The Meerkat Multicomputer [*]

Robert Bedichek
Curtis Brown

robertb,cpbrown@cs.washington.edu

Department of Computer Science and Engineering
University of Washington

Abstract

Meerkat is a distributed memory multicomputer architecture that scales to hundreds of processors. Meerkat uses a two dimensional passive backplane to connect nodes composed of processors, memory, and I/O devices. The interconnect is conceptually simple, inexpensive to design and build, has low latency, and provides high bandwidth on long messages. However, it does not scale to thousands of processors, does not provide high bandwidth on short messages, and does not provide cache coherent shared memory. Our hypothesis is that many general-purpose, database, and parallel numerical workloads work well on systems with Meerkat's characteristics. We describe the Meerkat architecture, the niche that Meerkat fills, the motivation behind our design choices, and give performance results obtained from our hardware prototype and a calibrated simulator.

1 Introduction

The need for computational power far greater than that delivered by single-processor systems drives the search for ways to connect many processors together to form *multicomputers*. This paper identifies a currently vacant part of the multicomputer design space and describes an architecture called Meerkat that fills this vacancy.

It is vital that a multicomputer's interconnect be both efficient and effective. By an *efficient* multicomputer interconnect, we mean one whose design cost, implementation cost, and processor and memory overhead are commensurate with its benefits. An *effective* interconnect is one that facilitates efficient use of processors and memory over a range of interesting parallel workloads. We show the performance of Meerkat's interconnect to argue for its effectiveness, and discuss its design and implementation costs to show its efficiency.

Meerkat is characterized by its ability to scale to several hundred nodes, low internode latency, high bandwidth on long messages, short design time, low implementation cost, and a simple fault model. We have taken a software intensive, performance-oriented approach similar to that taken by RISC processor architects in the early 1980's. In attaining Meerkat's good qualities we neither included hardware support for shared memory nor the ability to scale to thousands of nodes. We believe that there are many important multiprocessor workloads and computing environments that are well matched with Meerkat's characteristics.

We use both a four-node hardware prototype and a simulator to measure the performance of a Meerkat implementation. The process of implementing our architecture brought us in close contact with design issues that are easily missed in a paper design. We also use the hardware to calibrate the simulator and for program development. On programs that use four or fewer nodes, the limit of the hardware, the simulator and hardware usually report execution time which differ by less than six percent. We use the simulator to evaluate the performance of Meerkats with up to 256 nodes.

The prototype currently runs both a stand-alone run time environment and the Open Software Foundation's UNIX server on top of the Mach 3.0 microkernel. The stand-alone environment implements a subset of the Intel message passing library and thus allows programs written for this subset to execute on both Meerkat and Intel parallel computers such as the

[*] This is an expanded version of a paper by the same title to appear in the December 1993 SPDP proceedings.

Intel Touchstone Delta [7]. The stand-alone environment is small and makes it easier to measure the performance of the architecture in isolation. All of our measurements reported here use the stand-alone environment. The Mach and OSF-based system, on the other hand, challenges us to support a large operating system.

We present the Meerkat hardware in Section 2. Section 3 discusses tradeoffs in multicomputer interconnect design that motivated the choices we made. Measurements given in Section 4 show that Meerkat can attain good interconnect performance in practice. The design of the Meerkat hardware prototype is outlined in Section 5 and A paper design of a Meerkat implementation with current technology is detailed in Section 6. Section 7 discusses related work.

2 The meerkat hardware

This section describes the structure of a Meerkat system and the lowest level of interaction between processors, memory, and the interconnect.

A *node* is a processor tightly coupled with cache, memory, and optionally I/O devices. Two or more nodes are connected by a set of wires, a *bus*, that they may use to exchange information. Buses are divided into two groups, *horizontal* and *vertical*. Nodes are arranged on a grid and each node *taps* one horizontal and one vertical internode bus with circuits that can send and receive. Figure 1 shows an example of an arrangement of nodes and buses. A pair of nodes that tap the same vertical or horizontal bus communicate with *1-bus* connections. Node pairs that cannot use 1-bus connections must use an intermediate node as a *cross-point* to communicate using *2-bus* connections. Every node is potentially a cross point for 2-bus connections between nodes that share its vertical bus and nodes that share its horizontal bus. Nodes and buses are arranged so that all pairs of nodes can communicate with either a 1-bus or a 2-bus connection. The node sending information *owns* the bus or buses for the duration of the connection through which it sends information to the *receiver*.

A node's processor sees its two bus taps through a *bus interface* that occupies part of the processor's physical address space and is thus accessed via load and store instructions. The bus interface provides five primitive commands: *arbitrate*, *signal*, *data-send*, *data-receive*, and *release*. Each of these operations can be applied to either of the two taps of the bus interface. The arbitrate command instructs the bus interface to acquire control of a vertical or horizontal bus. In addition, it can request that a second bus, an *orthogonal bus*, be acquired by a second node that shares the first bus with the owing node. If the arbitration for both buses is successful the second node acts as cross point for the owning node. The processors in a node acting as a cross point are unaffected unless they attempt to use either of their buses, in which case they will be unable to do so until the owning node relinquishes ownership. The tap connecting an owning node to a bus is said to be *active*.

After the owning node has established a 1-bus or a 2-bus connection and before it may transfer data through the interconnect, it must alert the receiver by using the signal command. This command sets status bits in both the sender and receiver and it requests a processor interrupt in the receiver node. Either the set status bit or the interrupt tells the receiver that it should accept data from the interconnect. To enter a receptive state, the receiver executes the data-receive command. This resets the status bits that were set by the signal command and it resets the interrupt request. Once the sender senses that the receiver node is in a receptive state, it executes the data-send command to copy data from the owning node's memory through its active tap, onto a bus, possibly through a cross point and onto a second bus, through a tap in the receiver node, and into the memory of the receiver node. The sender includes the address of data to be copied from its memory as part of the data-send command. Likewise, the receiver includes the address of the buffer where the data is to be written with its issuance of the data-receive command.

The data sent in a single data-send operation is called a *packet*. Packets may be limited in length and may be restricted to be aligned in memory. Multiple packets may be sent over a connection by repeating the signal/data-receive/data-send sequence. The receiver may elect to mask the internode interrupts and instead poll the status bit in its bus interface to know when a sender is waiting for it to execute a data-receive. This is useful when a receiver knows from looking at one packet that another is due to arrive shortly. The receiver can poll and thus avoid the interrupt latency for each packet received.

When the owning node has sent all the data that it wishes, it releases the bus or buses. Other nodes can then arbitrate and become owners of the bus or buses that were released.

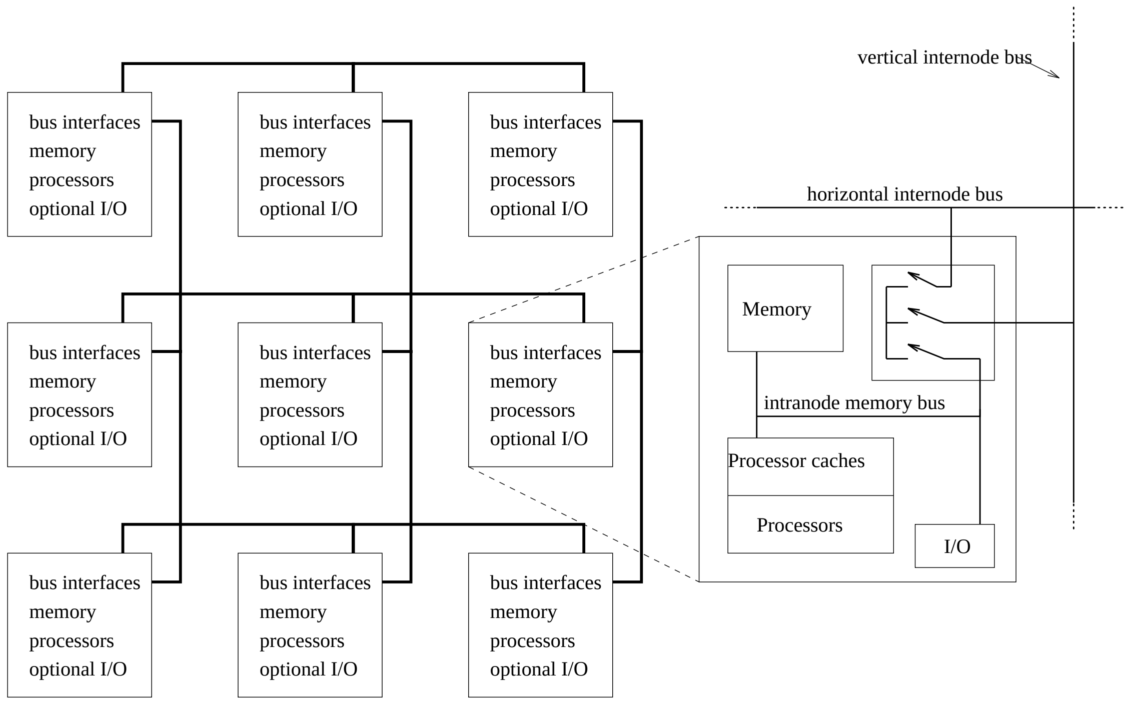

Figure 1: Structure of a 3x3 Meerkat. The rightmost box shows one possible organization of a node.

3 Multicomputer design tradeoffs

This section discusses the design issues that led to Meerkat. We motivate the choices we made in creating Meerkat and contrast Meerkat with other multicomputer architectures.

3.1 Interconnect scalability and economics of implementation

Multicomputer interconnects scale over some range, allowing system configurations with a variable number of processing nodes. The range of nodes over which an architecture is effective is called its scalability, or its scaling range. The Thinking Machines CM-5, for example, is scalable from 32 to 16,384 nodes, a wide range. Most existing multicomputers have a few tens to a few hundred processors despite the ability of manufacturers to make similar models with thousands of processors. There is no lack of demand for the greater computational power of larger systems, rather, size is limited by economics. Nodes of commercial multicomputers, such as the TMC CM-5 [13] and the Intel Paragon [8], use powerful microprocessors and large memories and therefore cost in the range of \$30K to \$100K per node. This puts the cost of a 256 node system, for example, in the range of \$7.5M to \$25M. Although larger systems are important, they are also rare: most prospective consumers of parallel systems can afford systems with a few tens to a few hundreds of nodes, but not thousands of nodes.

Meerkat's interconnect uses passive buses with the number of taps per bus equal to the square root of the number of nodes per system (we assume square configurations of nodes, i.e., an equal number of nodes on each of the horizontal and vertical buses). For electrical reasons this interconnect does not scale well beyond about 16 nodes per bus or 256 nodes per system. However, this limits the ability to scale to sizes that only a few can afford. In return, the simple interconnect model has an implementation which is correspondingly quick to design and is inexpensive to build.

3.2 Interconnect clock rates and planarity of interconnect wiring

Meerkat's interconnect is *planar*, which means that the wires that connect nodes can be routed in a limited number of wiring planes. This allows an inexpensive implementation with a conventional controlled-impedance backplane driven by CMOS integrated circuits. Wiring densities of the printed circuit boards out of which backplanes are made are about 80 wires per inch per plane. Non-planar interconnects must use cables to connect the node circuit boards. The wiring density of cables is approximately 10 wires per inch.

In addition to their order of magnitude advantage in density, printed circuit wires are less expensive and more reliable than cables.

Nonplanar interconnect topologies, such as the hypercube [11], cannot be wired using low cost, high density printed circuit wires. Instead, nodes of these systems are connected by links that use cables which have relatively few wires, often just one. In order to maintain the same data rates, these cables must use much higher clock rates. High clock rates in turn require expensive coaxial or fiber-optic cables and GaAs drivers. Thus, planar interconnects using printed circuit wiring are intrinsically cheaper per unit bandwidth.

The performance of GaAs is not improving as rapidly as CMOS and so we expect the relative advantage of CMOS-driven interconnects to increase. With recent advances in CMOS technology, it is possible today to fabricate a single low cost CMOS gate array that drives over 100 terminated bus wires at 100 MHz [6].

Fat-tree networks are planar in theory, but in practice manufacturers such as Thinking Machines are not able to take advantage of this property. The CM-5's fat-tree network, for example, uses coaxial cables to connect nodes of the fat tree.

The interconnects most like Meerkat's are 2-D meshes, such as that of the Intel Touchstone Delta. 2-D meshes are planar and so like Meerkat, they can also use inexpensive wires and CMOS circuit technology to make the necessary connections between routing components. They do not require coaxial cable or exotic logic to achieve high bandwidth.

3.3 Buffering and deadlock in the interconnect

Meerkat has no buffering in the interconnect or in the node interfaces to the interconnect. Because of this, and because processors on the sending and receiving nodes rendezvous before data transmission, there is no need for hardware flow control. This allows nodes to inject data into the interconnect at high rates and is a principal source of the high internode bandwidth that Meerkat achieves.

The lack of buffering in the network also eliminates a source of potential deadlock that exists in other multicomputer interconnects that buffer data. Deadlock is avoided in these other interconnects, of course, but usually with some penalty in performance or complexity. There are no buffers in the Meerkat interconnect and so it is not possible to have a circular dependency involving buffers.

Meerkat could deadlock during bus arbitration if two nodes each acquired the first bus in a 2-bus connection, were not able to acquire the same second bus, and then do not release the first bus. This strategy of not releasing the first bus on the failure to get the second would also lower interconnect utilization by preventing other nodes from using the first bus for a period of time. For both of these reasons Meerkat's low level arbitration software releases the first bus in a failed 2-bus arbitration and tries again a short time later. The length of the back-off interval is random to prevent live-lock. This back-off scheme is akin to that used by Ethernet [10], except that it is done in software and the Meerkat bus arbitration circuit grants ownership of a particular bus to one node at a time; in Ethernet collisions occur when two nodes try to use the wire at the same time.

The lack of buffering in Meerkat also makes the fault model simpler, the interconnect implementation less expensive, the latency lower (by removing levels of logic between the sender and the receiver), inherently provides in-order delivery of packets, and allows the interconnect to be managed by software. The Meerkat interconnect does not require the design complexity of high speed routing circuitry present in 2-D meshes.

4 Performance

In this section we compare the performance of programs running on a large simulated Meerkat with that of the same programs running on the Intel Touchstone Delta. Like Meerkat, the Delta is a multicomputer composed of high performance microprocessors, local memory, and an interconnect that is used explicitly by application software. The two systems differ in their interconnects: Delta employs a conventional mesh of 2-D routers while Meerkat uses sets of vertical and horizontal internode buses.

4.1 Hardware-calibrated simulations of large systems

To evaluate the performance of Meerkat we use both a hardware prototype and a detailed system simulator. Our simulator models the semantics of instruction execution, virtual address translation, physical memory, internode interconnect, and I/O devices. This allows us to run programs on the simulator that run on the hardware and vice versa without modification or recompilation. To allow performance measurements to be made with the simulator we also model the timing of instruction execution, several of the processor pipelines, the data cache, TLB misses, internode signaling latencies and transfer rates, intranode bus contention, interference with DRAM refresh, and instruction cache cold misses. Both execution environments execute the program binary. The fine detail of simulation and close correspondence between the execution times reported by the same programs executing on both the simulated and real Meerkat gives us confidence that our simulation results for large Meerkats are valid. However, there are differences between Meerkat and Delta that complicate the comparison of the two interconnect architectures.

4.2 Differences between systems measured

The processors in the Delta are about twice as fast as Meerkat's; the Meerkat message passing code is written in carefully crafted assembler while the Delta runs the NX/M operating system, which is written in C; the Meerkat internode interface copies to and from memory whereas the Delta interface requires that the processor load and store each byte moved through the interface. In addition, on the Delta the test program runs in an address space separate from NX/M and thus incurs context switching costs whereas on Meerkat the test program runs in the same address space as the message passing library. Some of these differences, however, tend to cancel each other. The slower Meerkat processors executing our small message passing library tends to cancel the effect of faster Delta processors executing the larger NX/M operating system. It is our intention in this section to show that Meerkat is an effective architecture by showing that a Meerkat implementation can perform well in comparison to a large commercial multicomputer.

4.3 Message granularity

Interconnect bandwidth and latency are functions of message size, which itself is a function of the algorithm, data layout, the number of nodes applied to the problem, and the problem size. In general, if the number of nodes is increased while keeping the other parameters constant, the size of messages will decrease. Likewise, if the problem size increases the message size will often increase. A complete discussion of message size is beyond the scope of this paper, but we give a few examples to motivate our subsequent performance discussion.

1. The butterfly FFT algorithm using a cyclic layout generates messages that are $\frac{N}{P}$ points long where N is the number of points in the FFT, P is the number of nodes. A point takes 16 bytes in a complex FFT using double precision numbers. Thus, a 32,768 point FFT on 256 nodes will send 2048 byte messages.

2. We speculate that operating system traffic on Meerkat would be composed of both short control messages and page-sized messages to support file system and virtual memory traffic. Process migration would generate messages in excess of 64K bytes.

3. Blocked iterative solution methods send messages that are proportional to the size of one edge of the block. A red/black successive over relaxation (R/B SOR) algorithm on a 4096 by 4096 grid running on 256 nodes will use a block that is 16 by 16. If double precision numbers are used, each message will be 1024 bytes.

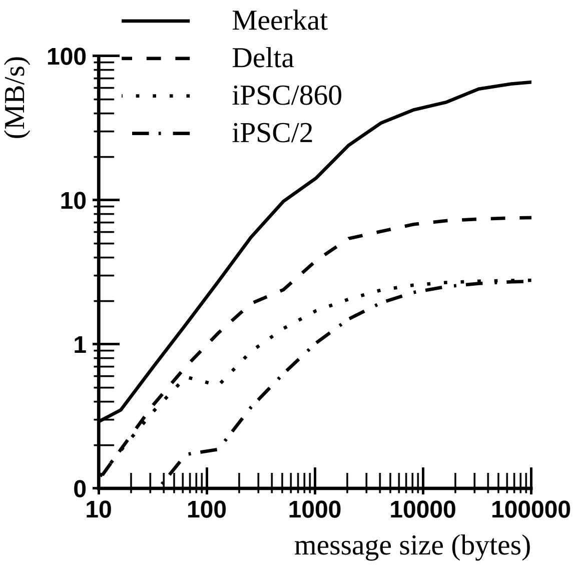

Figure 2: Throughput as a function of message size under light load.

4.4 Throughput under light load

Figure 2 shows the throughput achieved by a pair of nodes for both systems as a function of message size. We include in these graphs the throughput measured on the Intel Hypercube iPSC/2 [1] and iPSC/860. In this test the bandwidths reported by the Meerkat simulator and by the hardware differed by about one percent and so the Meerkat curve can be viewed both

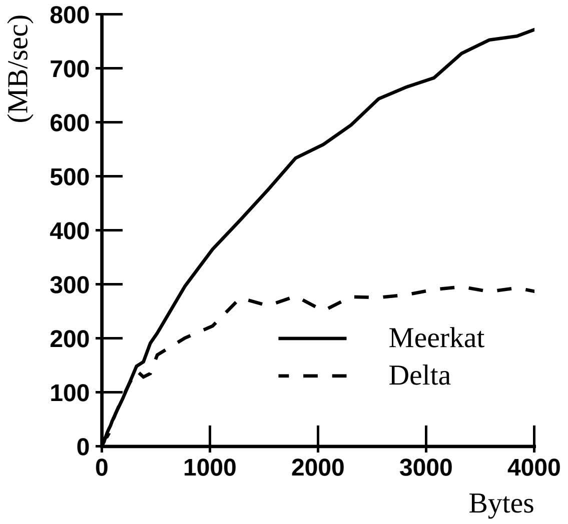

Figure 3: 256 Node system bisection throughput.

as measurements of a real system and as simulation results.

On small messages the throughput of both Meerkat and Delta is limited by the ability of the nodes to inject messages into the network. The lower performance of the Delta on small messages may be due to the extra work it is doing in NX/M that is not done in the Meerkat message passing library. Meerkat achieves 67 MB/sec for 100,000 byte messages whereas the Delta throughput levels off at about 8 MB/sec for 2,000 byte messages. On long messages both Meerkat and Delta are limited by their different abilities to drive their interconnects. Meerkat's internode bandwidth reaches 83 percent of its peak rate of 80 MB/sec, whereas the Delta reaches ten percent of its theoretical rate, which is also 80 MB/sec [12].

While Meerkat's maximum interconnect performance is seen at a message size that is longer than most applications will generate, its performance on shorter messages is still high. It may make sense, however, to reduce the per-message overhead by moving logic from low-level software into hardware. This would push the solid curve in figure 2 up and to the left, making the leftmost part steeper.

4.5 Throughput under heavy load

In this experiment there are two groups of 128 nodes each, A and B. Each node in group A sends a message to its partner in group B and then waits for a reply. Each node in group B first waits for a message from its partner in group A, then sends a message back to its partner. The nodes of each group are physically contiguous in a 8 by 16 block and the two groups are adjacent to form a 16 by 16 block of nodes. The distance between each node and its partner is eight. The total number of bytes moved between the groups is the product of the message size and the number of messages sent, which is 256. We calculate the throughput by dividing the total number of bytes moved by one half the round trip time.

Figure 3 shows both Meerkat and Delta with a nearly linear increase in throughput with increasing message size for messages of less than 500 bytes. As with the case of light interconnect load, this increase in throughput is due to the amortization of a fixed processor overhead per message over longer messages. The Delta throughput reaches a maximum of 280 MB/sec with a message size of 1000 bytes and Meerkat peaks at 750 MB/sec at a message size of 4000 bytes. Dividing each of these throughputs by the number of channels through which the data moves, in this case 16, we find that the Meerkat buses are are driven at an average of 46 MB/sec and the Delta channels at the midpoint are driven at 17 MB/sec.

This earlier plateau in Delta's throughput is due to the higher ratio of processor to interconnect performance in Delta than that of the same ratio in Meerkat. That is, Meerkat's slower processors need longer messages to be able to saturate its faster interconnect. However, the level of the plateaus depends principally on interconnect performance and not on processor speed.

4.6 Performance of 1-D FFT

Figure 4 shows the speedup achieved on a one dimensional FFT by both Meerkat and Delta as a function of the number of nodes applied to the problem. FFT is a common computational problem for large parallel systems that in practice often has poor speedup although it would be perfect if communication were free.

The total amount of computation is a function of the problem size and not the number of processors applied to the problem. With N points and P processors, there are $log_2 N$ steps each of which takes time proportional to N and of which the last $log_2 P$ are communication steps. Thus, as the number of processors increases, and the total running time becomes small, the effect of communication may dominate. Also, while the number of messages each node sends per step is constant, the message size is proportional to $O(\frac{1}{P})$. This means that when the number of processors dou-

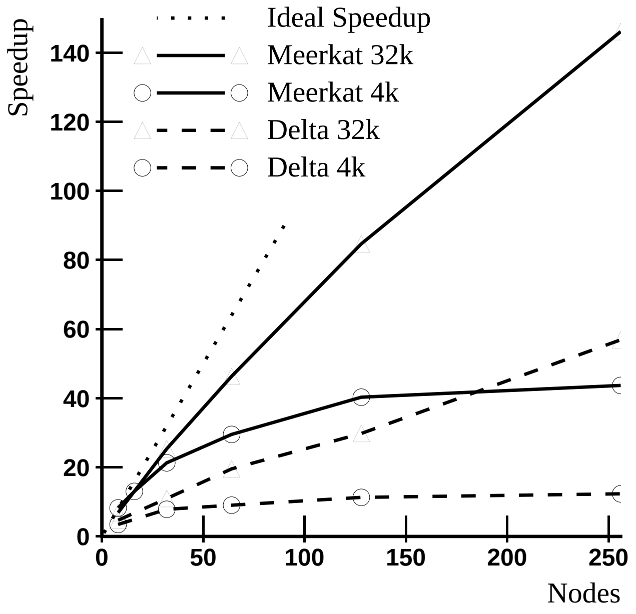

Figure 4: Speedup of 4k and 32k point FFT

bles, the amount of data per message is halved. Since the overhead of sending a message is fixed, the amount of overhead per byte sent doubles when P is doubled. In addition, another communication step must be performed when P is doubled.

Meerkat achieves a higher speedup on both problem sizes because its ratio of internode communication bandwidth to node performance is much higher than that of the Delta. The differences in these ratios is due to both the Delta's more powerful processors and to Meerkat's faster interconnect.

5 Hardware prototype

We have built a Meerkat prototype with four nodes arranged in a square. Each node has four Motorola 88100 processors, 16 KB each of instruction and data cache per processor, 32 MB of memory, two S-Bus slots for I/O adapters, interfaces to horizontal and vertical internode buses, an interrupt controller, a 32-bit cycle counter, and an interface to a 9-bit debug bus that connects all of the nodes with the host interface. Nodes plug together side-by-side to form horizontal buses and on top of each other to form vertical buses. The horizontal bus wires traverse the width of each node board to connect the the male and female horizontal bus connectors. These wires are tapped by the node's internode bus data switch. The vertical bus is carried by stacking connectors that span the 3 cm vertical spacing between nodes that are vertically adjacent. The system is clocked at 20 MHz, internode buses are four bytes wide, and internode data transfers move one four-byte word between the sending and receiving node per clock tick.

5.1 Interconnect cost in our prototype

Each node has components common to any system, such as processors, cache, and main memory, and those components used for internode communication. The main contributors to the cost of Meerkat's internode mechanism are the four 9-bit wide bus transceivers that form the internode bus data switch, four connectors per node that mate with the node's neighbors, and the board area that these parts require. The internode control logic requires a few hundred gates. In our implementation the logic is in an FPGA, but any commercial implementation would fold these gates into an existing gate array. Hence, the incremental cost of the Meerkat interconnect would be low.

It is common to compare gate counts, DRAM bits, SRAM bits, and silicon area in order to evaluate competing designs. However, given the simplicity of our approach, the cost of connectors and circuit board area are significant. The current cost for these items and the requisite integrated circuits is about $90 per node. Compared to the total cost of a node, which is $7,500, this is almost insignificant. This is on par with the cost of an Ethernet adapter while the bandwidth of Meerkat is 64 times higher and the processor overhead is lower than that of Ethernet. The overhead of the interconnect and distributed shared memory hardware in DASH [9] is about 20 percent of the system. While this is reasonable for a machine of its class, it represents a substantially higher system cost than Meerkat's two percent.

The design of our hardware prototype uses vintage 1990 integrated circuits and does not use special circuit tricks or any exotic technology. The detailed design, simulation, board layout, parts procurement, construction, debugging, host debugger development, and target monitor coding took two person years of effort. While it is difficult to quantify design complexity, and while a system designed for production requires more effort than that of an academic project, we feel that the small effort and high performance of our prototype is indicative of Meerkat's merit.

6 Meerkat Implemented with Current Technology

In this section we use our experience building a Meerkat prototype to see what a Meerkat implemented with 1993 technology would look like. The central issues in a modern Meerkat interconnect design are the limits of scaling, node size, circuit technology used for the interconnect, and clocking.

Bus length and the nature and the number of taps on a bus are critical in determining the rate at which data can be sent over the bus. This puts a severe limit on the size of a Meerkat. We anticipate that with current circuit technology a 16x16 system with the following characteristics can be engineered without much difficulty:

- internode bus lengths of 75 cm, stub lengths 5 cm or less
- a maximum of sixteen taps (each node having a tap) per internode bus
- internode buses clocked at 100 MHz with one 32-bit word transferred per clock tick
- use of GTL [6] logic levels for internode communication

These characteristics yield a system that can have 256 nodes and a raw per-bus internode transfer rate of 400 MB/sec. We estimate that under heavy load such a system would provide 20 MB/sec per node assuming that most communication requires two buses. If the maximum packet size is 1024 words, as in our prototype, the average latency under heavy load to send 1024 words from one node to another would be 100 microseconds. Under light load we estimate that a pair of nodes can achieve a sustained bandwidth of 330 MB/sec with a latency of 20 microseconds.

6.1 Node Size

Nodes need to be as small as possible in modern designs in order to keep the wires within the node as short as possible. The pressure to reduce intranode wire lengths is increasing because of higher clock rates and the concomitant use of logic technology with fast edge rates. For example, the current cluster of the DASH prototype is spread on five boards each about 1800 square cm. Planning for the next generation of DASH is for a cluster to be on a single 516 square cm board. The trend towards smaller nodes benefits Meerkat because it allows more nodes to tap a bus in a given length of internode bus. Circuit loading and cooling requirements due to an increased number of nodes will also limit the internode clock rate, but to a lesser degree.

6.2 Clocking

While asynchronous interconnects are popular, we see advantages in a synchronous design, namely that the hardware is simpler and it is much easier to debug. We believe that the difficulty in reproducing and isolating design errors and device failures in asynchronous designs is a serious problem. The principal disadvantage with synchronous designs is that clock skew must be controlled to achieve good performance.

Most synchronous backplane systems are limited by maximum clock skew, worst case device performance, round-trip propagation delay, and jitter. In these systems bus tenure is often just a few cycles and the bus standard was proposed before large and inexpensive standard cell CMOS gate arrays were available. We propose a clocking scheme that takes advantage of these inexpensive gate arrays and the relatively long bus tenures that we expect in Meerkat in order to compensate for clock skew, wire propagation delay, and device performance.

A system clock is distributed to each node in the system in a way that is convenient to implement and with a concomitant loose bound on skew of half a cycle. Each node is thus run at the same frequency, but nodes may be out of phase with each other by as much as half a clock cycle. The internode logic on each node contains a programmable delay chain for each bus tap that can shift the received system clock by as much as half a cycle. The bus protocol guarantees that on every cycle all nodes know which node is the current bus master. The internode logic uses this information to adjust its delay chains to compensate for clock skew, propagation delay, and device performance. It does this by looking up the delay value in a skew table that is indexed by node and tap number. In a system where B is 16, the table will be two by 16. Each element of the table is a delay value which will be four to six bits wide.

The skew table is loaded in a calibration procedure that is performed when the system is first started and could be updated periodically to adjust for temperature variations. When ownership shifts from one node to another there will be a loss of at least one cycle as each receiver adjusts its delay chain. Since we expect bus tenure to last for tens or hundreds of cycles, this is a small price to pay for the high clock rate that we hope to achieve. By this dynamic adjustment we

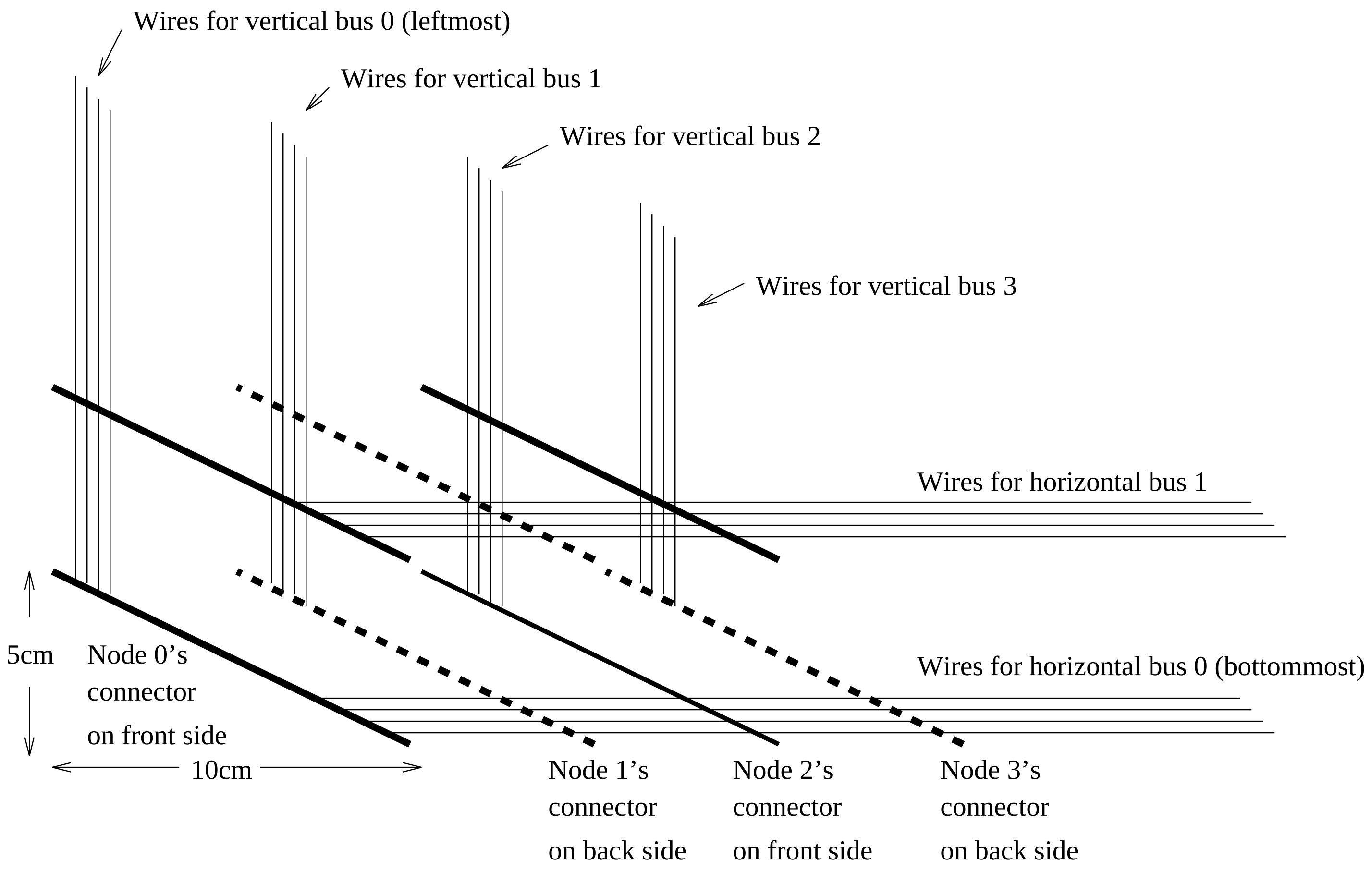
Wires for vertical bus 0 (leftmost)
Wires for vertical bus 1
Wires for vertical bus 2
Wires for vertical bus 3
Wires for horizontal bus 1
Wires for horizontal bus 0 (bottommost)
5cm
Node 0's connector on front side
10cm
Node 1's connector on back side
Node 2's connector on front side
Node 3's connector on back side

Figure 5: Lower Left Corner of a Meerkat Backplane.

compensate for many factors that limit performance in other systems.

We believe that clock skew will not be the limiting effect in a high performance Meerkat design. Rather, clock jitter and bus reflections from the taps that each node places on the internode buses will the primary constraints. The later effect can be helped by keeping stub lengths as short as possible, using a controlled-impedance backplane, and using GTL.

Any pair of horizontal and vertical buses can be connected through cross point switches. This requires all buses be clocked at the rate of the slowest bus. In an optimal design all of the buses would reach their limit at the same clock frequency. To do this we chose a physical arrangement that makes the wires that constitute the horizontal and vertical buses the same length.

Figure 5 shows how a Meerkat backplane would be laid out in order to maximize the number of nodes in two dimensions while keeping the horizontal and vertical bus lengths equal and as short as possible. This figure shows just a portion of a Meerkat backplane in the lower left corner. Connectors on the front of the backplane are shown as solid, and connectors on the back are shown as dashed. Node boards have connectors along one edge that mate with the backplane connectors. Nodes boards are plugged into the backplane connectors and thus are at right angles with the backplane circuit board. Having node connectors, and thus nodes, on both sides of the backplane doubles the number of nodes that can be attached over the number which can be attached with a single-sided backplane. Node connectors and thus the nodes themselves are set at a 30 degree angle with respect to the bottom edge of the backplane circuit board. The backplane is a multiple layer circuit board with wires for the internode buses running straight, i.e, not having any bends.

Figure 5 shows nodes spaced every 5 cm on horizontal and vertical buses. For a given number of nodes per bus, B, there are $(B-1)$ 5 cm bus segments. Thus, a system that supported up to 16 nodes per internode bus at 5 cm intervals has buses whose wires are 75 cm long. A system with 16 nodes per bus, 16 vertical, and 16 horizontal buses has 256 nodes. A backplane that is 75 cm on a side would probably have to be fabricated in four 37.5 by 37.5 square cm segments. The connections between the backplane segments would be designed to match the backplane's target impedance as closely as possible.

7 Related Work

The diagram for Meerkat resembles those of the Wisconsin Multicube [5] and Aquarius Multi-multi [4]. However, the latter two machines put their buses to different uses than does Meerkat. The Multicube and Aquarius implement a coherent shared memory system in hardware. Therefore, transactions on their buses are initiated by memory reference instructions and cache coherence operations. Packets on these buses are small, usually carrying a cache line or less of data. In contrast, Meerkat's buses are manipulated by low-level message passing code that is invoked explicitly by user programs to send packets that often will be hundreds or thousands of bytes in length.

The Intel Paragon and the Intel Delta are distributed-memory multicomputers that consist of powerful nodes connected by a 2-D mesh. The Paragon adds a second microprocessor between the main node processor and the interconnect. Thus, the Paragon is able to delegate to the dedicated processor some low level functions that on the Delta or Meerkat must be done in software on the main processor.

On both the Delta and the Paragon, all routing of messages is done in the interconnect hardware. In Meerkat the low level message passing code establishes and manages node-node connections explicitly. Software control of message routing is possible in Meerkat because the interconnect requires only a small number of routing decisions per connection. The three systems therefore represent different points in the spectrum of hardware/software tradeoffs. Meerkat is the most software-intensive, the Paragon is the most hardware-intensive, and the Delta is in between.

8 Conclusion

Meerkat is a vehicle for exploring and evaluating several closely-related ideas about the design of multicomputer interconnects. These ideas concern the benefits of planar interconnects, the elimination of levels of logic between communicating nodes, the elimination of buffering in the interconnect, and software implementation of several multicomputer interconnect functions. A number of academic and commercial multicomputers use hardware for functions that Meerkat does with software. These functions include data flow-control, interconnect switch control, error handling, and contention management.

Several of the ideas demonstrated by Meerkat could be used in multiprocessors with hardware-coherent shared memory systems. For example, the two dimensional grid of passive buses could lower communication latency to the point where time-of-flight dominates the internode communication latency. This would provide much lower latency than that of the commonly used grid of mesh routers. This, in turn, would lessen the need for latency hiding techniques and increase the single-thread performance of multiprocessors. Thus, some of the lessons that we have learned could be applied to parallel systems very different than ours.

A methodological contribution of our paper is the combined use of a hardware prototype and a detailed simulator. We used performance results from the prototype to calibrate the simulator. In addition, the prototype allows rapid execution of programs that use up to four nodes. The simulator extends our performance results to systems with up to 256 nodes. While the simulator is slow compared to the hardware, its on-the-fly generation of threaded code allows it to simulate roughly a million processor cycles per second on a modern workstation [2]. The Meerkat simulator allows us to do detailed simulations of Meerkat systems with 256 nodes running significant parallel programs.

Meerkat's simplicity allows implementations that have high performance without great design effort. The Meerkat hardware prototype supports this result. With modest effort, we built a multicomputer that yields interconnect latencies and bandwidths above that achieved by commercial parallel system manufactures who spend several orders of magnitude more effort. Of course, there is a big difference between the design of a commercial system and an academic project, but the large difference in effort and Meerkat's excellent performance nonetheless is indicative of the success of the Meerkat approach. We have shown the benefits of taking this hardware-minimalist performance-oriented approach to multicomputer design.

9 Acknowledgements

We thank the corporations that directly supported the Meerkat project: Data I/O, Horizon Research, Latice Semiconductor, Motorola, Tektronix, and Xilinx, as well as the people who helped us: Craig Anderson, Jeff Beachy, Linda Bender, Kevin Bolding, Jim Canon, Aaron Eastlund, Carl Ebeling, Ed Felten, George Forman, Melanie Fulgham, Scott Gardner, Dennis Gentry, Jerry Gipper, Gary Horbuckle, David Keppel, Alex Klaiber, Ed Lazowska, Hank Levy, Rik Littlefield, Cathy McCann, Dylan McNamee, Joao Setubal, Robert Short, Arun Somani, Larry Snyder, Chandu Thekkath, and John Theus. We also thank the anonymous reviewers for their helpful comments.

Our work is also supported in part by the National Science Foundation (Grants No. CCR-8907666, CDA-9123308, and CCR-9200832), the Washington Technology Center, Digital Equipment Corporation, Boeing Computer Services, Intel Corporation, Hewlett-Packard Corporation, and Apple Computer.

This research was performed in part using the Intel Touchstone Delta System operated by Caltech on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by The Pacific Northwest Laboratory of the US Department of Energy.

References

[1] Ramune Arlauskas. *iPSC/2 System: A Second Generation Hypercube*, January 1988.

[2] Robert Bedichek. Some efficient architecture simulation techniques. In *Proceedings of the Winter 1990 USENIX Conference*, pages 53–63, January 1990.

[3] Robert Bedichek and Curtis Brown. The Meerkat multicomputer. *University of Washington CSE Technical Report 92-09-05*, 1992.

[4] Michael Carlton and Alvin Despain. Aquarius project. *IEEE Computer*, pages 80–83, June 1990.

[5] James R. Goodman and Philip J. Woest. The Wisconsin multicube: A new large-scale cache-coherent multiprocessor. In *Proceedings 17th Annual Symposium on Computer Architecture*, pages 422–431, May 1990.

[6] Bill Gunning, Leo Yuan, Trung Nguyen, and Tony Wong. A CMOS low-voltage-swing transmission-line transceiver. *IEEE International Solid-State Circuits Conference*, pages 58–59, 1992.

[7] Intel Corp, 2065 Bowers Avenue, Santa Clara, California 95051. *Touchstone Delta System User's Guide*, 1991.

[8] Intel Supercomputer Systems Division. *Paragon XP/S Product Overview*, 1991.

[9] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. In *Proc. 17th Annual International Symposium on Computer Architecture*, pages 148–159, May 1990.

[10] R.M. Metcalfe and D.R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[11] Youcef Saad and Martin H. Schultz. Topological Properties of Hypercube. In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, vol. 1*, pages 867–872, January 1988.

[12] Charles L. Seitz and Wen-King Su. A family of routing and communication chips based on the mossaic. *Research on Integrated Systems*, pages 320–337, 1993.

[13] Thinking Machines Corp., 245 First St., Cambridge MA 02142. *CM-5 Technical Summary.*