# The RACE Network Architecture

Bradley C. Kuszmaul*
bradley@mc.com
Mercury Computer Systems, Inc.
199 Riverneck Road
Chelmsford, MA 01824

## Abstract

*The RACE® parallel computer system provides a high-performance parallel interconnection network at low cost. This paper describes the architecture and implementation of the RACE system, a parallel computer for embedded applications.*

*The topology of the network, which is constructed with 6-port switches, can be specified by the customer, and is typically a fat-tree, a Clos network, or a mesh. The network employs a preemptable circuit switched strategy. The network and the processor-network interface work together to provide high performance: 160 megabytes per second transfer rates with about 1 microsecond of latency. Priorities can be used to guarantee tight real-time constraints of a few microseconds through a congested network. A self-regulating circuit adjusts the impedence and output delay of the pin-driver pads.*

## 1 Introduction

Parallel computers often force a choice between low performance and high cost. High performance usually implies high price, and low price usually implies low performance. The RACE[1] embedded parallel computer system, however, provides high performance at low cost. Most multipurpose massively parallel computers cost an order of magnitude more than RACE, while most embedded-systems parallel computers provide an order of magnitude less performance. This paper shows how the RACE network architecture helps the RACE system by providing low cost, high-performance interconnect. (An overview of the RACE software architecture can be found in [3].)

The RACE system is designed for embedded applications, such as medical image processing and military signal processing. The key to success in embedded applications is keeping down the cost of the hardware, providing good performance, and minimizing power consumption. Embedded systems do not need to run timesharing operating systems, and embedded programs are relatively simple and infrequently modified.

For embedded systems, only one application is running at a time, and thus an embedded system does not require support for timesharing. There are several ways to support timesharing, most of which are either expensive or slow. Timesharing can be supported by using expensive operating system calls to protect the network from broken or malicious user-level code. Similar protection can be achieved by adding hardware to the processor-network interface. Such protection can cost a lot of hardware or time. The RACE system does not provide timesharing support, and the cost is kept down.

Embedded applications are typically simpler and easier to maintain than the applications designed for multipurpose parallel computers. A particular application is typically written only once, and then many systems are installed running the same application. Because the RACE system was designed as a production machine for today's applications, rather than a research vehicle for understanding tomorrow's applications, the RACE system's software environment can be kept much more focused, which keeps the system cost down. The RACE system does not require a large investment in compilers, debuggers, or operating systems software.

The RACE network provides high performance. In an otherwise unloaded network, the latency from application-space to application-space is approximately 1 microsecond. The point-to-point bandwidth is 160 megabytes per second. Both small systems of less than 32 processors and very large systems of about 700 processors have been built. This performance is an order of magnitude better than is found in most other scalable parallel systems, whether they are embedded systems or multipurpose parallel computers, and the cost is an order of magnitude less.

The RACE system currently uses i860 microprocessors, which achieve high performance at low cost and power. Mercury has announced that in the future it will introduce products based on the PowerPC microprocessor and SHARC signal processor, continuing the strategy of using processors with high performance, low cost, and low power per floating point operation per second.

The rest of this paper is organized as follows: Section 2

---

*Bradley C. Kuszmaul is a postdoctoral fellow at the Massachusetts Institute of Technology Laboratory for Computer Science, and consults for Mercury Computer Systems, Inc.

[1] RACE is a registered trademark of Mercury Computer Systems, Inc.

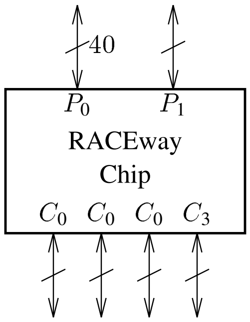


Figure 1: The RACE network chip has 6 bidirectional channels. Two of the channels connect to the parents, and four connect to children. (The parent-child naming reflects the fat-tree topology in which the network chips are typically connected.)

describes the RACE network. Section 3 describes some of the chip-level design strategies used to solve system-level problems. The RACE processor-network interface is described in Section 4. Section 5 concludes with an analysis of how well the RACE network meets its design goals.

## 2 The RACE network

The RACE network chip is the key to the high performance and low cost of the RACE system. The network chip is a crossbar with 6 input/output channels, each of which includes a 32 bit datapath, and 8 bits of control and clocking. (See Figure 1.) The chip runs at 40 Megahertz and is implemented in the LSI 500K CMOS logic family.

Routing through the network is performed by *source-path* routing. The message, before it leaves its source, is given a path to follow through the network. As the message moves through a network of chips, each chip takes subsequent instructions. At each chip, the message specifies an output port to which the message wants to go. In Figure 1, the output ports are named $C_0$, $C_1$, $C_2$, $C_3$, $P_0$, and $P_1$. (In the VITA RACE network specification [9], ports $C_0$, ..., $C3$, $P_0$, $P_1$ are refered to as $A$, $B$, $C$, $D$, $E$, and $F$, respectively.) The chip also supports the additional possibility of specifing "UP", which allows the message to be routed either to Port $P_0$ or to $P_1$. This adaptive routing option is specifically designed to support fat-tree and Clos networks, the most common configurations of RACE network chips. Adaptive routing improves performance in such networks by allowing the hardware to choose among several alternative paths. Messages are less likely to become blocked when they have several alternatives.

The RACE network can be configured into a wide variety of network topologies. The most common configuration is a fat-tree (shown in Figure 2). For the fat-tree network, we think of each RACE network chip as having two parent ports and four child ports. The processors are at the leaves of the fat tree. To route a message from one processor to another, the message goes up the tree, choosing either parent as it goes, until it reaches a chip that is a common ancestor of the source and destination. Then the message travels down the tree along the unique path from that common ancestor to the processor. An example route is shown, along with the path specification from one processor to another.

Leiseron's original description of fat-trees describes a system with monolithic switches [5]. The RACE interconnect topology is identical to the fat-tree with constant size switches described by Greenberg and Leiserson [2]. Several other vendors of higher-priced machines employ the fat-tree topology: The Connection Machine CM-5 uses a fat-tree topology with a packet-switched router [6], and the Meiko CS-2 uses a circuit-switched fat-tree network [7].

The RACE network also supports a one-to-many broadcast. At each hop along the path, the path specifier can say "ALL-CHILDREN", which causes a copy of the message to be routed to all the children of the chip. (If a broadcast message arrives from a child, the message is not sent back to that child.)

The RACE network's fat-tree topology has very good scaling properties. In a $P$ processor machine, the network diameter is $2 \cdot \lceil \log_4 P \rceil$ hops. The bandwidth across any bisection of a machine with $P = 4^i$ processors is $160\sqrt{P}$ megabytes per second in one direction.

Each channel is bidirectional, and is driven in only one direction at a time at 160 megabytes per second. Why bidirectional channels? The alternative is to build two dedicated one-directional channels, each with half the wires, that run at 80 megabytes/second each. The bidirectional appraoch was chosen because when the network is lightly loaded, the bandwidth available for any particular message is doubled. When the network is heavily loaded the total bandwidth is all that matters. Furthermore, the i860 bus can support burst transfer rates of 160 megabytes/second, but only in one direction at a time.

The RACE network is circuit switched. To send a message, a path is established through the network, and then the data of the message are sent. To establish a path through the network, a message header specifying the path is sent into the network. The header makes as much progress as it can, waiting at each chip until it can make progress. Each bidirectional channel is either *occupied* or *free*. When a free channel is available that matches the path specification, the channel becomes occupied, and the message header travels along that channel. If all the channels that match the path specification are ocupied, the message header waits. After the path is established, an acknowledgment is sent back to the source. The source then starts sending data down the path in a pipelined fashion. When the last byte of data is sent down the pipeline, the channels each become free, closing up the path behind the message.

Each message header also contains a *priority*, numbered

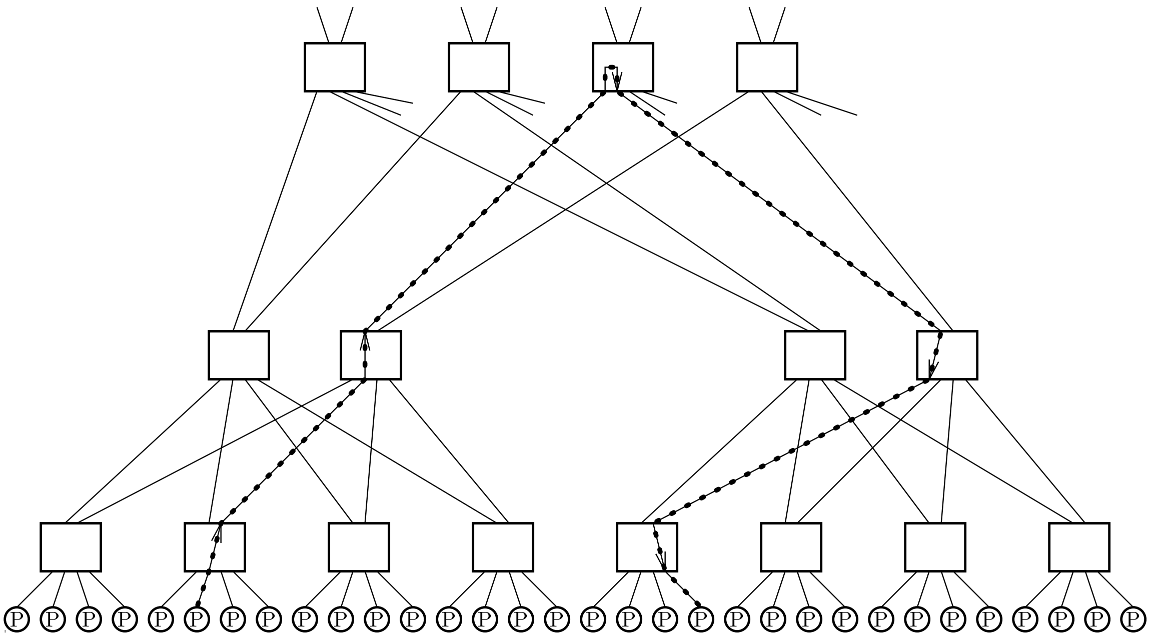


Figure 2: The standard RACE network interconnect topology is a fat-tree. Each chip has two parents and four children. An example route from one processor to another is shown with the dotted path. In the example, the route the message takes is specified by $(\mathrm{UP}, \mathrm{UP}, C_1, C_0, C_3)$, indicating that the message should travel up two levels, then take child 1, then child 0, then child 3.

from 0 to 3. If a high priority message arrives at a chip and all of the channels it needs are occupied, and if one of the channels is occupied by a message of lower priority, the lower priority message is forced to release the channel. This suspension of the lower priority message is accomplished by sending a "kill" signal backwards along the path all the way to the source. The source is required to relinquish the path. Any data that was already in the path propagates down the pipeline to its destination. The channels of the path become free, and the higher priority message has an opportunity to make progress. The mechanisms to handle the suspension of a message and reestablishment of a channel to continue sending the message are built into the processor-network interface hardware. Messages are suspended and then restarted without any intervention by the software running on the processing node.

The incoming port number is used as a tie-breaker between incoming messages with the same priority. All else being equal, messages coming from the parents have higher priority than message coming from children, and messages coming from higher numbered parent (or child) ports have higher priority than messages coming from lower numbered ports. This tie-breaker priority does not result in the suspension of any message that has already been routed to the next switch, however.

The priority scheme can be used to guarantee that an application will meet real-time constraints. In particular, one can bound the time it takes for a high priority message to get through any RACE network. For example, in a 64-processor RACE system built using the fat-tree topology, it takes no more than 2.58 microseconds for a high priority message to get through if all the other messages are low priority. To understand how the RACE system guarantees this bound, first observe that the height of a fat tree contaning $P$ processors is

$$l = \lceil \log_4 P \rceil . \tag{1}$$

The maximum distance a message must travel, in hops, is

$$D = 2l - 1. \tag{2}$$

It takes 6 clock cycles of startup time, plus 5 clock cycles per hop for the first word of a message to travel through the network if the message hits no contention. Thus the number of clock cycles for a message to get through an uncongested network is

$$\begin{aligned} T_{\text{no contention}} &= 5D + 6 \\ &= 10l + 1. \end{aligned} \tag{3}$$

If there is contention, then the high priority message must wait for the "kill" signal to propagate back to the source, and for the channel to close up behind the killed message. When the message is killed at a point where it is $h$ hops from its source, the number of clock cycles the killing message must wait is

$$T_{\text{kill}(h)} = 2h + 6. \tag{4}$$

Thus the worst case time for a high priority message to get to its destination is the sum along the longest path in a fat-tree of the worst case times to kill any messages that are using channels needed by the high priority message. For a fat tree of height $l$, the worst-case number of cycles spent waiting for kills is

$$\begin{aligned} T_{\text{total kills}(i)} &= 2 \cdot \left( \sum_{i=l+1}^{2l-1} (6 + 2i) \right) + 6 + 2l \\ &= 6l^2 + 8l - 6 \end{aligned} \tag{5}$$

and the worst case time to deliver a high priority message, measured in clock cycles, is

$$\begin{aligned} T_{\text{worst}} &= T_{\text{no contention}} + T_{\text{total kills}(l)} \\ &= 6l^2 + 18l - 5. \end{aligned} \tag{6}$$

For example in a 64 processor fat-tree we have $l = 3$ and

$$\begin{aligned} T_{\text{worst}(64)} &= 54 + 54 - 5 \\ &= 103, \end{aligned}$$

which, at 40 megahertz, is 2.58 microseconds. Thus, one of the big advantages of the preemptable circuit switched approach used in the RACE system is that it can meet very tight real-time constraints.

It is also possible to interconnect RACE network chips into other topologies such as grids. In most such networks, the advantages of adaptive routing and the ability to broadcast are usually lost, but the system's flexibility allows each customer to get exactly the network needed to solve their problem. Since most customers have a particular fixed problem to solve it is typically much easier to design a network to solve the problem, than it is to try to force the problem to run well on a particular fixed network. Customers with real-time constraints can use the priority scheme to help meet those constraints, and sometimes the network topology can also be chosen to help meet such constraints.

The RACE system was designed from the beginning to fit into a standard VME bus [1]. The RACE system uses the unused pins on the VME connector, and the network can operate concurently with normal VME bus operations. This concurrent operation is difficult to achieve, since both the VME signals and the RACE signals each could potentially suffer from crosstalk from the other. If the RACE network signals ran too fast or had too-sharp edges (with high frequency components) then the standard VME signals would suffer from crosstalk, which would require that the VME bus be quiesced before the network could be operated. If the network signals used too low a voltage swing, then the network signals would suffer from crosstalk induced by the VME signals, again requiring the quiescence of the VME bus. The RACE system's 160 megabyte/second channels push the standard VME connectors as fast as possible without requiring that the VME bus be quiesced.

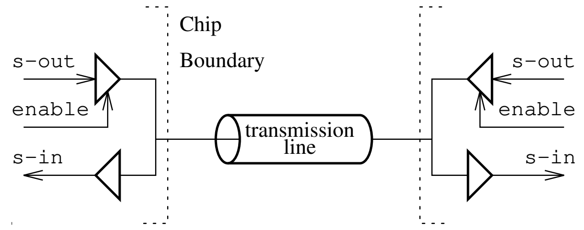


Figure 3: An ideal pad driver for a bidirectional transmission line drives the transmission line with the value of `signal-out` if the `drive-out` is asserted. The transmission line is sensed, and its current value is used to determine the value of `signal-in`. A higher-level protocol guarantees that only one end of the transmission line is being driven at any time. Ideally, when the driver is enabled, the impedence presented to its output should match the transmission line's impedence. When disabled, the driver should present no impedence to its output.

## 3 Coping with CMOS

Having described the architecture of the RACE network, let us examine how careful engineering at the chip level can reduce the cost and improve the reliability of the system. Building a reliable cost-effective system requires more than architecting a system and building a chip. It requires making sure that the chips can be connected together reliably and without any extra 'glue' logic. In this section, we show how careful design at the chip level can avoid even passive 'glue', such as termination resistors, and can make the clocking more robust.

The main problem with CMOS chip technology, from a systems perspective, is that the speed of CMOS logic varies widely (varying with it, the impedence of CMOS drivers). This makes it difficult to build resistors in CMOS, which means that it is difficult to interface to transmission lines. It also means that it is difficult to control clock skew and therefore to guarantee that hold times will be met. On the other hand, CMOS is inexpensive, and it pays to design a system that uses CMOS.

Figure 3 shows the setup for an ideal bidirectional transmission line. The key design problem is to match the output impedence of the drivers to that of the transmission line, so that signal reflections do not introduce noise. The internal impedence of CMOS drivers can vary by more than a factor of four, both because different chips have different characteristics, and because CMOS performance is sensitive to

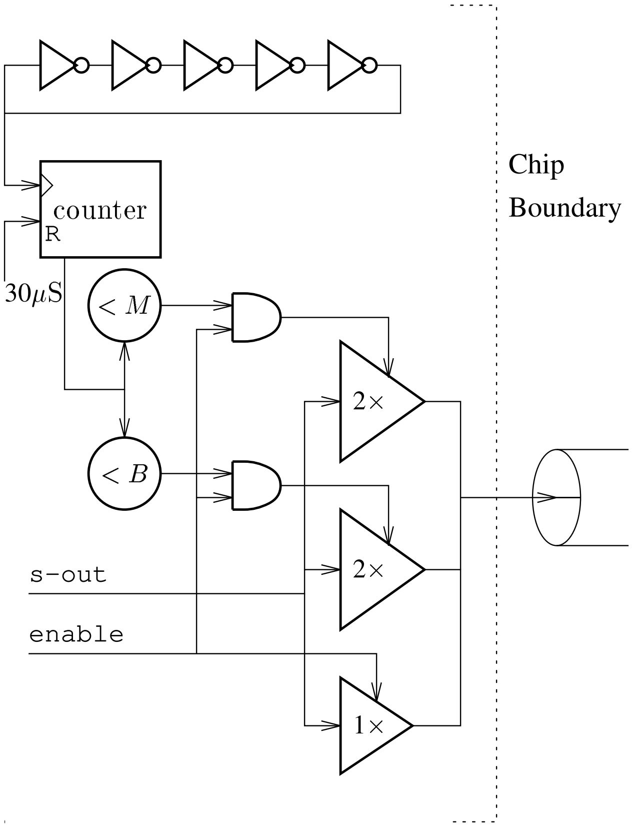


Figure 4: Three output drivers, of different sizes, are enabled to control the pad-driver output impedence. A counter, latched and reset every 30 microseconds deterimines the speed of a ring oscillator. The latched value of the counter is compared to determine which output drivers should be enabled. The `enable` signal can disable all the output drivers.

temperature, and so it is difficult to design a CMOS driver to match a particular impedence. The typical choice is to use another driver technology (such as GaAs or bipolar ECL) or to use external termination resistors. Either choice implies high cost, high power consumption, and reduced reliability.

The RACE network chip avoids off-chip termination resistors by dynamically adjusting the impedence of the transmission line drivers. The RACE approach is to vary the strength of the driver to vary the impedence. A faster driver has a lower impedence than does a slower driver. To accomplish this, each driver actually consists of three drivers wired in parallel. By choosing whether one, two, or three drivers are enabled, the impedence can be adjusted. Ideally, the driver is tuned so that it drives the transmission line to half its target voltage, and then the reflection raises the voltage to the full target voltage.

To control how many drivers are enabled, the RACE network chip uses a ring oscillator connected to a counter. (See Figure 4.) Every 30 microseconds the counter is sampled and compared to determine if it is *small*, *medium*, or *large.* If the counter value is small, all three drivers are enabled. If the counter is medium, two of the drivers are enabled. If the counter is large, then only one driver is enabled. Thus the speed of the oscillator controls the output impedence of the pad driver. When the oscillator runs faster, it indicates that the actual impedence of any transistor on the same chip is lower. Thus, the RACE network chip supports inexpensive matched impedence bidirectional transmission lines.

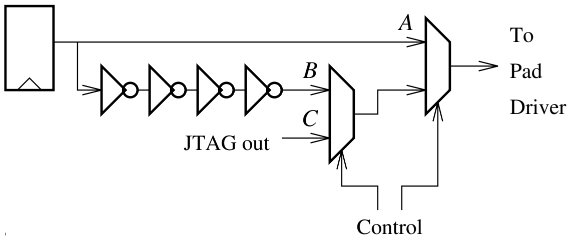


Figure 5: A circuit to help guarantee hold time on the next chip. If the oscillator-counter-comparitor control circuit determines that this chip is currently slow, then the output is selected from Source $A$. If the chip is currently fast then the output is selected from Source $B$, delaying it a small amount. During diagnostics the output is selected from Source $C$.

Knight and Krymm [4] describe a more sophisticated, and more complex, self-terminating pad driver. In their scheme, the internal impedence of the driver is controlled by adjusting the bias voltage on the transistors with a charge-pumping circuit. In practice, the relatively simple scheme used in the RACE network chip does a good job, even as the system is subjected to rapid heating and cooling.

The oscillator-counter-control circuit is also used to ensure that the system meets its hold-time constraints. A common problem in edge-clocked systems is that clock skew can violate the hold time constraint of a latch. To solve this, the RACE network chip uses the same oscillator-counter-comparitor circuit to decide whether to delay each signal leaving the chip. Figure 5 shows how this works. If the chip is running slowly, then the output is routed straight through the multiplexor (mux) that would be needed for an ordinary JTAG interface. If the chip is running quickly, then the output is delayed by a few inverters, and an extra mux. The JTAG path is routed through two muxes.

Another approach that guarantees that a circuit meets its timing constraints uses level-sensitive clocking rather than edge-sensitive clocking. With level-sensitive clocking one can independently control the setup and hold times. The RACE system needed a scheme that would work with off-the-shelf chips that use edge-sensitive clocks, however.

## 4 The RACE processor-network interface

In many systems, the interface between the processor and the network introduces performance bottlenecks or high cost. In the RACE system, the processor-network interface provides high bandwidth, low latency access to the network, but because the interface is kept simple, the interface can be kept inexpensive.

The processor-network interface provides each processor with the abstraction that it can operate on any other processor's memory. The operations can take two forms:

- Direct load and store operations on remote memory provide low latency access to any memory address in the entire machine.
- Block-transfer, implemented by a direct-memory-access (DMA) engine in the network interface, provides high bandwidth with only about 1 microsecond of additional latency.

The direct load and store operations are typically used to access remote I/O devices such as SCSI drives, while the block transfer is typically used to communicate between processors within a parallel application. Because the RACE system's global memory is uncached, the system provides a memory model with strong consistency. (For a survey of memory consistency models, see [8].)

## 5 Conclusions

The RACE computer system can be configured as many different kinds of networks. For example, we have implemented some systems that used meshes. This topology indepedence allows a customer, who typically has a particular fixed problem to solve, to fit the topology to the problem, instead of trying to fit the problem to the topology.

Using a point-to-point interconnect strategy for RACE, instead of a bus, helped to solve many system-level problems. Busses are much more difficult to engineer to run fast without introducing noise problems (through crosstalk and signal reflections.) By using a point-to-point interconnect strategy, the system can be set up as matched-impedence transmission lines with very low cost termination. By being careful to avoid sharp rising edges in the signal drivers, we avoided high-frequency waves that could cause crosstalk with the VME bus. By using TTL-level signalling instead of a low voltage signalling technology such as Gunning Transistor Logic (GTL), the RACE signals gained noise immunity from the standard VME bus signals. Thus, the VME bus is not required to quiesce before the RACE network is used, and the system can use the VME bus and the RACE network concurrently.

A network approach provides more interconnect as the number of processors increases, where a bus provides the same amount of bandwidth for a few processors as it does for many processors. What is enough bandwidth to keep a few processors busy is typically not enough to keep 16 or 32 processors busy.

The success of the RACE network at meeting its design goals can be traced to the use of readily available technology. By avoiding risky high-performance technologies the RACE system was implemented quickly. The RACE system was designed to fit into a standard environment, operating over the unused pins of the VME bus without causing crosstalk or performance degradation, and interfacing to inexpensive technology such as field programmable logic devices. An active backplane is plugged into the back side of the VME bus in order to provide the RACE network interconnectivity, allowing standard VME boards to be plugged into the same bus. It takes less circuitry to implement an interface to the RACE network than it takes to implement an interface to the VME bus, and the data transfer rate is much higher than for the standard VME bus.

## 6 Acknowledgments

Bob Blau, Bob Frisch, Barry Isenstein, and Craig Lund explained the RACE network architecture to the author, and corrected many errors in the manuscript. Brian Bouzas and Leigh McLeod also provided many helpful suggestions. Any errors remaining in this paper are the responsibility of the author.

[1] VMEbus specification. ANSI/IEEE STD 1014-1987.

[2] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. *Advances in Computing Research,* **5**, pages 345–374, 1989.

[3] Barry S. Isenstein and Bradley C. Kuszmaul. Overview of the race hardware and software architecture. In *First Annual Conference of the Rapid Prototyping of Application Specific Signal Processors (RASSP '94)*. IEEE Press, August 1994.

[4] Thomas F. Knight, Jr. and Alexander Krymm. A self-terminating low-voltage swing CMOS output driver. *IEEE Journal of Solid-State Circuits,* **23** (2), pages 457–464, April 1988.

[5] Charles E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers,* **C-34** (10), pages 892–901, October 1985.

[6] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St. Pierre, David S. Wells, Monica C. Wong, Shaw-Wen Yang, and Robert Zak. The network architecture of the Connection Machine CM-5. In *Symposium on Parallel and Distributed Algorithms '92,* pages 272–285, San Diego, California, June 1992.

[7] *CS-2 Product Description.* Meiko, 1992.

[8] D. Mosberger. Memory consistency models. *ACM Operating Systems Review,* **27** (1), pages 18–26, January 1993.

[9] *RACEway Interlink — Data Link and Physical Layers.* VITA Standards Organization (VSO), May 1994.