THE GF-11 SUPERCOMPUTER

John Beetem
Monty Denneau
Don Weingarten

Reprinted from the proceedings of the 12th Annual International Symposium on Computer Architecture, held Boston, Massachusetts, June 17-19, 1985

The GF11 Supercomputer

John Beetem, Monty Denneau, and Don Weingarten

IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

GF11 is a parallel computer currently under construction at the Yorktown Research Center. The machine incorporates 576 floating-point processors arranged in a modified SIMD architecture. Each processor has space for 2 Mbytes of memory and is capable of 20 MFLOPS, giving the total machine a peak of 1.125 Gbytes of memory and 11.52 GFLOPS. The floating-point processors are interconnected by a dynamically reconfigurable non-blocking switching network. At each machine cycle any of 1024 pre-selected permutations of data can be realized among the processors. The main intended application of GF11 is a class of calculations arising from quantum chromodynamics, a proposed theory of the elementary particles which participate in nuclear interactions.

Introduction

GF11 is a parallel computer conceived primarily for the numerical solution of problems in quantum chromodynamics (QCD), a proposed theory of the class of particles which participate in nuclear interactions. A typical calculation in QCD, for example an evaluation of the masses of the proton, neutron and a few related particles, is estimated to require 3×10^{17} arithmetic operations [1]. With a 100 MFLOP machine (such as the Cray I) this calculation would take 100 years. By a parallel application of its 576 processors, GF11 is capable of 11.5 GFLOPS peak and about 10 GFLOPS sustained performance for QCD. The mass calculation can be completed in about 1 year.

Although QCD is the main target of GF11, the machine's architecture is sufficiently flexible that arithmetic in the range of 3 to 10 GFLOPS can probably be delivered for a fairly wide range of problems in science and engineering.

Several innovations in hardware technology were essential to realizing this performance. The first is the existence of high-speed single-chip floating point processors, *viz.* the Weitek WTL1032 floating point multiplier and WTL1033 floating point ALU. Each of these chips can perform 5 million 32-bit computations/second in pipelined mode; this performance previously required a whole board of logic. However, the raw processing power of these chips is wasted unless operands and results can be transmitted fast enough. This is done in GF11 using a very high-speed register file implemented in ECL. By storing intermediate results and smoothing pipeline delays, the register file effects 92% utilization of the processors for typical QCD computations.

The other major hardware innovation of the GF11 is the processor interconnection. Rather than a fixed interconnect, such as a two-dimensional mesh, GF11 connects the processors through a full Benes network [2], a non-blocking switch capable of realizing any permutation of the processors and instantaneous reconfiguration. Using the switch, GF11 can be organized in many different topologies, such as a rectangular mesh of any dimensionality and size, any torus, a hexagonal mesh, or some irregular organization fitted to a single problem. In addition, the full permutation allows spare processors to be swapped in to replace failed processors.

For discussion of a related but distinct architecture for a parallel computer for QCD calculations see ref. [3].

Architecture

The GF11 architecture is shown in Figure 1. There are 576 20-MFLOP processors, of which some number are spare processors to be enabled if a primary processor fails. A typical partition is 512 primary processors and 64 spares. The processors are interconnected through a three stage Benes network, called the **Memphis switch**. Data is sent bytewise through the Memphis switch. A central controller broadcasts instructions to all of the processors and the switch, and also communicates with a 3084 host computer.

GF11 is a modified Single-Instruction Multiple-Data (SIMD) machine: all processors receive the same instruction at the same time. A SIMD architecture has a variety of advantages:

1. The machine is simpler to design, understand, program, and debug.
2. SIMD improves performance by eliminating synchronization overhead.
3. There is one common instruction memory for the whole machine. This store can be made much larger than would be possible if each processor had its own instruction memory.

0149-7111/85/0000/0108$01.00 © 1985 IEEE

4. It is very difficult to build a general-purpose processor which can keep up with a 20 MFLOP non-vector floating point unit. There are no microprocessors (and few mainframes) fast enough. In GF11, only one such processor is needed (the central controller) -- for a MIMD equivalent 576 processors would be needed.

Of course not all problems can efficiently utilize a set of processors receiving a single instruction stream. Fortunately a large class of scientific problems, and QCD in particular, do not encounter this difficulty. We will return to this topic later.

The GF11 Processor

One of the 576 processors comprising GF11 is shown in Figure 2. The heart of the processor is a 20 MFLOP 32-bit floating point unit and a 20 MIP 32-bit fixed point unit. Neither arithmetic unit requires data to be organized in vectors for full performance. The floating point unit contains four Weitek 32-bit IEEE floating point chips: two multipliers and two ALU's.[1] Each Weitek chip is capable of 5 MFLOPs in pipelined mode. The fixed point unit, implemented in TTL, includes a 32-bit barrel shifter and a general-purpose ALU supporting addition, subtraction, and all logic operations. GF11 can run fixed point code at 11.5 GIPS, or can use the fixed point unit in floating point applications for field extraction from floating point numbers and address calculations.

Each processor has a considerable amount of local data memory: 64 Kbytes of high-speed static RAM, to be used to store frequently accessed data, and 256 Kbytes (expandable to 2 Mbytes) of dynamic RAM, used to store long term data. The entire 576 processor GF11 thus has 36 Mbytes of static RAM and 144 Mbytes to 1.125 Gbytes of dynamic RAM. Both RAMs are fully random access -- vectored data is not required.

The static RAM alone does not have enough memory bandwidth to keep the arithmetic units busy continuously. This limitation is softened using a 256 word 12.5ns **register file**, implemented in ECL. On each 50 ns microcycle, the register file is used four times: Two register file operands are sent to the fixed or floating point unit, a fixed or floating point result is stored, and a word is transferred between static RAM and the register file. The register file reduces memory bandwidth by retaining operands which are used multiple times as well as acting as a "scratch pad" for intermediate results. It also smooths out the pipeline delays associated with the arithmetic units thereby simplifying code generation for the GF11.

The processor has two external data ports. One of these is a transceiver port for communicating with other processors through the Memphis switch. The transmitter takes a 32-bit word from the static RAM and sends it to the Memphis switch. The receiver accepts a word from the Memphis switch and stores it in the register file. Thus a processor can fetch operands from its own or a neighbor's SRAM. The other port is a 32-bit data path for communicating global data to and from the central controller over backplanes. Data from the processors is conditionally ORed to combine data for the whole array.

Finally, the processor has some special functions which circumvent some of the limitations of SIMD. These are:

1. A set of 256 base registers to relocate static RAM addresses. Each processor can have a different base value so that each processor can be processing a different variable or array.
2. A set of 8 condition code bits which are set as a result of fixed or floating point computations. These condition codes can selectively control processors based on the result of previous computations. For example, an operation **A op B** can be inhibited to instead produce the result **A**. Selection between local and neighbor operands is also conditional, to allow irregular boundary conditions. All data stores are conditional.

A GF11 processor is controlled by a 180-bit horizontal microcode word which controls each subfunction independently. This maximizes the flexibility and utilization of the processor and simplifies code generation. Since the same control word is broadcast to all 576 processors, the cost of a wide microcode word is quite small on a per processor basis.

The Memphis Switch

The GF11 processors are interconnected through a high-speed, pipelined, non-blocking three stage Beneš network, shown in Figure 3. Each stage of the Memphis switch consists of 24 24-input crossbar switches. The middle stage is connected to the outer stages by "perfect shuffle" fixed interconnections. By suitable configurations of the crossbars it is possible to realize any permutation of the 576 inputs. In addition, it is possible to realize mappings more general than permutations: for example, any input can be broadcast to all outputs.

All data paths in Figure 3 are nine bits wide: 8 data bits plus a parity bit. The data rate on each path is 20 Mbytes/sec; for 576 processors the aggregate data rate is 11.5 Gbytes/sec. The 24-24 crossbar switching is accomplished using a high-speed semicustom CMOS gate array chip designed by the authors and manufactured by LSI

[1] The ALU can add, subtract, take absolute values, and convert between fixed and floating point. There is no built-in division. However, quotients can be computed efficiently by making an initial estimate and improving it using two adds and three multiplies.

Logic Corporation. 18 of these chips are used per crossbar; 1296 are needed to implement the entire Memphis switch.

For most expected GF11 applications, only a few switch configurations are required. For example, if the application involves a two dimensional mesh where only nearest neighbors are important, just four configurations are needed: send north, send south, send east, and send west. A four dimensional torus requires only eight. The Memphis switch allows up to 1024 distinct configurations, enough to cover a wide variety of applications and allow easy communication with non-adjacent neighbors.

Every 200 ns the Memphis switch receives a new word of data from each board and is assigned a new configuration from among the 1024 preloaded possibilities. It takes considerably longer, however, to compute and load a completely new configuration. For most applications, it is expected that the configurations will be loaded at the beginning of the run and will remain constant throughout the whole execution of the problem.

The GF11 Controller

Since GF11 is a quasi-SIMD machine, a single central control is adequate for controlling all aspects of machine execution. The controller (Figure 4) has several functions:

1. Storage and broadcast of GF11 instruction streams.
2. Address relocation and remap.
3. Communication with the host CPU.
4. Status and error checking.

The foremost requirement for the GF11 controller is speed -- the GF11 microcycle is 50 ns and the controller must be able to broadcast instructions, addresses, and data at that rate. This eliminates the possibility of using a microprocessor or even a group of several microprocessors as the central controller. It is conceivable to build a specialized controller out of high-speed bit-slice elements, but this would reduce flexibility and possibly reduce the range of future applications beyond QCD.

The controller design which was finally adopted uses a very large memory to store horizontal microcode for the processors and switch. This microcode is computed by the host and loaded into the controller prior to problem execution. The Control RAM can broadcast instructions, addresses and data to the GF11 array at a rate of one 256-bit word every 50ns. In addition, it can receive result data from the GF11 array at the same rate.

The depth of the Control RAM is 512 Kwords, expandable to 4 Mwords. GF11 uses microcode quite rapidly, however, and as a result the total microcode store represents only 25 to 200 milliseconds of execution. Since microcode generation is too time consuming to be done on the fly, sustained operation is possible only by reusing at least some microcode sequences a large number of times. This turns out possible in the QCD applications we have considered. The time consuming sections of these algorithms all consist of a large number of iterations of an inner loop which uses the same instruction sequence on each pass.

GF11 treats microcode sequences as *subroutines* to be *called* by the host. Although the instruction sequence of a subroutine is fixed, it is necessary that different invocations of a subroutine can pass different variables and arrays as arguments. GF11 has two mechanisms to accomplish this:

1. Static RAM addresses can be relocated within the controller by one of 1024 relocation registers. These relocation registers are loaded from the host between microcode subroutine invocations.
2. There is a very general "re-map" mechanism for mapping any relocated static RAM address into any other by table lookup. This is used to implement gather/scatter, periodic arrays, or virtually any unusual mapping required by a future application.

These mechanisms, along with the per-processor relocation mentioned earlier, provide powerful address manipulation and allow massive re-use of microcode subroutines.

While microcode subroutines are very fast, they do not have any decision-making capability. In particular, they have no conditional branching and do not allow further levels of subroutine calls. The intelligence of the GF11 lies in the **Control CPU**, which lies between the microcode generators and the host. The Control CPU decides which sequence of microcode is to be executed next, sets up the necessary relocation and remap values, and starts the next sequence when the previous one finishes. Deciding what to do next can be based on results from previous subroutines, or status bits. In addition, the Control CPU contains hard disk storage, to act as a buffer between the host and the GF11 processor array and to store machine checkpoints for error recovery during very long computations.

The machine currently acting as the Control CPU is an IBM PC/AT. Provided that the microcode subroutines are long enough (i.e. several thousand instructions), the IBM PC/AT has enough time while a microcode subroutine is executing to set up the next one.

Software Development

The GF11 software strategy is to provide adequate program development tools in as simple a way as possible. GF11 is essentially impossible to program by hand at the microcode level -- the processor function is too rich[2] and the pipelines are too complex -- thus some sort of high level language is needed. However, we did not want to create a

[2] For example, there are 59 microcode fields.

new language and compiler at this time -- we want to get more experience with the hardware and initial applications. The approach we have taken is to use Pascal in an unusual way for software development.

There are two parts to GF11 software: the microcode subroutines and the master program which calls them. The master program, which runs on the IBM PC/AT, is a conventional Pascal program which can take advantage of all features of the language. To call microcode subroutines, the master program invokes several predefined procedures to (1) set up relocations values, (2) invoke a microcode subroutine, and (3) transfer instructions and data between the IBM PC/AT and the Control RAM.

The microcode subroutines are also written in Pascal, but in a special way. The purpose of running the Pascal microcode subroutine is not to generate a numeric result, but rather to obtain the sequence of GF11 operations needed to generate the result. The Pascal microcode subroutine does not use the conventional Pascal operations like + and *. Instead, the Pascal subroutine calls predefined code-generation procedures such as **ADD** and **MULT**. Then, when the Pascal subroutine is compiled and executed, instead of numeric results, the code generation procedures generate the graph of operations to be executed by GF11.

For example, the microcode Pascal statement:

```
COMPUTE(A,MULT(B,C))
```

does not produce the matrix product of B and C, but rather the GF11 code which computes B*C and stores the result in A. It is important to note that all the address calculations are performed when the Pascal microcode is processed by the host. GF11 does not incur the significant overhead of array address calculations.

After the initial GF11 code has been produced in the form of an operation graph, an optimizer compacts it as tightly as possible into GF11 horizontal microcode. When appropriate, the operations are reordered to ensure a tight fit; 92% utilization is typical for QCD algorithms. This microcode can then be combined with other microcode subroutines and loaded into the Control RAM.

Implementation

GF11 is implemented using conventional vendor technology. Except for the gate array used in the Memphis switch, all ICs are off-the-shelf vendor components, mostly Fairchild FAST TTL logic. Fairchild 100K ECL is used for very high-speed sections such as the register file, and for driving cables (all of which are differential pair). The total chip count for the GF11 is approximately 400,000.

GF11 is housed in standard 19" and 24" equipment racks. The 576 processors occupy 20 racks, including air cooling and power supplies. The Memphis switch occupies 5 racks; two of these are used only for the cables connecting successive stages. The central controller fits in two racks. The total power consumption is approximately 200,000 watts.

Application

The main intended application of GF11, as we have already mentioned, is the numerical evaluation of some of the predictions of quantum chromodynamics. We will now give a brief overview of QCD, the mathematical tasks involved in obtaining predictions, and how these calculations can be done on GF11.

QCD is a theory of the particles which participate in nuclear interactions. Among these are the proton, neutron, delta baryon, pion and rho. According to QCD, these particles are composed of still more elementary objects called quarks and antiquarks bound together by the action of something called the chromoelectric field. In a similar way, the electrons and nucleus of an atom are bound together by the electromagnetic field. QCD provides a formula for the probability that any specified configuration of quarks and field at one instant of time will arrive, at some later instant, at another specified configuration. Given such transition probabilities between any initial configuration and any final configuration, relatively simple formulas can then be used to extract a variety of testable predictions. The masses of the proton, neutron and delta baryon can be obtained, for example, from transition probabilities for three quark systems.

A mathematically well-defined expression for transition probabilities is provided by a formulation of QCD [4] in which space and time are approximated by a four-dimensional hypercubic lattice of points. Predictions for the real world are obtained by taking the limit of the lattice theory's predictions as the lattice spacing goes to zero and lattice volume goes to infinity. There is some evidence that a reasonable approximation to the real world can be obtained on lattices as small as $6\times6\times6\times6$. Quite accurate predictions are expected from a lattice $16\times16\times16\times16$. In the lattice theory the chromoelectric field is represented by a 3×3 complex matrix $\mathbf{U}(x,y)$ assigned to each link (x,y) of the lattice joining a pair of nearest neighbor sites. In addition, each site of the lattice carries a 12 component complex vector $\phi(x)$ which, in a sense, may be thought of as a quark field. Using the lattice formulation, any transition probability $\mathbf{T}_{AB}$ between configurations A and B can be expressed as an integral, over all $\mathbf{U}(x,y)$ and $\phi(x)$, of a function $F_{AB}(\mathbf{U}, \phi)$ which depends on the transition considered multiplied by a second universal function $\exp[S(\mathbf{U}, \phi)]$:

$$\mathbf{T}_{AB} = \int d\mu(\mathbf{U}, \phi)\, F_{AB}(\mathbf{U}, \phi) \exp[S(\mathbf{U}, \phi)] \quad (1)$$

If such integrals can be evaluated with sufficient accuracy, any testable prediction of QCD can be extracted. For a $6\times6\times6\times6$ lattice, the integral is over 72,576 real dimensions. For a $16\times16\times16\times16$ lattice, the integral is over 3,670,016 dimensions. The simplest methods of numerical integration, such as use of the trapezoidal rule, would require astronomical amounts of time to evaluate eq. (1) even using GF11.

A Monte Carlo method which is capable of evaluating eq. (1) on GF11 was suggested in ref. [5] To do the integral of eq. (1) following ref. [5], a random sequence of lattice configurations of the U and ϕ variables is generated with differential probability

$$d\mu(\mathbf{U}, \phi) \exp[S(\mathbf{U}, \phi)] \qquad (2)$$

This is a well defined probability distribution since, it turns out, $S(\mathbf{U}, \phi)$ is real while $d\mu(\mathbf{U}, \phi)$ is defined in such a way that the total integral of the quantity in eq. (2) is necessarily one. The function $F_{AB}(\mathbf{U}, \phi)$ is then found on each random configuration, and this set of values is averaged. The result is $\mathbf{T}_{AB}$.

To execute the algorithm of ref. [5] for an $8\times8\times8\times8$ lattice, GF11 is configured as an $8\times8\times8$ array of 512 processors with nearest neighbor connections through the switch. This uses only 6 of the 1024 available switch settings. Each processor manages all the data for a line of sites with a single value of lattice coordinates x_1 , x_2, and x_3, and values of x_4 running from 1 to 8. To generate the sequence of random U and ϕ, some starting configuration is chosen and then successively modified site by site. The modification at a single lattice site x consists of making a random trial change in either $\mathbf{U}(x,y)$ or $\phi(x)$ at that site, then keeping or rejecting this change with a probability determined by the shift $\Delta S(\mathbf{U}, \phi)$ in the function $S(\mathbf{U}, \phi)$ to which the trial field update gives rise. The trial modification of fields at a site and final determination whether to keep or reject the change, once $\Delta S(\mathbf{U}, \phi)$ has been found, require only a tiny fraction of the total operation count for the algorithm. These operations are carried out one site at a time by disabling stores back into memory in all but one processor. The difficult step is evaluating $\Delta S(\mathbf{U}, \phi)$. This is done in parallel on all 512 processors.

Essentially all the work in the evaluation of $\Delta S(\mathbf{U}, \phi)$ is spent in the solution of a linear equation

$$\psi_a(x) = \sum_{by} \mathbf{M}^{\mathbf{U}}_{ab}(x,y)\, \psi_b(y) + \chi^{\phi}_a(x) \qquad (3)$$

for an auxiliary 12 component complex vector $\psi_a(x)$ on each lattice site x. The matrix $\mathbf{M}^{\mathbf{U}}_{ab}(x,y)$ is nonvanishing only for x and y which differ by a single lattice spacing and is a simple function of $\mathbf{U}(x,y)$. The source vector $\chi_a(x)$ for a site x, on the other hand, is a simple function of the quark field $\phi_a(x)$ at the same site. Eq. (3) is solved iteratively by choosing some convenient initial $\psi_a(x)$ at each x, and then successively calculating better approximations to $\psi_a(x)$ by evaluating the right side of eq. (3) using the best preceding approximation. Each of the 512 active processors operate in parallel to evaluate this expression for $\psi_a(x)$ for the 8 values of x it has been assigned. The arithmetic operations needed at each site are identical so the single instruction stream sent out by the central controller is adequate. The evaluation of this expression for the 12 components $\psi_a(x)$ at a single x takes about 2500 operations. This calculation requires about 200 words of data through the switch from nearest neighbor processors. The resulting rate of about $1/12$ word through the switch per arithmetic operation is a factor of 3 lower than the peak rate of $1/4$ word per operation which the switch is capable of delivering.

The iterative solution of eq. (3) converges fastest if the sites which are updated simultaneously in different processors are not coupled by the matrix $\mathbf{M}^{\mathbf{U}}_{ab}(x,y)$ and therefore are not nearest neighbors in the space-time lattice. This presents a problem for the SIMD architecture we have chosen if the 8 site line carried by each processor is put into memory in the same way. Since each processor receives the same data addresses from the central controller, nearest neighbor processors would wind up acting simultaneously on nearest neighbor sites. Instead of addressing identically in each processor we therefore place data associated with site x_4 in the processor with coordinates x_1 , x_2, x_3 at the location which homogeneous addressing would have used for the site

$$x'_4 = (x_1 + x_2 + x_3 + x_4) \bmod 8. \qquad (4)$$

A single address broadcast by the central controller then corresponds to physical sites, in nearest neighbor processors, which differ by at least 2 units of lattice spacing. By assigning the physical lattice periodic boundary conditions in all directions and making appropriate use of the address remap capabilities of the central controller, the iteration of eq. (3) for the address pattern of eq. (4) can be driven by microcode nearly identical to the microcode which would have been required for homogeneous addressing.

Addressing according to eq. (4) and its generalizations is also useful for Monte Carlo calculations in which distinct processors update distinct sites in parallel. Monte Carlos of this sort arise in approximate versions of QCD and for lattice systems in condensed matter physics. For such calculations again it is often necessary to insure that sites which are physical neighbors are not acted on simultaneously.

The combination of hardware, software and algorithms which we have chosen, we expect will yield a sustained performance of 10 GFLOPS. For the calculations we an-

ticipate a total consumption in the neighborhood of 3×10^{17} operations will be required. As pointed out in the introduction, this will use 1 year on GF11. The time requirements for such calculations on other present and proposed computers in shown in Table 1.

Finally, it may be useful to mention that performance comparable to what GF11 is expected to deliver for QCD can probably also be obtained for a variety of other scientific and engineering applications. The main feature of QCD which makes it amenable to treatment using our modified SIMD architecture is that the theory governs a physical system (fields on space-time) composed of a large number of individual constituents (field values at individual sites) each obeying the same law. A second feature of the QCD calculation, somewhat less important than the first but still a help, is that nearly all the detailed arithmetic, for example in updates by eq. (3), involves at least some chaining. This permits the register file on each processor to compensate for limited memory bandwidth. Both of these features are present in a wide range of problems in solid state physics, in the physics of fluids, and in design simulation.

Table 1

Machine	CPU time
VAX 11/780	30,000 years
Cosmic cube [6]	3,000 years
Cray I	100 years
Columbia [3]	10 years
GF11	1 year

References

[1] D. Weingarten, "Algorithms for Monte Carlo Calculations with Fermions", IBM Yorktown Technical Report, 1984.

[2] V. Beneš, "Optimal Rearrangeable Multistage Connecting Networks", *Bell System Technical Journal*, vol 43, no 4, Part 2 (July 1964), pp. 1641-1656.

[3] N. Christ and A. Terrano, "A Very Fast Parallel Processor", *IEEE Transactions on Computers*, vol C-33, no 4, April 1984, pp. 344-350.

[4] K. Wilson, "Confinement of Quarks", *Phys. Rev.*, vol D10, 1974, p. 2445.

[5] D. Weingarten and D. Petcher, "Monte Carlo Integration for Lattice Gauge Theories with Fermions", *Phys. Letters*, vol 99B, no 4, Feb 1981, pp. 333-338.

[6] C. Seitz, "Experiments with VLSI Ensemble Machines", *Journal of VLSI and Computer Systems*, vol 1, no 3, Computer Science Press, 1984.

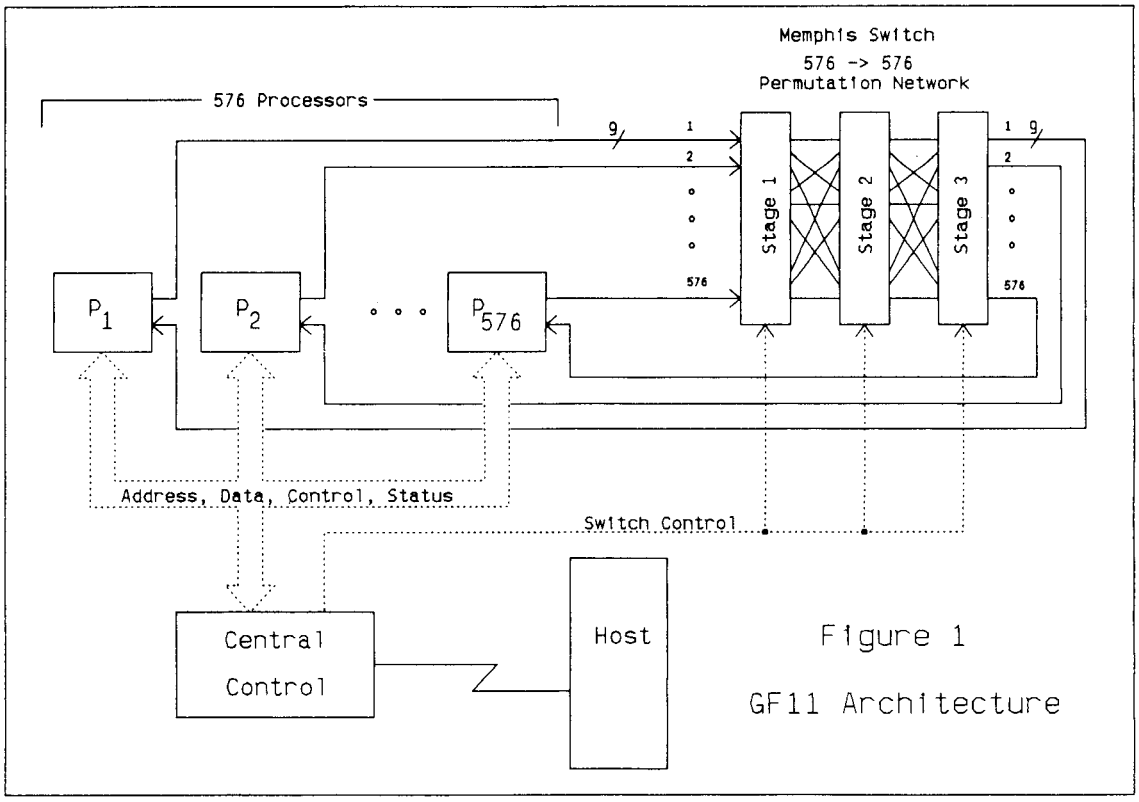


Figure 1

GF11 Architecture

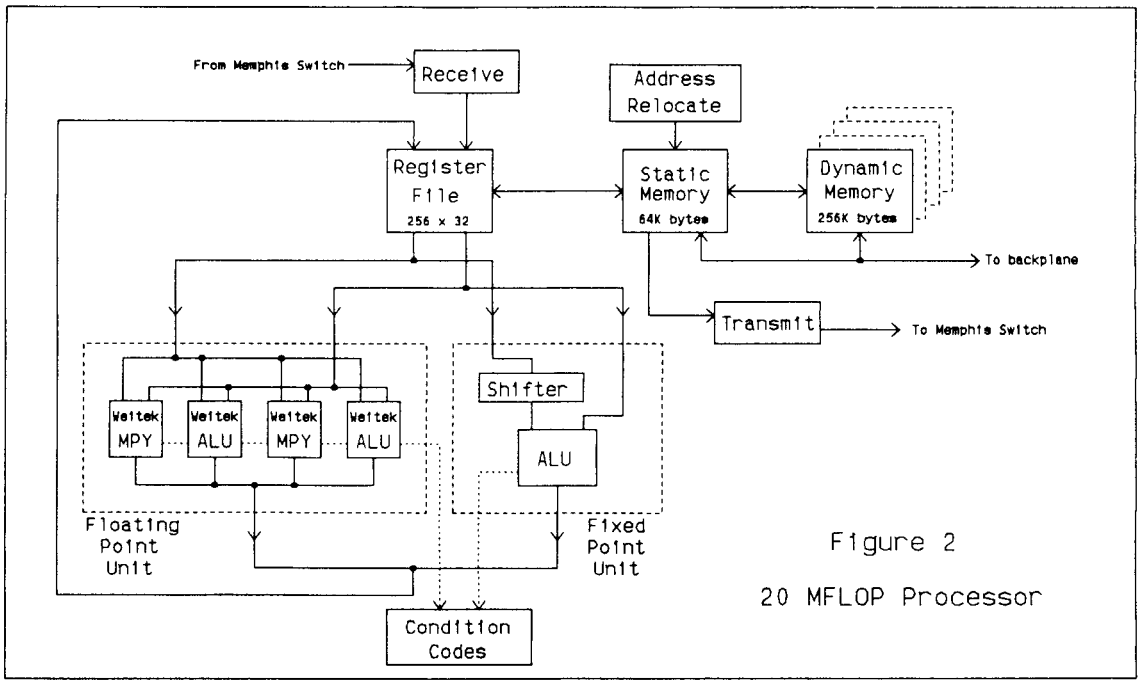
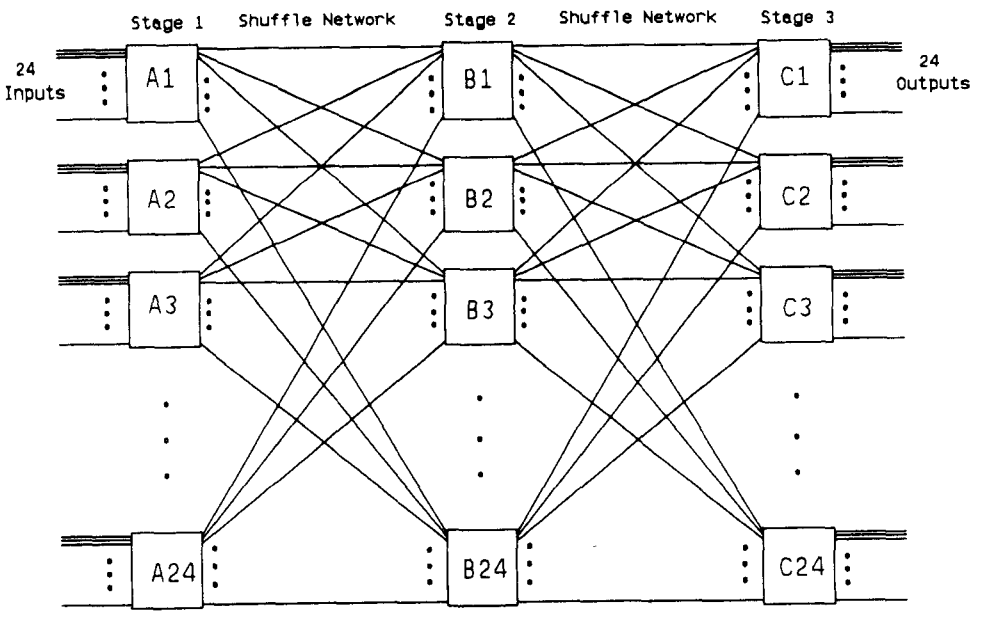


Figure 2

20 MFLOP Processor



Total number of inputs and outputs is 576.
Each square is a 24 - 24 crossbar switch.
All lines represent 9 bit data paths.

Figure 3
Memphis Switch

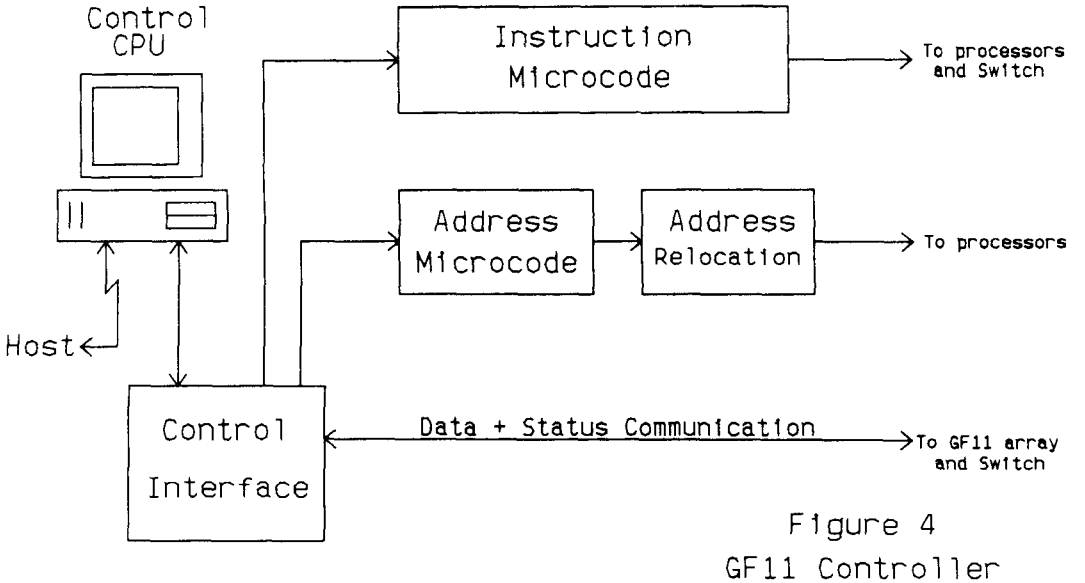


Figure 4
GF11 Controller