Unique design concepts in GF11 and their impact on performance

by M. Kumar

GF11 is a 512-way SIMD parallel computer currently used to verify quantum chromodynamics theory and to explore the SIMD approach to parallel processing. System design choices, such as network design, processing element design, and other architectural features, allow GF11 to sustain very high performance, close to the 10-gigaFLOPS peak. Several applications, such as structural analysis, seismic modeling, computational fluid dynamics, and linear algebra, have been ported to GF11. Applications execute in the range of 4 to 10 gigaFLOPS. The diversity in applications that perform well on GF11 demonstrates that the SIMD architecture is effective for a much larger set of applications than previously believed. The high network and data-memory bandwidths minimize the effort required to tune applications for optimum performance.

Introduction

GF11 is a 512-way SIMD (single-instruction-stream multiple-data-stream) parallel computer that is operational at the IBM Thomas J. Watson Research Center [1–7]. Though its peak performance of 10 gigaFLOPS (GFLOPS) has been surpassed by several other computers, it is still unique in its ability to sustain performance close to its peak on a wide range of scientific and engineering applications. (The floating-point chips and memory technology used in GF11 are almost two generations older than those used in current parallel computers. The performance of floating-point arithmetic logic unit chips and the density of memory chips have increased by more than a factor of 10 since the selection of components for GF11 in 1984. Thus, the appearance of computers with higher peak performance is not surprising. However, GF11 has several unique system-design concepts that enable it to sustain a very high fraction of its peak performance on most applications. Therefore, current parallel computers, even with higher peak performance, cannot match the sustained performance of GF11 on several important applications [8, 9].)

The GF11 hardware consists of 566 identical processors and 10 disk drives connected through a 576 × 576 Benes network [10]. A central controller controls the operation of all processors, disk drives, and the network. (It is possible to replace the 10 disks with processors so as to have 576 processors, as described in [2], or to replace processors

©Copyright 1992 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

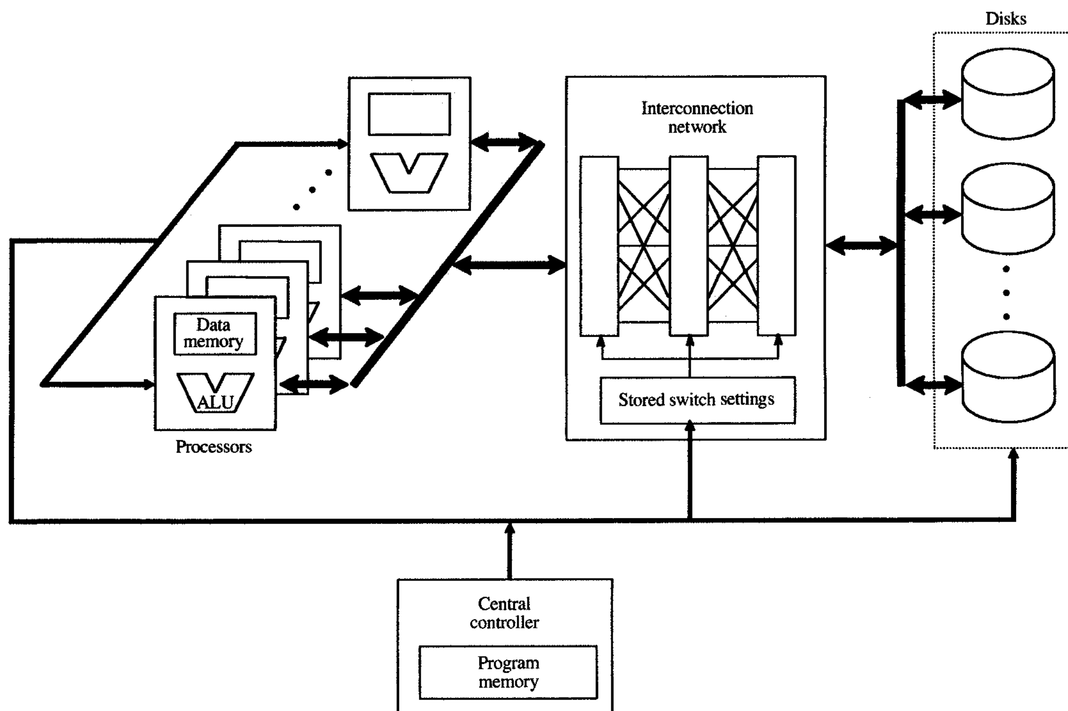

Figure 1

Hardware organization of the GF11 parallel computer.

with disk drives.) Each processor has a peak performance of 20 megaFLOPS (MFLOPS). (Each processor has two floating-point adders and two floating-point multipliers, all operating at 5 MFLOPS. The peak processor performance of 20 MFLOPS is achieved when all four floating-point units are continuously busy.) Therefore, the maximum possible performance of a 566-processor system is 11.3 GFLOPS. However, 500 or 512 of the 566 processors, depending on how the data in an application can be partitioned, are intended to be used by an application program at any given time, and the remaining ones function as spares. Thus, from the application point of view, the peak performance of GF11 is 10 GFLOPS.

A high-level overview of GF11 is shown in **Figure 1**; details can be found in [2, 4]. Each processor has its own data memory (there is no shared data memory). A single copy of the program exists in the program memory in the central controller, and instructions broadcast from the program memory are executed by all active processors as soon as they are received by the processors. The whole system operates with a 50-ns machine cycle, and the network transports one byte of data received from the processor at every input in each cycle.

GF11 was designed by John Beetem, Monty M. Denneau, and Donald H. Weingarten, and is described in detail in [1–7]. Several individuals made significant contributions to turn the design into an operational system (see the Acknowledgments), and Yurij A. Baransky played a pivotal role in demonstrating the usability of GF11. Many applications have been programmed on GF11 since it became operational in November 1990 [4, 5]. In the next few sections of this paper we describe the unique design concepts of GF11, such as the use of a Benes network to provide conflict-free, low-latency interprocessor communication for a large number of communication patterns (up to 1024), system-wide synchronous operation, the ability of each processor to perform multiple operations concurrently in each machine cycle, and the balanced design of the memory system. GF11 is the only parallel computer in which interprocessor communication is synchronous with respect to the computations in the processors. We also discuss how these design

choices contribute to the efficient operation of the entire system.

In the last section of this paper, we analyze the outcome of programming various scientific and engineering applications on GF11. Not only did this programming activity prove the effectiveness of the GF11 design concepts in enabling GF11 to sustain more than 50% of its peak performance on a large number of applications, but it also demonstrated that good designs for the network and memory subsystems reduce the effort required for restructuring the application programs or developing new algorithms for optimal performance. The significant effect of the network approach for interprocessor communication on the performance of applications is also discussed.

Prior to the availability of current SIMD machines [11, 12], the SIMD architecture was widely believed to be effective for only a small set of scientific and engineering applications—those characterized as using explicit finite difference methods on structured grids. Nonetheless, applications outside this narrow domain have been successfully programmed on GF11 and other parallel machines. These new application results [12–15] and analyses [16] suggest a much wider applicability of the SIMD architecture.

Innovative use of Benes network for interprocessor communication

The earlier SIMD computers connected processors directly with one another in simple topologies, primarily the mesh connection [17–20]. Communication on the network was controlled directly by the instructions of the application program. In these computers, communication was very efficient if the communication pattern of the program conformed to the connectivity of the processors (for example, each processor communicating only with its nearest neighbors on a 2D mesh of processors). Communication could be optimally coordinated with the calculations. However, more general communication patterns in a program had to be broken down by the user or compiler into simpler steps that conformed to the connectivity of the processors, and the communication was extremely inefficient for many of these patterns.

In addition to the above type of communication mechanism, current SIMD computers use the hypercube (e.g., MasPar [11]) and shuffle-exchange (e.g., CM-2 [12]) types of networks. Because of their "self-routing" nature, these networks can handle more general patterns of interprocessor communication, but they operate asynchronously with respect to the processing elements and introduce significant overhead into the communication process. The sources of this overhead are analyzed below. This communication overhead also adversely affects performance of MIMD (multiple-instruction-stream multiple-data-stream) computers [21, 22]. By using a Benes network, GF11 can provide full connectivity between the processors without incurring the overhead of asynchronous operation.

Benes networks were proposed for handling telephone traffic in 1961 [10], and their use in parallel processing was suggested in the early eighties [23–25]. However, GF11 is the only parallel computer that uses this type of network for interprocessor communication. The approach of using a set of preprogrammed switch settings, discussed below, is the idea essential to the use of the Benes network in a SIMD parallel system. The key characteristics of this network and their contribution to the efficiency of the overall system are discussed below. The network is perhaps the most important factor that allows GF11 to sustain performance close to the peak on a wide variety of applications.

- *Characteristics of the GF11 interconnection network*

The communication network in GF11 is a three-stage Benes network, constructed from 24×24 switches with 24 such switches in each stage. The data paths in the switches and network are one byte wide. In an ordinary Benes network, these switches must be able to connect the 24 inputs to the outputs according to any permutation. The 24×24 switches used in GF11, however, are more general: They allow inputs to be connected to multiple outputs. Each of several switch inputs can broadcast its data to multiple switch outputs simultaneously, as long as two inputs do not attempt to broadcast to the same output. This communication pattern, in which each of multiple inputs simultaneously broadcasts its data to multiple nonoverlapping outputs, is known in the literature as *multicast* communication. Using this structure, the GF11 network has the following characteristics:

Full connectivity Benes networks have the same functional capability as full crossbar switches; i.e., they can provide connections from all the network inputs to the network outputs simultaneously, according to any specified permutation. Networks with this property are known in the literature as *nonblocking* networks. Furthermore, the added capability of the switches to perform multicast communication allows the GF11 network also to perform multicast communication efficiently. The network can deliver data from each of its several network inputs to multiple nonoverlapping network outputs simultaneously. However, while any permutation can be performed by sending data once over the network, data must be sent over the network twice to perform a multicast. In the first pass over the network, the data broadcast from each input is replicated within the network in order to create as many copies as the number of outputs to which it must be broadcast. Then, in the second pass, the correct number of copies available for each broadcast item are permuted in

order to deliver them to the desired outputs. Details of the algorithm used to perform multicast communication can be found in [26].

Fixed delay In a SIMD parallel processing environment, the nonblocking property implies that if in a given cycle each processor sends a word of data to another, according to any permutation between the sending and the receiving processors, all processors will receive their data simultaneously after a fixed number of cycles. The delay depends only on the hardware implementation of the Benes network and is independent of the pattern of communication.

Use of precomputed switch settings Benes networks require substantially less hardware than full crossbar switches, at the expense of requiring a more complex algorithm to calculate the switch settings needed to provide each specified permutation. In contrast, the frequently used multistage networks in MIMD computers use very simple algorithms, which are essentially implemented in switch hardware. The switches determine their own settings by examining the destination-address part of each message packet while the packet, consisting of the data and the destination address, passes through the network. Networks of this type are known as *self-routing* networks.

The drawback of requiring complex algorithms (therefore, complex hardware) to determine switch settings is eliminated in GF11 by allowing the switch settings for a predeclared set of communication patterns to be computed and loaded into the switches in the network. The network can store the switch settings for 1024 permutations, and a switch setting, once stored, can be used by simply broadcasting a 10-bit switch-setting number from the central controller to the network. When all processors simultaneously send a word across the network as a result of executing an identical instruction received from the central controller, the 10-bit switch-setting number is simultaneously broadcast from the central controller to all the switches in the network and is used to select the precomputed switch-setting for the desired permutation. Thus, even the simplest routing algorithm need not be executed in the switch.

- *Advantages of the GF11 network*

The GF11 interconnection network, because of its above-mentioned characteristics, has several advantages over the most commonly used self-routing network. These advantages, discussed next, allow the network to efficiently meet the interprocessor communication requirements of almost all applications we have studied and to hide the communication overhead completely by overlapping it with the calculations being performed by the processors and by eliminating the involvement of the processors in the communication process.

High bandwidth sustained for any definable interprocessor connectivity pattern In almost all applications studied, GF11 processors exchange data with other processors at a rate close to the peak network bandwidth of 20 megabytes (MB) per second per processor ($\approx$11 GB/s for 566 processors). Since the peak performance of each processor is 20 MFLOPS (a total of $\approx$11 GFLOPS), the ratio of communication bandwidth to computational power for arbitrary communication patterns is almost two orders of magnitude better than that of most contemporary parallel computers [27, 28]. This is primarily because of the nonblocking nature of the Benes network.

Since the network can support any permutation or multicast communication pattern, the processors can easily be configured or logically connected as arrays of various dimensions and of different extents in each dimension, or as rings, tori, butterflies, trees, etc. [Specifying a permutation for the GF11 network is equivalent to specifying the interprocessor connectivity pattern or configuration. For example, assume that the P processors are numbered from 0 to $P - 1$. If the permutation pattern on the Benes network connects network input i ($0 \leq i < P$) to network output $i + 1$ modulo P, the processors are logically connected or configured as a ring.] An application program may use several configurations during its execution. As long as these configurations are declared in the application program, switch settings can be computed by the compiler for each configuration and loaded into the switch when the program is loaded into memory, or computed and loaded into the switch as a part of the program initialization phase (depending upon whether the configuration is known at compile time). Settings for 1024 configurations can be stored in the switches of the network. During the execution of the program, the central controller can select any stored configuration for interprocessor communication by broadcasting the switch-setting number to the network.

The configuration to be used is specified in every GF11 instruction that causes data to be transferred over the network, and is found in the "switch-setting number" field in the instruction. Word transfer for the previous instruction is completely overlapped with switch setting for the current instruction. Since transfer of a word (four bytes) takes four cycles (200 ns), a GF11 program could change configurations five million times a second while maintaining the data transfer rate of 20 MB/s per processor.

No packetization/depacketization overhead To send a message from one processor to another in a parallel computer with a typical self-routing network, the sending

processor must first form a packet consisting of the data to be sent and the address of the destination processor. An operating system routine is invoked to form the packet and relay it to the network interface. The destination address and data to be transmitted are provided as parameters by the user program to the operating system routine called. Similarly, the receiving processor also invokes an operating system routine to receive the packet from the network interface and to retrieve the data field from this packet. Depending on the software implementation, this process can require the processors to execute from a few extra instructions to several thousand.

By comparison, completing the transfer of a word over the network in GF11 uses only a few fields of one instruction. An instruction in the single program controlling all processors specifies a memory-read operation, one bit to specify that the data being read out of the processor memory should be placed on the network interface, the number of the precomputed switch setting to be used by the network, and a final bit to specify that the processor register file should receive a data word from the network interface after the fixed network delay. For most cases there is no difference between a simple load operation from their own data memories and the transfer of a word over the network from remote processor data memories. (In very rare cases, in which network data transfers are required more frequently than once every four instructions, additional delay occurs because of the byte-wide data path.)

Simple hardware, no queues, no queuing delays From the hardware point of view, the network interfaces in GF11 are also trivially simple. Both the send and receive interfaces are simply shift registers to convert between 32-bit processor data paths and byte-wide network data paths. The simplicity of network interface hardware minimizes the latency through the network. Also, since the network is nonblocking, there is no need to queue the messages in the switches or the network interface. This simplifies the switch design.

Shorter messages, no routing delays In a self-routing, packet-switched network, each switch must examine the destination-address part of the message to determine its route through the network. This adds to the delay through the network. With precomputed routes for the flow of data through the network, the GF11 network does not incur this added delay. Also, since the route of the data has been precomputed, there is no need to carry the destination address with the data. This results in shorter messages and more efficient utilization of the network bandwidth.

Replacement of failing processors by spares Since Benes networks can support all permutations, any subset of processors can be chosen from the set of available processors for running an application. Network faults can also be avoided by not using the processors affected by the faulty network switches. The application program is always written in terms of logical processors and the logical connectivity patterns between them. The run-time environment maintains a list of operational (physical) processors, chooses a subset of the operational processors, and maps them to logical processors. The switch settings are then computed to provide the logical connectivity requested by the application.

Checkpoints are inserted in lengthy applications, and if faulty processors are discovered (by running the diagnostics or looking at the status information in the processors), the mapping between the logical processors and the remaining good processors is performed, including the recalculation of the switch-settings, so that the faulty processors may be replaced and the calculation restarted from the previous checkpoint.

System-wide synchronous operation

Another very important, unique feature of GF11 is that the operation of the multistage network, like that of the processing elements, is controlled by the application program instructions. This is possible because delays through the Benes network are fixed. Thus, calculations within processors and communication over the network are synchronized. Similarly, the transfer of data from the I/O disks to and from the processor local memories is also synchronized with the calculations in the processors. This is accomplished by using buffers with the disks that can transmit data to and receive data from the GF11 network synchronously, under the control of the program instructions. In fact, all components of the GF11 system (the central controller, the instruction-distribution bus, the I/O buffers, and the network) operate synchronously, using a common clock distributed throughout the GF11 system.

In contrast, only the processing elements operate synchronously in other SIMD computers [11, 15]. Thus, their interconnection network delays cannot be determined at compile time, because the networks used by the other computers are self-routing and blocking. Therefore, the operation of the network cannot be synchronized with the operation of the processing elements. Since data from the I/O devices are transferred to the processor data memories through the network, data transfer from the I/O devices cannot be synchronized with the calculations either. The advantages of having system-wide synchronous operation are given in the following subsections.

Low synchronization overhead Most applications implemented on parallel computers have a large number of computation steps or phases, usually separated by communication steps. If the communication delays are

nondeterministic, the processors must be synchronized after each communication step, especially in a SIMD computer, because synchronization is the only mechanism to ensure that data transmitted over the network in a communication step arrive at the processors before the computation step that uses them begins.

In GF11, operation of the network and of the processing elements is controlled by the same instruction sequence. When a communication step initiated by some instruction causes the processors to send data out to the network, all the receiving processors are guaranteed to receive their data in the same cycle following the fixed network delay. Therefore, no synchronization code is necessary to detect the completion of a communication step.

Opportunity for overlapping calculations with communication and I/O operations Since the delay through the network is constant and all the switches in the network as well as all the processing elements operate synchronously, using a common clock, the GF11 compiler can optimally interleave computation with communication. For example, following an operation that transmits data over the network, an arithmetic operation that uses the data as an input operand can be initiated after a delay equal to the network delay. The operand, when received from the network, is used immediately in the arithmetic operation, without having to be stored in the receiving processor memory first. This eliminates the additional delay that would be incurred if the data being received from the network were first stored in the memory and then read back as an input operand, and also reduces the likelihood of memory bandwidth becoming a bottleneck.

Similarly, the transfer of data from the I/O-device buffer to the processing-element memory can be initiated by an instruction in the program; then the processing elements can access the data as soon as they arrive, because the delay from the I/O device buffer to the processing element memories is fixed (determined solely by hardware).

Memory organized to provide high bandwidth

The processors in GF11 can overlap the calculations being performed by the arithmetic unit with data transfer from the memory subsystem and with the address calculation arithmetic for memory accesses. Thus, unlike most other computers, GF11 does not require extra cycles for memory accesses and address calculations, and for most instructions the processors perform useful arithmetic operations. The three levels of memory hierarchy, described below, allow processing elements to access operands without incurring memory-access delays.

• *Balanced memory hierarchy*

The organization of the GF11 memory subsystem is shown in **Figure 2**. Each processor in GF11 has 2 MB of memory; therefore, a 512-processor system has 1 GB. Memory on each processor, implemented in DRAM technology, is organized as two banks with 256K 32-bit words each. If the two DRAM banks are accessed alternately, a sequence of load operations or a sequence of store operations can be performed once every four cycles (an arithmetic operation is performed once per cycle). However, because of the hardware design, if consecutive accesses are to the same DRAM bank or the load and store operations are intermixed, the bandwidth to the DRAM is reduced. Without a faster intermediate memory, the DRAM bandwidth would clearly be the bottleneck in most applications.

To prevent the DRAM bandwidth from affecting the performance of GF11, data from the DRAM are moved into a 16K-word buffer implemented in static RAM (SRAM) technology, and from there into a 256-word register file. The SRAM buffer can be accessed once every cycle, and the register file can be accessed four times every cycle (four accesses per arithmetic operation). Two of the four accesses to the register file provide operands for the arithmetic unit, one access is used to store back the result of the arithmetic operation, and the fourth access is used to bring a new data word into the register file from the SRAM or the network or to transfer a result back from the register to the SRAM or the relocation registers for SRAM and DRAM.

The SRAM is used mostly as a user-programmable cache. In many computation-intensive scientific or engineering problems, the data being manipulated consist of large multidimensional arrays, and essentially identical operations are performed on the individual elements of an array. The SRAM buffer is used to hold in one section of the array data from the DRAM and manipulate it (possibly requiring several accesses to it) before writing it back to the DRAM (if it is modified). This mitigates the impact of limited DRAM bandwidth. The register file is large enough to store all the scalar data being used by the program and to store a few vectors of modest size.

• *Hardware support for address calculation*

The memory subsystem contains hardware to support address calculations for data in SRAM and DRAM. Each SRAM address can be modified by adding an offset to it from one of 256 SRAM relocation registers. Two accesses are performed on relocation registers for each instruction: a read access to obtain the relocation amount to be added to the SRAM address being broadcast in the instruction, and a write access to update one of the relocation registers. Each instruction carries, in addition to the SRAM address, the read and write addresses for the SRAM relocation registers. The DRAM address can be modified similarly by adding the offset from the single DRAM relocation register to the DRAM address being

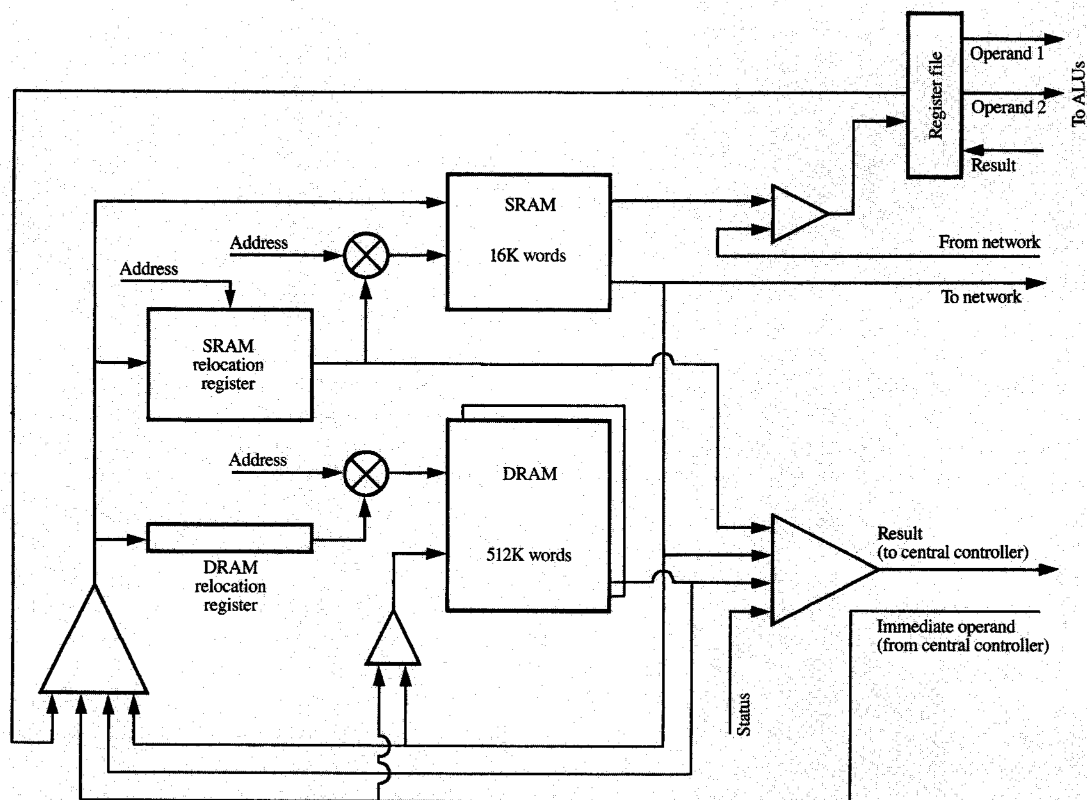

Figure 2

Organization of the GF11 memory subsystem.

broadcast in the instruction. One relocation register suffices for the DRAM, because the DRAM is accessed at most once every four cycles and the relocation values can be obtained from the SRAM, as shown in Figure 2.

The address relocation hardware gives each processor the ability to generate a different SRAM or DRAM address based on local data. This is useful in many situations, some of which are the following:

- Table lookup for calculating transcendental functions, using series expansion.
- Table lookup for properties of materials when simulating a physical medium, where the property has different values for different ranges of some other variable being computed in the processors.
- In combinatorial algorithms, for traversing a tree or a graph according to some rule that prescribes a different path for each processor.
- Indirect addressing into arrays stored locally within the processors, and into large arrays distributed across data memories of all processors. The latter situation occurs very frequently in all unstructured grid applications; an efficient way to handle it on GF11 is explained in the discussion of PAM-CRASH implementation in [5]. The capabilities of the GF11 network are also crucial in handling indirect addressing across large arrays.

Multiple concurrent operations per instruction

It was mentioned in the preceding section that arithmetic operations are overlapped with network operations and memory operations. Actually, multiple activities occur within the arithmetic section and the memory section of the processors. We call such architectures, in which several operations are performed concurrently by each processing element for each instruction, "super-scalar."

Table 1 Performance sustained by GF11 on scientific/engineering applications.

Application	*Number of processors used*	*Performance* (GFLOPS)	*Problem size*
PAM-CRASH (finite element method)	500	5.5	10,000 elements
TPP (linear algebra, LU decomposition)	500	4.3	1000 × 1000
TPP (linear algebra, LU decomposition)	500	5.6	2500 × 2500
TPP (linear algebra, LU decomposition)	450	7.4	5400 × 5400
Gaussian elimination	500	9.3	2500 × 2500
Gaussian elimination	500	9.5	6000 × 6000
2D FFT	512	7.2	1024 × 1024
Shallow Water equations (weather code)	512	7.5	256 × 256
Matrix multiplication	512	10.0	1024 × 1024

The GF11 processors can perform the following operations concurrently in each instruction (50-ns cycle time):

- An arithmetic operation (integer or floating point), for which the operands are received from the register file and the computed result is stored back in the register file.
- A shift or rotate (logical or arithmetic) operation on one input operand, if the simultaneous arithmetic operation is an integer or logical operation.
- A read or write operation on the SRAM. For a read operation, the data retrieved from the SRAM can be sent to the register file, the DRAM, the network, another location of the SRAM itself, the relocation registers, or the central controller. (If multiple processors send data to the central controller simultaneously, the central controller receives the "logical OR" of the data sent.) In the case of a write operation, the data to be loaded can come from the register file, the DRAM, an immediate operand in the instruction, or another location in the SRAM itself.
- A write operation to the SRAM and DRAM relocation registers. The data written are the same as those available for writing to the SRAM during that cycle.
- Translation of the SRAM address using SRAM relocation registers, as described in the preceding section.
- Selecting a bit from the several condition code bits generated by the current operation, and storing it in one of the eight 1-bit condition code registers in the processor.
- Performing five read operations on the condition code registers, and using the logical values read back to conditionally disable the write operation into the SRAM or DRAM, to disable the data transfer to the central controller, and to modify the network and integer ALU operations. Thus, even though all processors receive the same instruction, different operations are performed for it in different processors, depending on the logical values stored in the condition code registers.

In addition to the above operations, which are performed for every instruction, the following DRAM and network operations can occur once every four instructions:

- A word is transferred over the network according to the specified communication pattern. It takes four cycles to transfer a word because the network is one byte wide.
- A new communication pattern is selected for the next word transfer.
- A word is stored into the DRAM (either from the SRAM or the immediate operand), or a word is read from the DRAM. The DRAM-bank cycle time of four cycles is a technology limitation. The compiler must determine that no DRAM-bank conflict will occur; bank conflicts reduce the access rate.

The instruction broadcast from the central controller has 201 bits. Of these, 177 are sent to the processors, and the remaining ones control other parts of the system. In the 177 bits reaching the processors, 85 are used as addresses for various memories and registers, 32 are used as the immediate operand, and the rest directly control the operations in the processors.

Though the processors in GF11 perform only one arithmetic operation in each instruction, they do many other operations to move data, so that operands are available for an arithmetic operation in every cycle. This contributes significantly to the ability of GF11 to sustain near-peak performance on most applications. Most other processors spend a significant number of cycles to get the operands to the arithmetic units.

Summary of application studies

Since GF11 became operational, several scientific/engineering applications have been programmed on it. Some of the representative applications are listed in **Table 1**. The key application remains quantum chromodynamics. More detailed discussion on the implementation of these applications on GF11 can be found in [3–7, 29]. The decision concerning the number of

processors to use depended upon how the problem could be conveniently partitioned. The key observations from this programming effort are as follows.

High sustained performance
It is quite clear from Table 1 that GF11 sustained more than 50% of its peak performance on all of the applications. This ratio of actual performance to peak performance is significantly better than that being observed on the currently available parallel computers [8, 9]. Imbalance between the number of add and multiply operations in a GF11 application is usually the primary cause of the gap between the peak and actual performance. To achieve the peak performance, the two floating-point adders and two floating-point multipliers in each processor must be continuously busy, requiring the application to have an equal number of add and multiply operations. This imbalance accounts for almost all of the performance degradation in the Shallow Water benchmark [14] and half of the performance loss in the PAM-CRASH benchmark [5, 30].

Network bandwidth limitations account for the remaining performance loss in PAM-CRASH, which could have been avoided if domain decomposition [31] or some similar approach had been used to reduce the network traffic, instead of the most straightforward data-partitioning scheme, which was chosen for this implementation. The memory or network bandwidths rarely become serious bottlenecks in GF11.

The performance of GF11 on the TPP benchmark for a 1000×1000 matrix (4.3 GFLOPS) is still the highest absolute performance achieved on any computer for a matrix of this size [32]. For this matrix size, the performance loss is primarily due to the distributed memory aspect of GF11, which creates load imbalance during the execution of the program [5]. Naturally, for a larger matrix (6000×6000) the performance improves. Though current parallel computers can claim higher absolute performance on even larger matrices, their ratios of actual performance to peak performance are substantially lower.

In addition to the applications listed in Table 1, a neural network simulation program [33], a program to simulate the evolution of galaxies [4], an FFT program, and a wave mechanics application from Sandia National Laboratories [34] have also been implemented on GF11. All of these applications sustain good performance (in excess of 50% of the peak) on GF11.

Ease of developing and debugging new programs
The programming model for SIMD computers is inherently simpler than the programming model for MIMD computers, because there is a single flow of control. In the programming model for MIMD computers, one must generate the multiple flows of control and coordinate them. The MIMD programming model is less restrictive but more complex. Most of the programs implemented on MIMD computers follow the SIMD programming model [16].

Because of the system-wide synchronous operation, it was easy to develop an instruction-level simulator for a GF11 system comprising a few (4 to 16) processors and small data memories. This simulator proved extremely useful in initial development and debugging of GF11 programs. Since the simulator runs on an IBM RT PC® workstation, the initial application development did not depend on the availability of GF11 hardware.

The SIMD architecture and system-wide synchronous operation also guarantee that program execution can be repeated with identical results. In GF11, the application program can be stopped after the execution of any instruction, and the values of its data structures can be examined. This further simplifies the task of debugging programs.

Ease of programming for optimum performance
The architectural features that enable GF11 to sustain high performance consistently also simplify the task of achieving this performance. Because of the high network and memory bandwidths, a simple partitioning of the program suffices in most cases because it does not usually create memory or network bottlenecks. In most other parallel computers, such bottlenecks appear more readily, and complex program transformations and data partitioning are required to circumvent them. The research community in parallel processing has already invested significant effort in developing new algorithms to efficiently use parallel computers with limited interprocessor communication bandwidth, and users of these algorithms may perceive the GF11 network as overkill. However, we believe that new applications will continue to be developed, and architectures like GF11 will accelerate the development of new applications for parallel computers by freeing the programmer from the burden of program transformations and having to invent new parallel algorithms.

For example, if the standard matrix multiplication algorithm, in which the inner product to determine one element of the result matrix is performed in the innermost loop, were partitioned naively on GF11 to calculate one column of the results matrix on every processor, GF11 would still sustain 5.0 GFLOPS, even though the computation-to-communication ratio has dropped to 0.5 FLOPS per byte from $\approx$40 in the more sophisticated partitioning described in [4]. To achieve 10-GFLOPS performance, it was necessary to restructure the program [4].

In the Shallow Water benchmark included in Table 1, the 2D matrix data were assigned one row per processor. This partitioning is much simpler than the blockwise

partitioning necessary in other computers, and it uses simpler interprocessor communication patterns. It was possible to use the simpler partitioning because the network could handle the extra communication generated by it without increasing the computation time. Similarly, in the PAM-CRASH application, data were partitioned in a straightforward manner, and domain decomposition algorithms were not needed to reduce the communication overhead.

The execution time of a sequence of instructions on GF11 can be determined precisely from the number of instructions in the sequence, because each instruction takes exactly one cycle to compute. (In traditional processors, however, a load operation, for example, takes different times to complete depending on whether the data are in the cache or not.) Even the conditional operations take one cycle, irrespective of whether they are performed or not. Therefore, the performance of a program can be readily determined by counting the number of instructions in the compiled code of various subroutines, and performance tuning can be done without actually running the programs.

Conclusions

By sustaining more than 50% of its peak on several types of scientific and engineering programs, GF11 has provided strong evidence that the SIMD architecture is effective for a much larger set of applications than has been credited by conventional wisdom. We continue to program new applications to explore the limits of this set. We have also shown that the new system-design concepts employed in GF11 are effective in enabling GF11 to sustain near-peak performance on different types of applications. Memory or network bandwidths have never been found to be the bottlenecks.

High network and memory bandwidths significantly simplify the task of mapping programs to multiple processors and distributing the data to the memory, because the extra network and memory traffic generated by less-than-optimal mapping does not cause the network or memory to become a bottleneck and, therefore, does not affect the performance of the system. Finally, the nonblocking nature of the network and its operation in synchronism with the processors make interprocessor communication very efficient, even for one-word data transfers. The GF11 network can sustain its peak bandwidth for almost all types of interprocessor communication.

Acknowledgments

We acknowledge the effort of Yurij Baransky, Michael Cassera, Molly Elliot, David George, Edward Nowicki, James Sexton, Michael Tsao, and Donald Weingarten in making GF11 operational. Early work on implementing LU decomposition and FFT was done by Michael Witbrock and Banu Ozden, and is discussed in detail elsewhere. We also thank Piero Sguazzaro for helping us with the PAM-CRASH code, and Richard Lawrence for helping us with the Shallow Water equations code.

RT PC is a registered trademark of International Business Machines Corporation.

References

1. J. Beetem, M. Denneau, and D. Weingarten, "The GF11 Parallel Computer," *Experimental Parallel Processing Architectures*, J. J. Dongarra, Ed., Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1987.
2. J. Beetem, M. Denneau, and D. Weingarten, "The GF11 Supercomputer," *Proceedings of the 12th International Symposium on Computer Architecture*, IEEE Computer Society Order No. 634, June 1985, pp. 108–115.
3. M. Denneau and D. Weingarten, "The GF11 Processor and Compiler," *Lattice Gauge Theory Using Parallel Processors*, X. Li, Z. Qiu, and H. C. Ren, Eds., Gordon and Breach, New York, 1987, pp. 303–326.
4. Manoj Kumar and Y. Baransky, "The GF11 Parallel Computer: Programming and Performance," *Future Generation Computer Systems* **7**, No. 2&3, 169–179 (1992).
5. Manoj Kumar, Y. Baransky, and M. Tsao, "The GF11 Parallel Computer," *Research Report RC-16596*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1991.
6. J. Sexton, "The Status of GF11," *Lattice 88: Proceedings of the 1988 Symposium on Lattice Field Theory*, A. S. Kronfeld and P. B. Mackenzie, Eds., *Nucl. Phys. B (Proc. Suppl.)* **9**, 566–570 (1989).
7. D. Weingarten, "The Status of GF11," *Lattice 89: Proceedings of the 1989 Symposium on Lattice Field Theory*, N. Cabbibo et al., Eds., *Nucl. Phys. B (Proc. Suppl.)* **17**, 272–275 (1990).
8. *International Journal of High Speed Computing* **1**, No. 2 (June 1989); *Proceedings of the Conference on Scientific Applications of the Connection Machine*, NASA Ames Research Center, CA, September 12–14, 1988.
9. *Proceedings of Supercomputing '89*, Reno, NV, November 13–17, 1989.
10. V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, Inc., New York, 1965.
11. Tom Blank, "The MasPar MP-1 Architecture," *Proceedings of IEEE COMPCON Spring '90*, February 1990, pp. 20–24.
12. L. W. Tucker and G. G. Robertson, "Architecture and Application of the Connection Machine," *COMPUTER* **21**, 26–38 (1988).
13. John R. Nickolls, "The Design of MasPar MP-1: A Cost Effective Massively Parallel Computer," *Proceedings of IEEE COMPCON Spring '90*, February 1990, pp. 25–28.
14. R. K. Sato and P. N. Swarztrauber, "Benchmarking the Connection Machine 2," *Proceedings of Supercomputing '88*, pp. 304–309.
15. D. L. Waltz, "Applications of the Connection Machine," *COMPUTER* **20**, 85–97 (1987).
16. G. C. Fox, "What Have We Learnt from Using Real Parallel Machines to Solve Real Problems?" *Caltech Report* C^3P–522, California Institute of Technology, Pasadena, March 1988.
17. G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, "The ILLIAC IV Computer," *IEEE Trans. Computers* **C-17**, 746–757 (1968).

18. K. E. Batcher, "Design of a Massively Parallel Processor," *IEEE Trans. Computers* **C-29**, 836–840 (1980).
19. P. M. Flanders, D. J. Hunt, S. F. Reddaway, and D. Parkinson, "Efficient High Speed Computing With Distributed Array Processor," *High Speed Computer and Algorithm Organization*, D. J. Kuck, D. H. Lawrie, and A. H. Sameh, Eds., Academic Press, Inc., New York, 1977, pp. 113–128.
20. D. L. Slotnick, W. C. Borck, and R. C. McReynolds, "The SOLOMON Computer," *Proceedings of the AFIPS 1962 Fall Joint Computer Conference*, 1962, Vol. 22, pp. 97–107.
21. W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milliken, and T. Blackadar, "Performance Measurements on a 128-Node Butterfly Parallel Processor," *Proceedings of the 1985 International Conference on Parallel Processing*, IEEE Computer Society Order No. 637, August 22–23, 1985, pp. 531–540.
22. J. P. Hayes, T. Mudge, Q. F. Stout, S. Colley, and J. Palmer, "A Micro Processor-Based Hypercube Supercomputer," *IEEE Micro* **6**, 6–17 (1986).
23. G. Lev, N. Pippenger, and L. G. Valiant, "A Fast Parallel Algorithm for Routing in Permutation Networks," *IEEE Trans. Computers* **C-30**, 93–100 (1981).
24. D. Nassimi and S. Sahni, "A Self-Routing Benes Network and Parallel Permutation Algorithms," *IEEE Trans. Computers* **C-30**, 332–340 (1981).
25. D. Nassimi and S. Sahni, "Parallel Algorithms to Set Up the Benes Permutation Network," *IEEE Trans. Computers* **C-31**, 148–154 (1982).
26. Manoj Kumar, "Supporting Broadcast Connections in Benes Networks," *Research Report RC-14063*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1988.
27. *The Connection Machine CM-5 Technical Summary*, Thinking Machines Corporation, Cambridge, MA, October 1991.
28. R. Stevens and P. McDonough, "Instruction Timings and Message Passing Performance of the Connection Machine 2," *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, Monterey, CA, March 6–8, 1989.
29. F. Butler, H. Chen, J. Sexton, A. Vaccarino, and D. Weingarten, "Volume Dependence of the Valence Wilson Fermion Mass Spectrum," *Lattice 91—Proceedings of the 1991 Symposium on Lattice Field Theory*, M. Fukugita, Ed.; *Nucl. Phys. B (Proc. Suppl.)* **26**, 287–289 (1992).
30. *PAM-CRASH: User Manual*, Engineering Systems International, Paris, 1987.
31. R. Glowinski and A. Lichnewsky, "Computing Methods in Applied Science and Engineering," Society for Industrial and Applied Mathematics, Philadelphia, 1990.
32. Jack J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software," *Technical Report CS-89-95*, University of Tennessee, March 1991.
33. M. Witbrock and M. Zagha, "An Implementation of Back-Propagation Learning on GF11, a Large SIMD Parallel Computer," *Parallel Computing* **14**, 329–346 (1990).
34. John L. Gustafson, Gary R. Montry, and Robert E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube," *SIAM J. Sci. & Statist. Comput.* **9**, 609–638 (1988).

Received November 6, 1990; accepted for publication November 22, 1991

Manoj Kumar *IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (MKUMAR at YKTVMH, mkumar@watson.ibm.com).* Dr. Kumar received the B.Tech. degree from the Indian Institute of Technology, Kanpur, India, in 1979, and the M.S. and Ph.D. degrees in electrical engineering from Rice University, Houston, Texas, in 1981 and 1984, respectively. He was manager of the High Performance SIMD Systems group, and was intimately involved with the GF11 parallel computer; he is currently the manager of the Multimedia Information Systems group. His research interests are in architecture and design of multimedia application servers, telecommunication switches, design of massively parallel computer systems, application of parallel computers in solving computation-intensive problems, and interconnection networks for multiprocessor systems.