be cleared up in a fast deterministic final step but we have not explored this.

In the *neural_router* case, we tried different strategy. The algorithm was applied twice: in the first step, the load balancing term was set on, resulting in the desired fan-out of processes, in the second part it was switched off and only the deterministic sink term was retained, resulting in the exact final result.

This simplified 'annealing'-type strategy resulted in quite satisfactory performance. The algorithm automatically reproduces the *crystal_router* strategy in the cases where this is optimal. We present here the numerical results for the inhomogeneous routing case, particularly difficult for the *crystal_router* algorithm: the whole vector (of size M), originally in node $n = 0$ is to be moved to the most distant node ($n = N-1$ on an N-node hypercube). The optimal analytic algorithm (pipe) would take time $T_{\text{comm}} \sim M$ to perform this transfer, the *crystal_router* performance is $T \sim \log_2 NM$, whereas the results for the *neural_router* are shown in figure 19. The numerical values are close to the simple analytic estimate $T_{\text{comm}} \sim 2M$ ($T_{\text{comm}} \sim M$ to fan-out, $T_{\text{comm}} \sim M$ to fan-in). The data flow for the special case $M = N$ on 16-node is presented in figure 18. In the first stage of the algorithm, one can observe the dominant route, driven by the sink term and the fan-out component, enforced by the string repulsive–load balancing term.

The error bars in figures 19 and 21 show the typical spread of the results, obtained by varying the network and/or problem parameters in some 'reasonable' range.

In summary, our preliminary results seem to be promising; we have found that the proposed neural communication algorithms are capable of reproducing and sometimes even improving upon the performance of known analytic algorithms for the discussed tasks. More work is required to analyse the parameter sensitivity, the various 'annealing' modes, and extension of the technique to other, more complex problems.

8. Conclusions

We have tried to show that not only have many problems been successfully run on many parallel machines but also that we are beginning to understand the issues that determine both the performance and optimal decomposition of problems. We have used a complex system framework with a general space-time analogy leading to a seemingly powerful theory of computation based on interacting strings. We have seen that three rather different architectures, the multicomputer, shared memory, and SIMD computer, are all very viable high-performance parallel machines. We have not treated dataflow or systolic arrays and we must leave them to the future to explore these interesting architectures within our framework. Further, the whole and perhaps crucial issue of software has been ignored; this also needs further work.

We expect that the theory of interacting strings proposed in §6 has many applications in parallel supercomputing and other areas. Particularly interesting would be its application to

general networks for transmission of data ranging from those in telephone systems, the structure of management in large companies, and even the network of control satellites proposed for defensive purposes.

This work was supported in part by DOE grant DE-FG03-85ER25009, the Program Manager of the Joint Tactical Fusion Office, and the ESD division of the USAF, as well as grants from IBM and SANDIA.

We thank the many members of the Caltech Concurrent Computation Program whose work was essential input to the analysis of this paper.

References

Aldcroft, T., Cisneros, A., Fox, G. C., Furmanski, W. & Walker, D. 1988 A banded matrix LU decomposition on the hypercube. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C*^{3}*P*-348B.)

Aloisio, G., Fox, G. C., Kim, J. S. & Veneziani, N. 1987 A concurrent implementation of the prime-factor algorithm on hypercube. *IEEE Trans. Acoustics, Speech, Signal Process.* (Submitted.) (*C*^{3}*P*-468.)

Bower, J. M., Nelson, M. E., Wilson, M. A., Fox, G. C. & Furmanski, W. 1988 Piriform (olfactory) cortex model on the hypercube. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C*^{3}*P*-404B.)

Chandy, K. M. & Misra, J. 1987 Conditional knowledge as a basis for distributed simulation. Caltech report 5251:TR:87.

Chen, M., DeBenedictis, E., Fox, G. C., Li, J. & Walker, D. 1988 Hypercubes are general-purpose multiprocessors with high speed-up. (In preparation.) (*C*^{3}*P*-499.)

Dongarra, J. J. 1988 In *Proceedings of ICS87, International Conference on Supercomputing* (ed. C. Polychronoupolos). New York: Springer-Verlag.

Dongarra, J. J., Du Croz, J., Hammaling, S. & Hanson, R. J. 1987 An update notice on the extended Blas. *ACM Signum Newslett.* **21** (4), 1.

Felten, E., Morison, R., Otto, S. W., Barish, K., Fatland, R. & Ho, F. 1986 Chess on the hypercube. In *Proc. 2nd Hypercube Conf., Knoxville* (ed. M. T. Heath). SIAM. (*C*^{3}*P*-383.)

Flower, J. W. 1987 Lattice gauge theory on a parallel computer. Caltech Ph.D. thesis. (*C*^{3}*P*-411.)

Flower, J. W., Otto, S. W. (Caltech) & Salama, M. C. 1988 A preprocessor for finite element problems. In *Proc. Symp. on Parallel Computations and Their Impact on Mechanics, ASME Winter Meeting, 14–16 December 1987, Boston.* (In the press.) (*C*^{3}*P*-292B.)

Fox, G. C. 1983 Scientific calculations with ensemble computers. In *Padua High Energy Physics Microprocessor Conference, 23–25 March 1983.* Preprint (CALT-68-1032. *C*^{3}*P*-37.)

Fox, G. C. 1984 Concurrent processing for scientific calculations. In *Proc. IEEE COMPUCON 1984 Conference, San Francisco, 28 February 1984.* IEEE Computer Society Press. (*C*^{3}*P*-48.)

Fox, G. C. 1985*a* Are concurrent processors general purpose computers? In *IEEE Nuclear Science Symposium, 31 October 1984.* (*IEEE Trans. NPSS* **34**.) (*C*^{3}*P*-122.)

Fox, G. C. 1985*b* The performance of the Caltech hypercube in scientific calculations: a preliminary analysis. *Symp. on Supercomputers-algorithms, architectures and scientific computation, 18–20 March 1985, Austin* (ed. F. A. Masten & T. Tajima). Austin: University of Texas Press. (*C*^{3}*P*-161.)

Fox, G. C. 1986*a* Questions and unexpected answers in concurrent computation. In *Experimental parallel computing architectures* (ed. J. Dongarra). Amsterdam: North Holland. (CALT-68-1403, *C*^{3}*P*-288.)

Fox, G. C. 1986*b* Caltech concurrent computation program annual report for 1985–1986. In *Proc. 2nd Hypercube Conf., Knoxville* (ed. M. T. Heath). SIAM. (CALT-68-1404, *C*^{3}*P*-290B.)

Fox, G. C. 1986*c* A review of automatic load balancing and decomposition methods for the hypercube. In *Proc. Workshop on Numerical Algorithms for Modern Parallel Computer Architectures, IMA, November 1986.* (*IMA Volumes in Mathematics and its Applications*, vol. 13.) (*C*^{3}*P*-385.)

Fox, G. C. 1987 The hypercube as a supercomputer. In *2nd Int. Conf. on Supercomputing, Santa Clara, May 1987.* St Petersburg, Florida: International Supercomputing Institute, Inc. (*C*^{3}*P*-391.)

Fox, G. C. (ed.) 1988*a* Hypercubes, concurrent computers and applications, 1988. *Proc. 3rd Hypercube Conference, Pasadena, 19–20 January 1988.* (In the press.)

Fox, G. C. 1988*b* The hypercube and the Caltech concurrent computation program, a microcosm of parallel computing. In *Special purpose computers* (in *Methods in computational physics*) (ed. B. Alder.) (*C*^{3}*P*-422.)

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Fox, G. C. 1988*c* Domain decomposition in distributed and shared memory environment. I A. Uniform decomposition and performance analysis for the NCUBE and JPL Mark IIIfp hypercubes. In *ICS 87, International Conference on Supercomputing, Athens, 8–12 June 1987* (ed. C. Polychronopoulos) (*Lecture Notes in Computer Science*). Berlin: Springer-Verlag. (*C³P*-392.)

Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K. & Walker, D. 1988 *Solving problems on concurrent processors*. Prentice Hall: New Jersey.

Fox, G. C. & Otto, S. W. 1984 Algorithms for concurrent processors. *Physics Today*: (May issue). (*C³P*-71.)

Fox, G. C. & Otto, S. W. 1986 Concurrent computation and the theory of complex systems. In *Proc. 1st Hybercube Conf., Knoxville* (ed. M. T. Heath). SIAM. (*C³P*-255.)

Fox, G. C., Lyzenga, G., Otto, S. W. & Rogstad, D. 1986*a* The Caltech Concurrent Computation Program – project description. In *Proc. Conf. on The 1985 ASME International Computers in Engineering, 4–8 August 1985*, Boston. ASME; Berlin: Springer-Verlag. (*C³P*-157.)

Fox, G. C., Otto, S. W. & Umland, E. A. 1986*b* Monte Carlo physics on a concurrent processor. In *Conf. on Frontiers of Quantum Monte Carlo, Los Alamos, 6 September 1985* (*J. statist. Phys.*), vol. **43**, pp. 1209. (*C³P*-214.)

Frey, A. & Fox, G. C. 1988 Features of a teraflop supercomputer. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C³P*-606.)

Furmanski, W. & Fox, G. C. 1988*a* Load balancing by a neural network. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (CALT-68-1408, *C³P*-363B.)

Furmanski, W. & Fox, G. C. 1988*b* Optimal communication algorithms on the hypercube. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C³P*-314B.)

Furmanski, W. & Fox, G. C. 1987*a* Communication algorithms for regular convolutions on the hypercube. In *Proc. 2nd Hypercube Conf., Knoxville* (ed. M. T. Heath). SIAM. (*C³P*-329.)

Furmanski, W. & Fox, G. C. 1987*b* Hypercube communication for neural network algorithms. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C³P*-495B.) (*C³P*-405B.)

Furmanski, W. & Fox, G. C. 1988*c* Optimal matrix algorithms on homogenous hypercubes. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C³P*-386B.)

Gottlieb, A. 1987 An overview of the NYU Ultracomputer Project. In *Experimental parallel computing architectures* (ed. J. J. Dongarra). Amsterdam: North-Holland.

Gottschalk, T. D. 1987 Multiple-target track initiation on a hypercube. In *2nd Int. Conf. on Supercomputing, Santa Clara, May 1987*. St Petersburg, Florida: International Supercomputing Institute, Inc. (*C³P*-398.)

Heath, M. T. (ed.) 1986 Hypercube multiprocessors, 1986. *Proc. 1st Hypercube Conf., Knoxville*. SIAM.

Heath, M. T. (ed.) 1987 Hypercube multiprocessors, 1987. *Proc. 2nd Hypercube Conf., Knoxville*. SIAM.

Hey, A., Nicole, D. (Southampton), Fox, G. C. & Otto, S. W. (Caltech) 1988 Communication Overheads for MIMD Computers. G. (In preparation.) (*C³P*-221.)

Hillis, W. D. 1985 *The connection machine*. Cambridge, Massachusetts: MIT Press.

Hillis, W. D. 1987 The connection machine. *Scient. Am.* **256**, 108–115.

Hipes, P. & Kuppermann, A. 1988 Gauss Jordan Matrix-inversion with pivoting on the hypercube. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C³P*-495B.) (*C³P*-578.)

Ho, C.-T. & Johnsson, S. L. 1986*a* Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *Proc. IEEE 1986 Int. Conf. on Parallel Processing*.

Ho, C.-T. & Johnsson, S. L. 1986*b* Matrix transposition on Boolean n-cube configured ensemble architectures. Yale report YALEU/DCS/TR-494.

Hopfield, J. J. & Tank, D. W. 1986 Computing with neural circuits, a model. *Science, Wash.* **233**, 625.

Jefferson, D. & Sowizral, H. 1985 Fast concurrent simulation using the time-warp mechanism. In *SCG Conference on Distributed Simulation, San Diego, California*.

Johnsson, L. & Ho, C.-T. 1988 Matrix multiplication on Boolean cubes using generic communication primitives. In *Proc. ARO Workshop on Parallel Processing and Medium-Scale Multiprocessors*. (In the press.)

Kolawa, A., Fox, G. C. & Williams, R. 1987 The implementation of a dynamic load balancer. In *Proc. 2nd Hypercube Conf., Knoxville* (ed. M. T. Heath). SIAM. (*C³P*-328.)

Koller, J. 1988 A dynamical load balancer in the INTEL hypercube. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C³P*-497.)

Mandelbrot, B. 1979 *Fractals: form, chance, and dimension*. San Francisco: Freeman.

Meindl, J. D. 1987 Chips for advanced computing. *Scient. Am.* **257** (4), 72–88.

Messina, P. C. & Fox, G. C. 1987 Advanced computer-architectures. *Scient. Am.* **257** (4), 66–77. (*C³P*-476.)

Salmon, J., Flower, J., Kolawa, A. & Collaha, S. 1988 MOOSE: a multitasking operating system for hypercubes. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C³P*-586.)

Stout, Q. F. & Wager, B. 1987*a* Intensive hypercube communication. I. Prearranged communication in link bound machines. CRL-TR-9-87 Michigan report.

Stout, Q. F. & Wager, B. 1987*b* Passing messages in link-bound hypercubes. In *Proc. 2nd Hypercube Conf., Knoxville* (ed. M. T. Heath). SIAM.

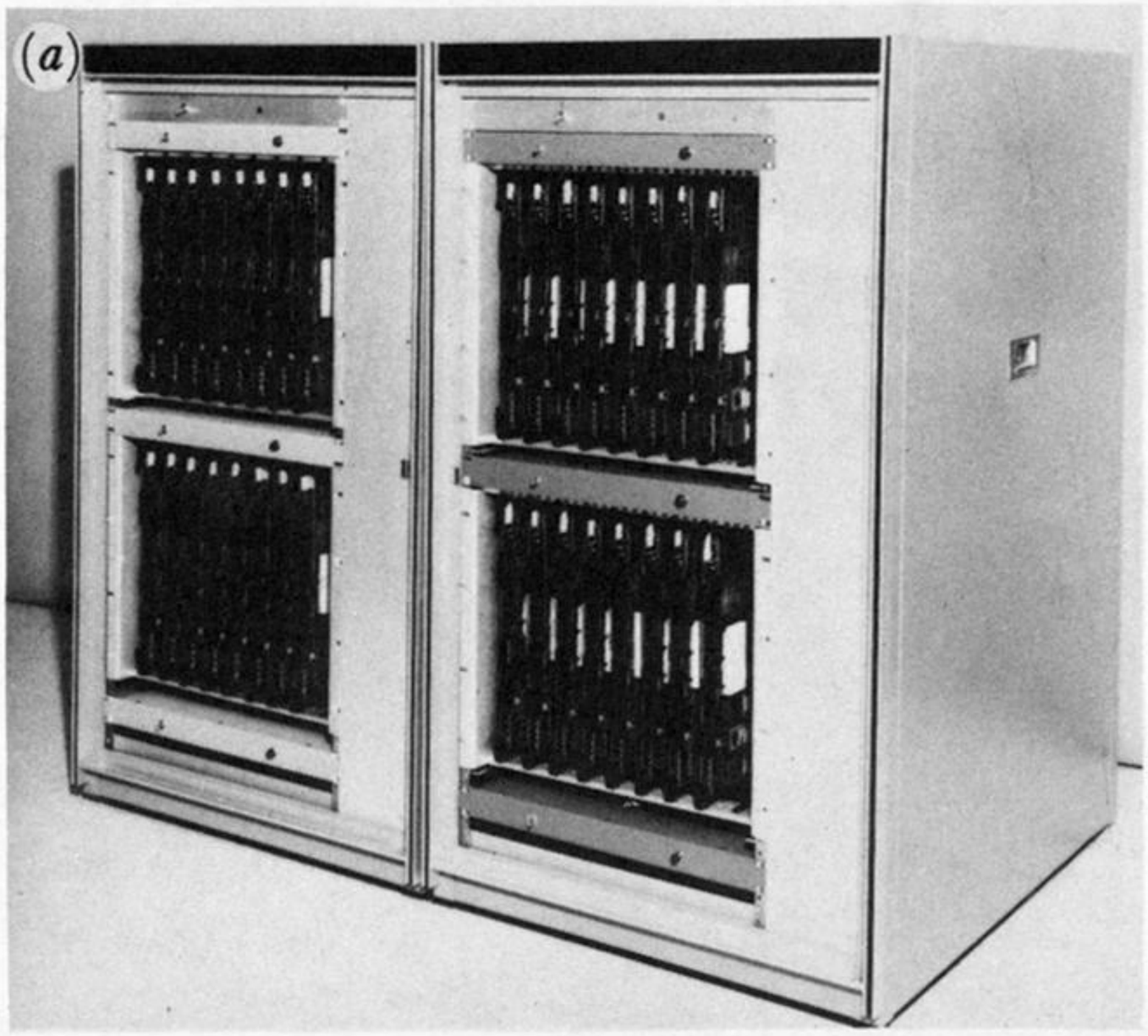
Walker, D. 1988 Performance of the Caltech QCD Code on the NCUBE. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.)

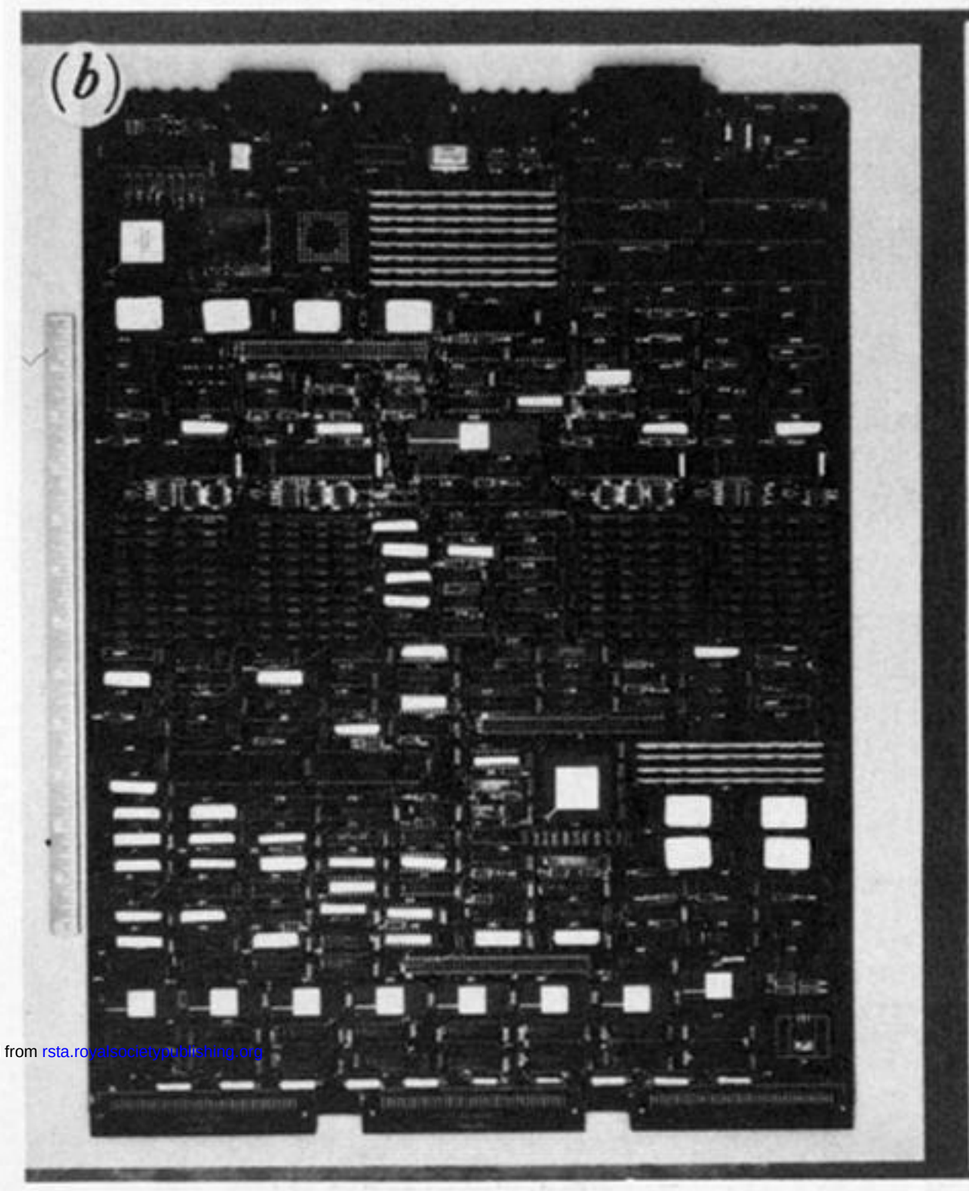
PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

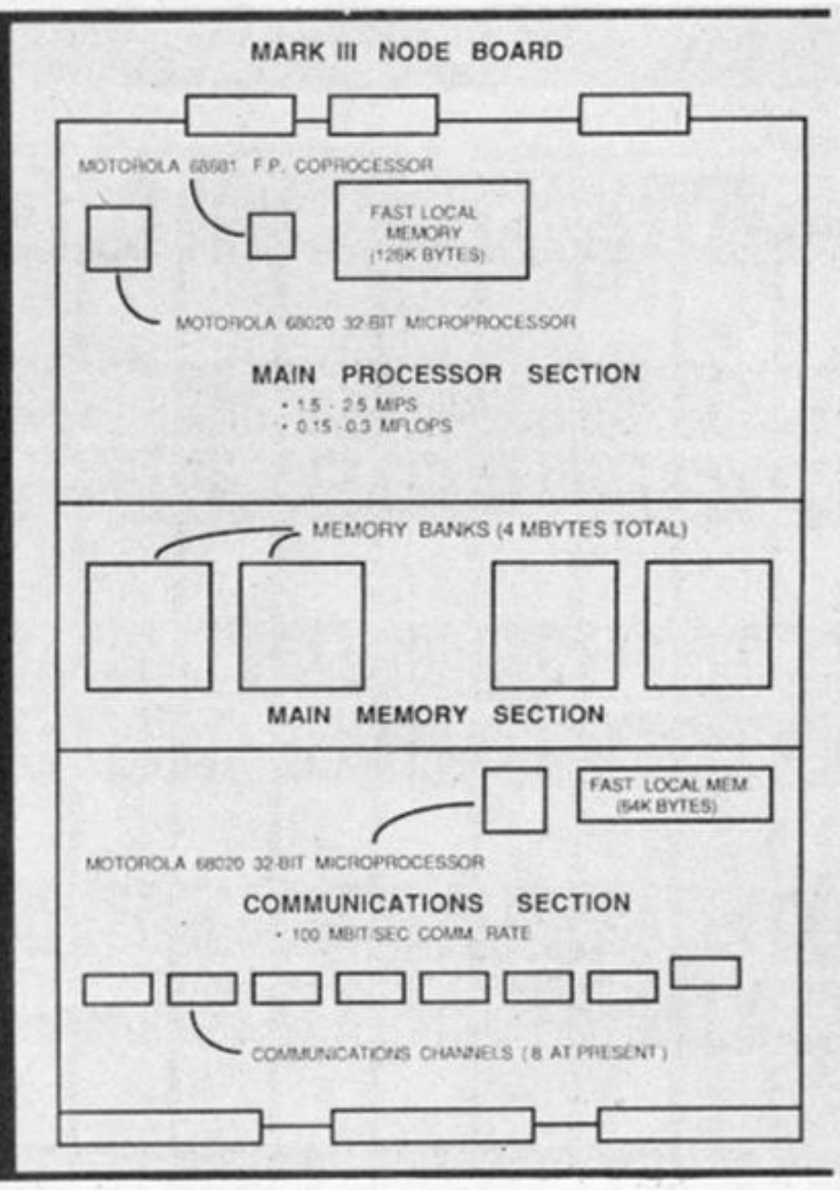
Walker, D. & Fox, G. C. 1988 A portable programming environment for concurrent multiprocessors. (In preparation.) (To be presented at the 12th IMACS World Congress on Scientific Computation, Paris, July 1988.) (*C³P*-496.)

Walker, D. & Montry, G. (Sandia) 1988 Implementation and performance of a two dimensional flux corrected transport code on the NCUBE. In *Proc. 3rd Hypercube Conf., Pasadena, 19–20 January 1988* (ed. G. C. Fox). (In the press.) (*C³P*-495B.)

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES







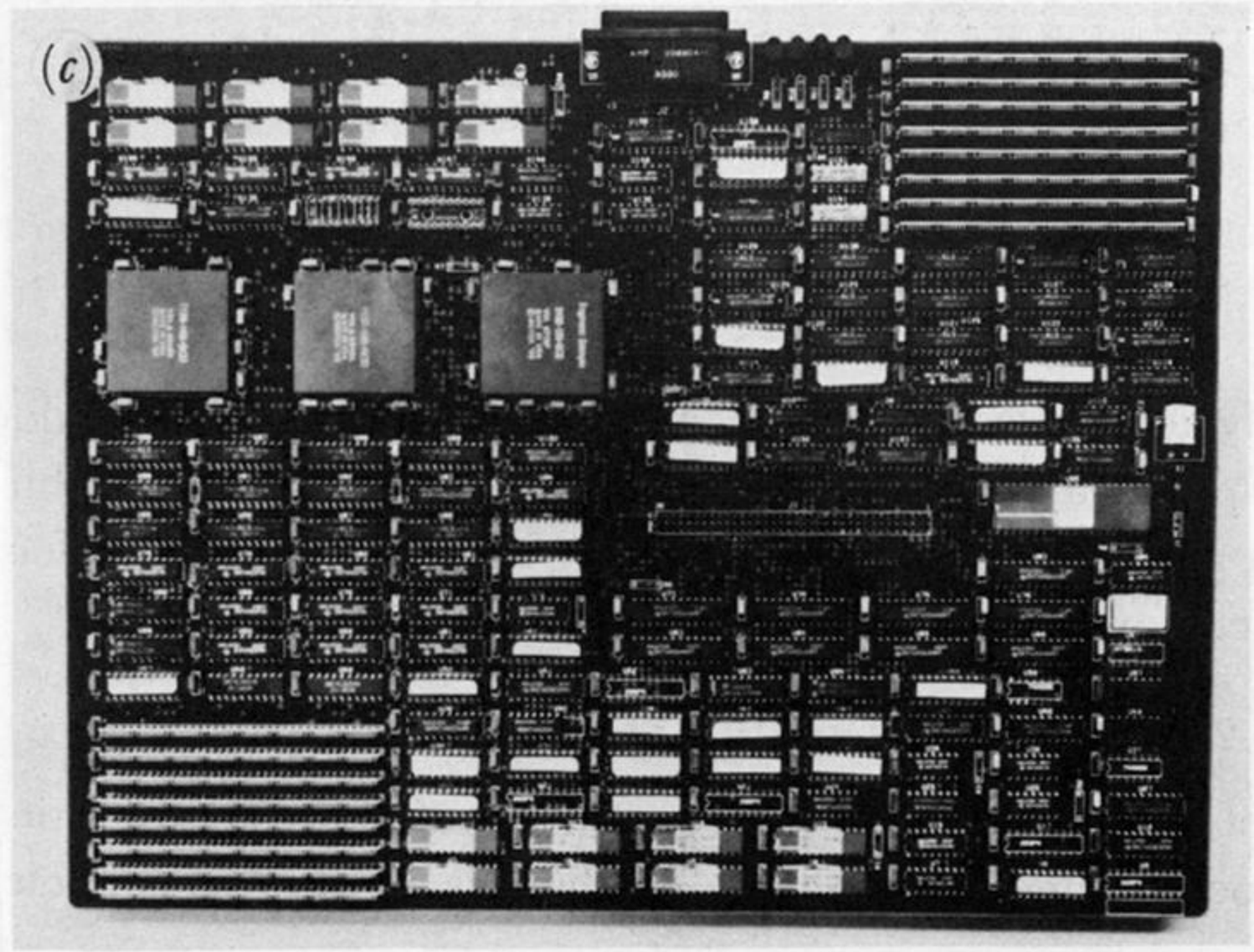
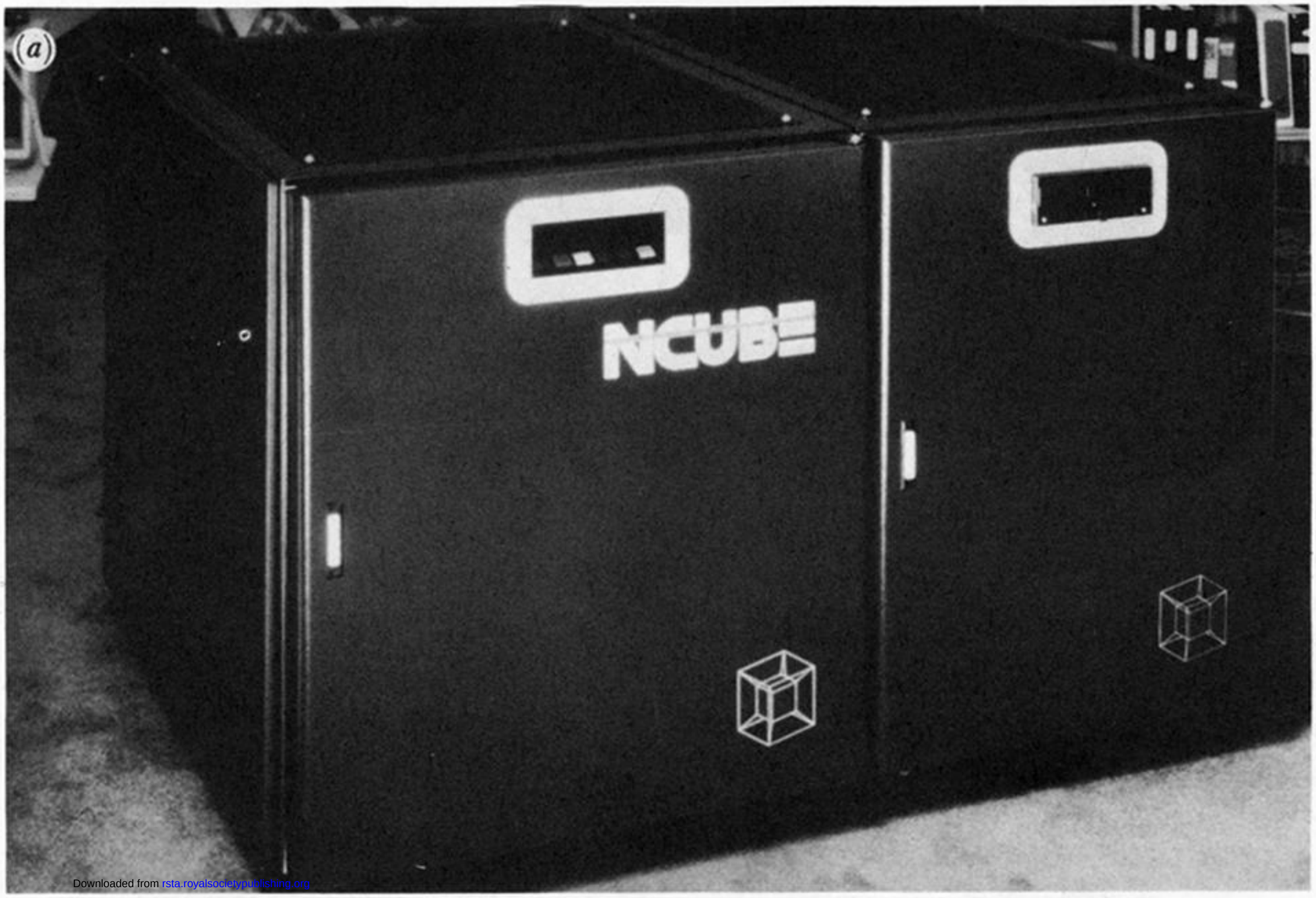


FIGURE 1. The Mark IIIfp hypercube designed and constructed at Caltech's Jet Propulsion Laboratory (JPL). (*a*) The basic 32 node package which can be extended up to 128 modes. (*b*) The dual 68020 based main node board. (*c*) The WEITEK XL chip set based secondary board.



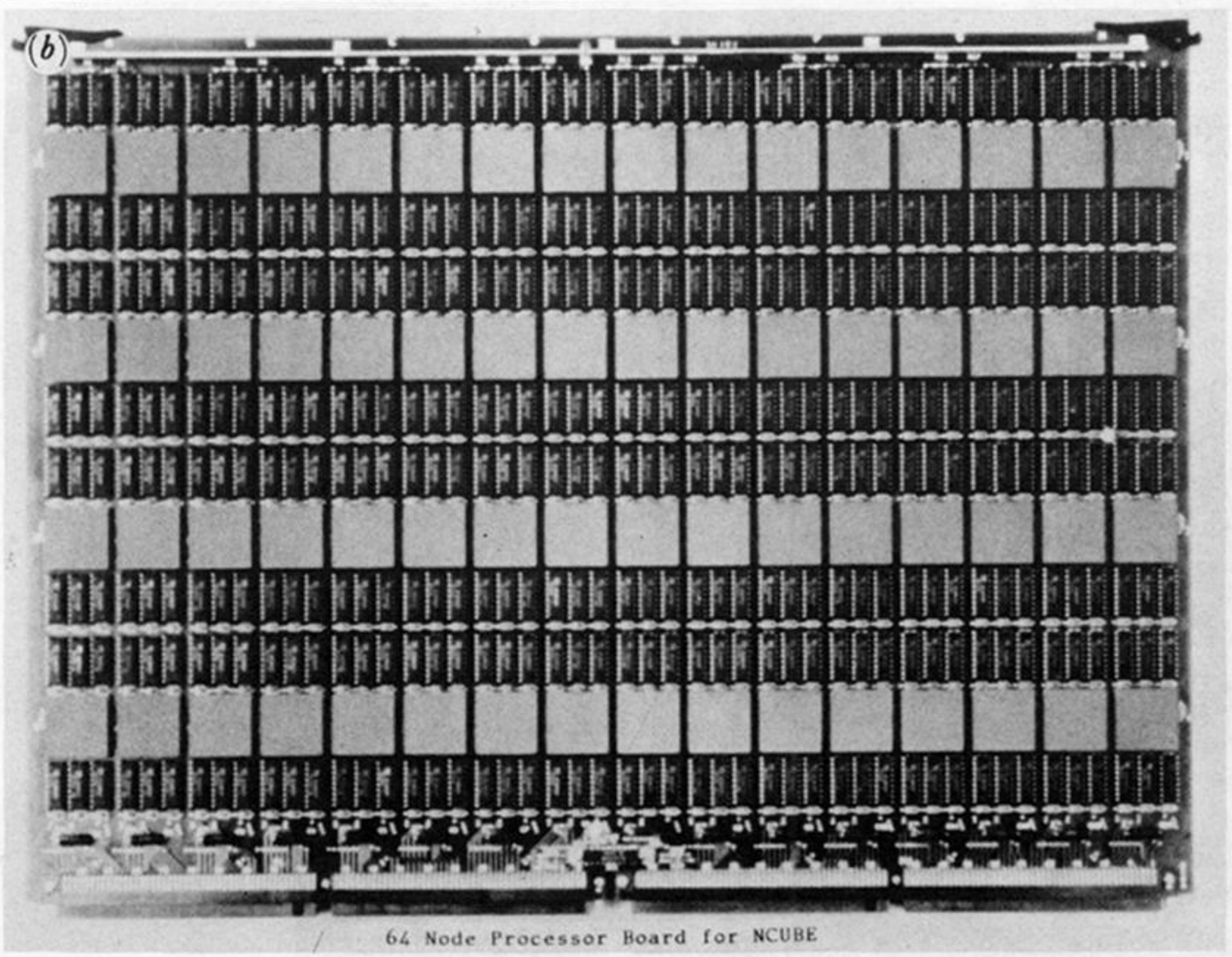


FIGURE 2. The commercial NCUBE/10 hypercube with (*a*) cabinet which can hold up to 1024 nodes configured as a 10 dimensional hypercube; (*b*) basic board containing 64 nodes.