On Characterizing Performance of the Cell Broadband Engine Element Interconnect Bus

Thomas William Ainsworth and Timothy Mark Pinkston
Electrical Engineering Department
University of Southern California
Los Angeles, CA 90089-2562, USA
E-mail: tainsworth@alumni.nd.edu, tpink@usc.edu

***Abstract* – With the rise of multicore computing, the design of on-chip networks (or networks on chip) has become an increasingly important component of computer architecture. The Cell Broadband Engine's Element Interconnect Bus (EIB), with its four data rings and shared command bus for end-to-end control, supports twelve nodes—more than most mainstream on-chip networks, which makes it an interesting case study. As a first step toward understanding the design and performance of on-chip networks implemented within the context of a commercial multicore chip, this paper *analytically* evaluates the EIB network using conventional latency and throughput characterization methods as well as using a recently proposed 5-tuple latency characterization model for on-chip networks. These are used to identify the end-to-end control component of the EIB (i.e., the shared command bus) as being the main bottleneck to achieving minimal, single-cycle latency and maximal 307.2 GB/sec raw effective bandwidth provided natively by the EIB. This can be exacerbated by poorly designed Cell software, which can have significant impact on the utilization of the EIB. The main findings from this study are that the end-to-end control of the EIB which is influenced by the software running on the Cell has inherent scaling problems and serves as the main limiter to overall network performance. Thus, end-to-end effects must not be overlooked when designing efficient networks on chip.**

Index terms: Cell Broadband Engine, Element Interconnect Bus, interconnection networks, on-chip network, network on chip, heterogeneous multicore, network characterization, performance bottleneck.

I. Introduction

With the rise of multicore computing, the design of on-chip interconnection networks has become an *increasingly* important component of computer architecture. On-chip networks (OCNs)—also known as networks on chip (NOC)—interconnect various functional and computational resources on a chip. These days, researchers are actively pursuing the design of efficient OCNs after becoming aware, in recent years, that wires are now a dominant factor in determining system performance, cost, power dissipation, reliability, and other important characteristics [1] – [5]. On-chip network architectures are being proposed that no longer are flat shared bus structures [6], [7], [8] but, instead, are point-to-point and switched networks taking the form of hierarchical buses, rings, meshes, crossbars, and even irregular topologies [9] – [15]. As summarized in [16], on-chip networks are being proposed and finding use in research and commercial processor chips whose microarchitecture are partitioned into multiple (and soon to be many) clusters [17] – [20], tiles [21] – [24], and cores [25] – [31].

On-chip networks may be used for different purposes and take on different forms by the microarchitecture to transfer instructions, operand values, cache/memory blocks, cache/memory coherency traffic, status and control information, etc. For example, of the eight on-chip networks implemented in the TRIPS EDGE research prototype architecture [23] for interconnecting its 106 heterogeneous tiles composing two processor cores and L2 cache integrated on-chip, two are 141-bit and 128-bit bidirectional mesh networks used for transporting operand and memory data, respectively, sent broadside; the remaining six are dedicated fan-out trees or recombination networks with unidirectional channels as wide as 168-bits for instruction dispatch and as narrow as 10-bits for global status and control. Alternatively, the Cell Broadband Engine (Cell BE) [32]—a commercial heterogeneous multicore processor—employs the Element Interconnect Bus (EIB) as its on-chip network used for transporting instructions and data between twelve elements interconnected on the chip.

The goal in designing wire-efficient on-chip networks, typically, is to achieve the highest possible effective bandwidth (i.e., throughput) and the lowest possible end-to-end latency for a given cost. In designing OCNs for specific or general purpose functions, it is important to understand the impact and limitations of various design choices in achieving this goal. As a first step toward understanding OCN design-performance tradeoffs in the context of a commercial heterogeneous multicore processor chip, we perform a case study on the Cell BE EIB. The EIB is an interesting OCN to study as it provides higher raw network bandwidth and interconnects more nodes than most mainstream commercial multicore processors. In this paper, we *analytically* evaluate the EIB network using conventional latency and throughput characterization methods as well as using a recently proposed 5-tuple latency characterization model for on-chip networks. Collectively, these enable us to identify the end-to-end control component of the EIB (i.e., the shared command bus) as being the main bottleneck to achieving minimal, single-cycle latency and maximal 307.2 GB/sec raw effective bandwidth provided natively by the EIB. This bottleneck can be exacerbated further by poorly designed Cell software which can significantly impact EIB utilization. The main findings from this study are that the end-to-end control of the EIB which is influenced by the software running on the Cell has inherent scaling problems and serves as the main limiter to overall network performance. This points to the importance of not overlooking control and other end-to-end effects when designing multicore networks on chip.

The rest of the paper is organized as follows. Section II gives an overview of the Cell BE architecture and the requirements placed on its OCN. Section III provides background on the network characterization methods used in this study. Section IV presents EIB latency and throughput characterization results gathered principally from analytical cycle-counting and calculations based on published data (from IBM). Section V presents an alternative view of EIB latency results gathered by application of the 5-tuple delay model proposed in [22]. Section VI describes how the Cell BE software can affect the performance of the EIB. Finally, Section VII briefly discusses future research directions, followed by Section VIII which concludes the paper.

II. CELL BROADBAND ENGINE: AN OVERVIEW

The Cell Broadband Engine (Cell BE), or Cell, is the result of over four years of development by the STI consortium—made up of Sony, Toshiba and IBM—to develop a high-performance media multiprocessor chip. For Sony, Cell is the central processing unit for the Sony PlayStation 3 gaming console. Toshiba plans to use the Cell for media processing in its high-definition television products. IBM provides a Cell BE-based server product for scientific computing applications and has a number of future Cell BE-based machine designs in development. In general, STI developed Cell to rapidly process large amounts of parallel data. The processing capabilities of Cell are made possible by its heterogeneous processors and its on-chip network architecture. The Cell BE architecture, shown in Fig. 1, is a heterogeneous multiprocessor on a chip (i.e., heterogeneous multicore) consisting of 12 core elements connected together through the EIB. It integrates one Power Processing Element (PPE), eight Synergistic Processing Elements (SPEs), one Memory Interface Controller (MIC), and one bus controller split into two separate elements (IOIF0 and IOIF1). The PPE is a complete 64-bit IBM Power architecture microprocessor featuring a 32KB L1 and a 512KB L2 cache that operates at the PPE core clock frequency of 3.2 GHz. The PPE is dual-threaded with in-order execution. It contains a VMX execution unit that supports SIMD and floating-point processing. Direct memory access (DMA) and memory-mapped input/output (MMIO) are both supported by the PPE. The SPEs are 128-bit SIMD processing units that use a new VMX-like instruction set architecture (ISA). Each SPE contains 256KB of Local Store (LS) memory and operate at 3.2 GHz. The memory flow controller (MFC), found in the SPEs, controls the memory management unit and DMA engine. DMA is the only method supported for moving data between LS and system memory. The MIC provides memory access of up to 512MB of RAMBUS XDR RAM. Finally, the bus controller is a RAMBUS FlexIO interface with twelve 1-byte lanes operating at 5 GHz. Seven lanes are dedicated to outgoing data; five lanes handle incoming data. The lanes are divided between the non-coherent IOIF0 and IOIF1 interfaces. IOIF0 can also be used as a coherent interface by using the Broadband Interface (BIF) protocol. The BIF protocol supports access to other Cell processors in multiprocessor Cell system configurations.

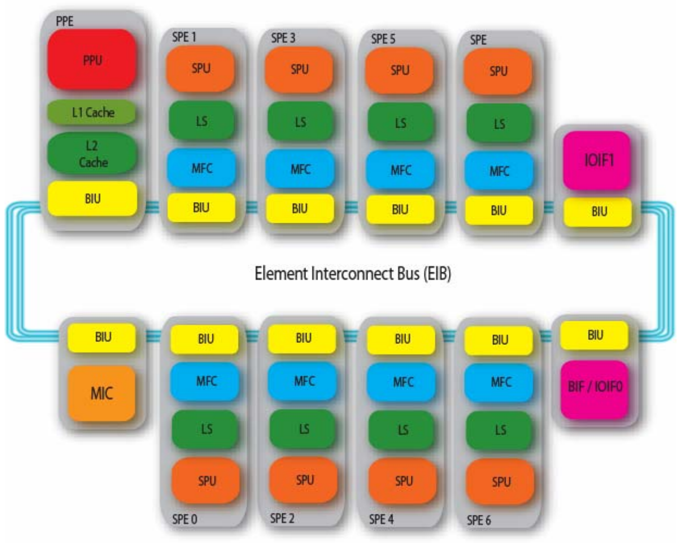

Fig. 1 – Block diagram of the Cell BE architecture, with the four EIB data rings shown.

The Cell BE architecture places a number of requirements on its on-chip network. The OCN must support the twelve elements in the architecture, which is greater than the typical two- to eight-node interconnect support needed by most other commercially available multicore processors to date. In addition, each Cell node is capable of 51.2 GB/s of aggregate throughput, so the OCN must be able to support very high data rates. In contrast, mainstream multicore processors such as those from Intel and AMD support far less: typically, less than 15 GB/s per core. The Cell design also requires the OCN to support coherent and non-coherent data transfers. Finally, the OCN must efficiently support the network traffic created by typical Cell software applications.

The Element Interconnect Bus is STI's solution to the OCN that is required by the Cell BE architecture. Each element contains a Bus Interface Unit (BIU) which provides the interface from the element components to the EIB. The EIB consists of four 16 byte-wide data rings (two in each direction), a shared command bus, and a central data arbiter to connect the twelve Cell BE elements [33]. The command bus, shown in the center of Fig. 2, distributes commands, sets up end-to-end transactions, and handles coherency. It is composed of five distributed address concentrators arranged in a tree-like structure to allow the possibility of multiple commands to be outstanding at a time across the network. It operates at 1.6 GHz, which is one-half the PPE core clock frequency of 3.2 GHz. The data rings also operate at the bus clock frequency of 1.6 GHz. Each ring is capable of handling up to three concurrent non-overlapping transfers, allowing the

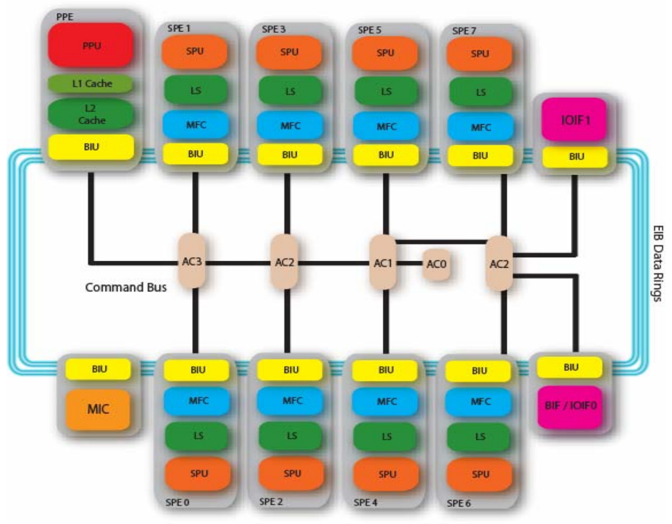

Fig. 2 - Element Interconnect Bus – the Command Bus component (center) and four data rings.

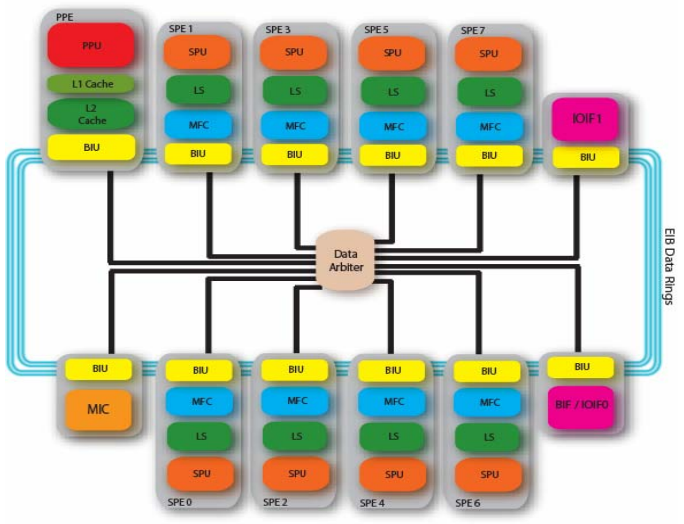

Fig. 3 - Element Interconnect Bus – the Data Arbiter component (center) and four data rings.

network to support up to twelve data transfers at a time across the EIB. The central data arbiter is shown in the center of Fig. 3, implemented in a star-like structure. It controls access to the EIB data rings on a per transaction basis. In the following sections, performance capabilities of the EIB are evaluated using conventional and recently proposed network characterization methods.

III. Network Performance Characterization

With the increasing importance of on-chip networks comes the need for useful methods of characterizing network performance. By characterizing network performance in a consistent manner, comparisons can be made between various network designs. Comparisons facilitate network architecture design tradeoffs by allowing the designer a clear view of the advantages and disadvantages of particular network design choices and, in particular, identification of potential performance bottlenecks. In this paper, we use conventional methods and a recently proposed method for characterizing the performance of the Cell BE EIB.

The conventional method of charactering network performance applicable to OCNs as well as other network classes is described in [34]. The two basic performance factors are *latency* and *throughput* (also called *effective bandwidth*). Latency is the amount of time (e.g., cycles or nsec) required for a network to perform a given action, such as a packet to traveling from one node to the next (i.e., hop latency) or from its source to its destination (i.e., end-to-end latency). The total latency of a network transaction is the summation of the latencies of each step of the transaction from beginning to end. Throughput is the amount of information transferred per unit of time (e.g., GB/sec). Throughput is calculated by dividing network packet size by either the transmission time or the overhead associated with sending or receiving the packet, whichever is the larger [34]. There are different values of throughput for various parts of the network end-to-end. For example, each node has a certain amount of sending (alternatively, receiving) throughput, called injection (alternatively, reception) bandwidth, which defines the rate at which it can inject (alternatively, receive) data into/from the network. Likewise, the network core fabric has a certain amount of bandwidth it can support across its bisection, called the bisection bandwidth. Following after [34], the overall throughput of the network is limited by the smaller of these accumulated bandwidths across the entire network. Network design aspects such as topology, routing, switching, arbitration, flow control, microarchitecture, etc., influence the latency and throughput performance of the network. In Section IV, we analytically evaluate the EIB using the above conventional methods.

An alternative method of characterizing network performance, as proposed by Taylor, et al., in [22], consists of describing the network in terms of a 5-tuple delay model. Although originally targeted to scalar operand networks (SONs), this method is equally applicable to OCNs as well as other network classes. A SON is an on-chip network designed to transport instructions and their operands across elements. One example of a SON is the traditional operand bypass network found in most superscalar pipelined commercial microprocessors, while another example is the static mesh OCN employed by the MIT RAW research prototype multicore processor [21]. The 5-tuple delay model comprises five costs presented as <SO, SL, NHL, RL, RO>. Send Occupancy (SO) is the number of cycles that a processing element spends occupied in doing the work needed to send a packet. Send Latency (SL) is the number of cycles a packet spends on the sending side of the network waiting to be injected into the network without occupying the processing element. Network Hop Latency (NHL) is the number of cycles needed to transport a message from one node to an adjacent node in the network. Receive Latency (RL) is the number of cycles an instruction needing the data on the receiving end spends waiting in an issue-ready state waiting for the remote data to be received and ready for use. Receiver Occupancy (RO) is the number of cycles the receiving processing element spends occupied in making the remote data ready for use. The 5-tuple delay model is meant to allow for comparisons of the operation-to-operand matching aspects of a SON. For more generalized OCNs, the 5-tuple delay model can be used as a tool for comparing different components of network latency from end to end. In Section V, we apply and evaluate the 5-tuple delay model to the EIB.

IV. Conventional Chacterization: Latency and Throughput Analysis

The EIB is a pipelined circuit-switched network that employs lossless flow control in the form of end-to-end command credits. An element on the EIB must have a free command credit in order to make a request on the command bus. This allows the EIB to have a simple, deterministic flow control design. The EIB allows up to 64 outstanding requests for each element on the bus. In order not to bottleneck network performance, the MIC also supports the maximum 64 requests. The SPEs, on the other hand, support only 16 requests.

The EIB's data arbiter implements round-robin bus arbitration with two levels of priority. This scheme works the same for the data arbiter as it does for the shared command bus. The MIC has highest priority while all other elements have lower priority, the same for each of them. The data arbiter does not allow a packet to be transferred along hops totaling more than half the ring diameter. That is, a data ring is granted to a requester only so long as the circuit path requested is no more than six hops. If the request is greater than six hops, the requester must wait until a data ring operating in the opposite direction becomes free. As briefly mentioned above, the command bus uses end-to-end command credits, or tokens, for control of the bus. Elements use tokens to post commands to the command bus. An element may send only as many commands as it has tokens. Each token is held by the command bus until the associated command is complete, at which point the token is returned to the element.

The Cell BE has a software-controlled Resource Allocation Manager that allows applications to manage EIB resources in software. The Resource Allocation Manager is an optional feature that allocates the usage of resources to prevent individual elements from becoming overwhelmed. It divides memory and IO resources into resource banks and separates requestors into groups. Software controls access to each group again using tokens.

IV.1 Best-case Latency Estimates

Given the above mechanics, the EIB network performance can be characterized in terms of packet latency and effective bandwidth (throughput). The EIB supports two types of data transfer modes: DMA and MMIO. The overall network latency is, essentially, the same for both transfer modes. The latency of a packet through the network end-to-end can be calculated using the following equation:

$$\text{Latency} = \text{Sending Overhead} + \text{Time of Flight} + \text{Transmission Time} + \text{Receiver Overhead} \quad (1)$$

EIB transactions can be divided into four phases: the Sending Phase, the Command Phase, the Data Phase, and the Receiving Phase. The Sending Phase is responsible for initiating a transaction. It consists of all processor and DMA controller activities needed before transactions are injected into any components within the EIB. At the end of the Sending Phase, a command is issued to the command bus to begin the Command Phase. The Command Phase coordinates end-to-end transfers across the EIB. During the command phase, the EIB informs the read or write target element of the impending transaction to allow the target to set up the transaction (i.e., data fetch or buffer reservation). The Command Phase is also responsible for coherency checking, synchronization, and inter-element communication (i.e., injection into BIUs). The Data Phase begins at the conclusion of the Command Phase. The Data Phase handles data ring arbitration and actual data transfers across the rings. Since end-point setup was completed during the Command Phase, the Data Phase is left with the task of granting access to one of the four data rings to packets when a ring becomes free and no more than six hops are needed along the ring. Pipelined circuit-switched transport of packets occurs end-to-end over the granted EIB ring. The Receiving Phase concludes the transaction by transferring received data from the receiving node's BIU to its final destination at that receiver, such as LS memory or system memory. With these four phases, the latency of packets across the EIB can be calculated using a revised version of the network latency equation above, where Sending Overhead is composed of the Sending Phase and Command Phase, and Time of Flight and Transmission Time combine to form the Data Phase:

$$\text{Latency} = \text{Sending Phase} + \text{Command Phase} + \text{Data Phase} + \text{Receiving Phase} \quad (2)$$

In the following, we calculate the number of processor cycles needed to execute the communication phases above (in the best case) for DMA transfers based on the required number and type of instructions needing to be executed and the microarchitecture of the elements executing them. All calculations assume ideal, best-case scenarios for simplicity, leading to lower-bound latency estimates. For example, the EIB is assumed to be unloaded—meaning that all queues are empty and no additional waiting due to resource contention or otherwise is incurred during arbitration. Also, no processor stalls are assumed to occur in executing instructions, i.e., due to cache misses or any other type of anomaly.

The Sending Phase of an EIB transfer is made up of the following: processor pipeline latency, DMA setup, and DMA controller processing. Both the PPE and SPE have 23-stage processor pipelines, yielding a pipeline latency of 23 processor clocks. The next step, DMA setup, requires 5 instructions to begin a DMA: 1) Write SPE Local Store Address, 2) Write Effective Address High (upper 32-bits), 3) Write Effective Address Low (lower 32-bits), 4) Write DMA size, and 5) Write DMA command (send, receive, etc). Since the DMA controller operates at the bus clock rate, these five instructions can complete in 10 processor clocks. Once the DMA is issued, the DMA controller takes an additional 20 processor clocks to select the command, unroll the bus transaction, send the command to the BIU, and then issue the command to the command bus [35], [36]. Thus, the Sending Phase takes a total of 53 processor clock cycles, which is 26.5 bus cycles.

The Command Phase consists of five steps [37], [38]: command issue, command reflection, snoop response, combined snoop response, and final snoop response. The length of the command phase depends on whether or not the transaction is coherent or non-coherent. The EIB master initiates a transaction during the command issue setup, which takes 11 bus cycles for coherent commands and 3 bus cycles for non-coherent commands. The command reflection step depends only on the distance an element is from the AC0 address concentrator. From AC1, command reflection takes 3 bus cycles. Two bus cycles are added for elements connected to AC2. Elements connected to AC3 add another 2 bus cycles for a total of 7 cycles of command reflection. Responses to the reflected commands are gathered during the snoop response step, which takes 13 bus cycles, regardless of coherency. In the combined snoop response step, the responses are distributed to all elements in 9 bus cycles for coherent transactions and 5 bus cycles for non-coherent transactions. The final snoop response ends the Command Phase, requiring an additional 3 bus cycles. In total, the Command Phase minimally requires 43 bus cycles for coherent transactions and 31 bus cycles for non-coherent transactions.

The Data Phase consists of data arbitration steps, data propagation, and transmission time. The Data Phase latency is independent of the coherency of the data. A transaction takes 2 bus cycles to make a request to the data arbiter, another 2 cycles for data arbitration and, assuming no contention, 2 more cycles to be granted access to a data ring by the data arbiter [38]. Time of flight of a DMA transfer after arbitration is limited to a maximum of six hops by design (half the network diameter). Each hop takes one bus cycle. Therefore, the worst-case time of flight is 6 bus cycles after arbitration. To make the EIB more deterministic, all packets are designed to be 128 bytes in size. Since each data ring of the EIB is 16 bytes wide, every transaction takes 8 bus cycles to complete. Therefore, transmission time is 8 bus cycles. All together, arbitration, worst-case time of flight, and transmission time take at total of 20 bus cycles to complete.

The Receiving Phase takes only 2 bus cycles for the SPEs and PPE. The first cycle is used to move data from the BIU to the MFC. Conservatively, the DMA engine in the MFC can write the 128 byte data packet into LS memory in the second bus cycle.

The end-to-end network latency for packets is obtained by summing each of the phases together. A non-coherent transfer minimally takes 79.5 bus cycles or 49.6875 nsec to complete. Coherent transfers minimally require 91.5 bus cycles or 57.1875 nsec to complete. Table 1 summarizes the EIB latency calculations described above. Again, we emphasize that the latency calculations should be considered as *best-case, lower-bound estimates*. Any bus arbitration time, network contention, stalls, or other delays will directly impact the packet latency across the EIB.

MMIO transfers essentially have the same latency as DMA transfers as they require the full Command and Data Phases mentioned above. All MMIO transfers also are coherent, meaning that some DMA transfers will actually have a lower latency. Regardless of their latency, MMIO transfers are far less efficient than DMA transfers as every EIB transfer assumes a packet size of 128 bytes, whether or not the full width is needed. DMA transfers use all 128 bytes, but MMIO transfers only use 4 or 8 bytes, resulting in significant under-utilization of the network bandwidth.

One major advantage of the EIB is its ability to pipeline DMA data transfers. The data arbiter can start a transfer on every bus cycle and each data ring supports a new operation every three cycles. This allows a single element to continuously pipeline transmissions on the EIB assuming there is low network contention, i.e., no conflicting overlaps of

TABLE 1 – EIB ZERO-LOAD PACKET LATENCY ESTIMATES – BEST-CASE, LOWER-BOUND RESULTS.

		Sending Phase (29% of Coherent Total) (33% of Non-coherent Total)			Command Phase (47% of Coherent Total) (39% of Non-coherent Total)							Data Phase (22% of Coherent Total) (25% of Non-coherent Total)					Receiving Phase (2% of C) (3% of NC)	Totals
		xPE Pipeline Latency	xPE Issue to Queue	DMAC Issue to Cmd Bus	Cmd Issue	Cmd Refl.	AC2 Cmd Refl.	AC3 Cmd Refl.	Snoop Resp.	Comb. Snoop Resp.	Final Snoop Resp.	Data Req. to Arbiter	Data Arb,	Data Grant	Time of Flight	Trans. Time	Overhead	Clock Cycles
Coherent	CPU Clock	23	10	20	22	6	4	4	26	18	6	4	4	4	12	16	4	183
	Bus Clock	11.5	5	10	11	3	2	2	13	9	3	2	2	2	6	8	2	91.5
Non-Coherent	CPU Clock	23	10	20	6	6	4	4	26	10	6	4	4	4	12	16	4	159
	Bus Clock	11.5	5	10	3	3	2	2	13	5	3	2	2	2	6	8	2	79.5

source-destination paths over a ring. The overlap provided by pipelined data transfers reduces sender/receiver overhead, and increases the effective bandwidth when considering the command phase and the data arbitration and propagation portions of the data phase (as we show below). Through pipelining, there are only 8 bus cycles or 5 nsec of latency between DMA transactions, equating to the 8 bus cycles of transmission time. Thus, since there are no wasted cycles between transmissions, the data rings can be considered as running at 100% efficiency in this case.

IV.2 Best-case Effective Bandwidth Estimates

The maximum effective bandwidth (throughput) required by a network design is determined by summing across all interconnected elements the maximum data rate at which packets can be injected (alternatively, received) at each element. Each element in the EIB is theoretically capable of sending and receiving 16 bytes per cycle, which equates to 25.6b GB/sec of injection (alternatively, reception) bandwidth. The total bidirectional effective bandwidth of an element is, therefore, 51.2 GB/sec. In order to accommodate all twelve elements, the EIB should be capable of providing a total throughput of 307.2 GB/sec, based on the effective bandwidths of each element. The EIB, as designed, indeed provides this in terms of raw network bandwidth. It supports a maximum of three concurrent transfers on each of its four 16-byte rings. Each ring can begin a new transfer every three cycles, and all transfers are eight cycles or 128 bytes long, which means that when the data rings are fully utilized, the EIB supplies a peak effective bandwidth of 307.2 GB/sec. Thus, *theoretically*, the EIB fully supports the bandwidth requirements of the twelve elements in the Cell BE processor. Unfortunately, as we show below, the EIB suffers from a number of limitations that, *realistically*, result in much lower effective bandwidth in the best case.

The EIB faces the same throughput limitations as all networks. For example, if a single element is in high demand, network throughput will be limited by the response of the most heavily-loaded element or set of links along the path to that element. In the Cell processor, the MIC serves as such a heavily utilized element as most SPE/PPE threads require regular access to system memory. In addition to this challenge, the design of the EIB poses additional limitations on effective bandwidth, as described below.

The first major limitation on EIB throughput is that which is imposed by the command bus. The command bus is severely limited in the number of bus requests it can process concurrently. For non-coherent transactions, it is limited to one request per bus cycle; for memory coherent transactions, the command arbiter can handle only one request every two bus cycles [39]. Thus, the non-coherent data rate is limited to 204.8 GB/sec, and the coherent data rate is limited to 102.4 GB/sec due to the shared command bus posing as a major bottleneck. All system memory commands such as MIC and PPE cache transfers are coherent commands. Some I/O commands, i.e., those typically for multi-Cell applications, are also coherent. Thus, a large fraction of EIB transactions (depending on the application) make use of only 33% of the raw effective bandwidth provided by the EIB network. The effect of coherency on EIB throughput need not be linear, however. As long as coherent transactions are limited to one command every other clock cycle, there is no loss due to coherency. However, if two or more coherent transactions enter the command queue consecutively, there will be bandwidth losses. Fortunately, SPE to SPE transfers are non-coherent, but even for these EIB transactions, only 66% of the raw effective bandwidth provided by the EIB network can be used. Thus, regardless of the additional limitations caused by coherent transfers, the design of the command bus prevents applications from being able to exploit the 307.2 GB/sec effective bandwidth provided natively by the four EIB data rings, even under best-case operational scenarios. In its current implementation, the EIB is unable to handle the full injection bandwidth of the twelve Cell BE elements.

The choice of a ring topology for the EIB serves as another possible design limitation in achieving full injection bandwidth. In order to support three packet transfers per ring, the transfers must occur over non-overlapping source-destination paths. If a transaction traverses an element, that element is unable to perform another transfer on the same data ring until the first transaction completes. A maximum transfer distance of six hops potentially prevents up to five elements from being able to access a data ring. In [5], IBM conducted a series of throughput tests using non-coherent transfers between SPEs to measure effective bandwidth under various conflicting and non-conflicting traffic scenarios over the data rings. Depending on the location of transfer sources and destinations, measurements show that throughput can range from 96% of the command bus-limited 204.8 GB/sec (realistic) maximum down to 38% of that maximum. Fig. 4 shows the traffic pattern used for the high-efficiency test that resulted in a throughput of 197 GB/sec. As illustrated, since no ring contention exists and all transfers are non-coherent, the command bus maximum of eight concurrent ring transfers can be achieved (*note*: this is less than the twelve concurrent ring transfers provided natively by EIB without the command bus limitation). Fig. 5 illustrates how conflicted overlapping transfers (i.e., transfers requiring the same links of a ring) limit the EIB to only one transfer per ring, thus further reducing the effective bandwidth down to 78 GB/sec. Coherent transfers would further throttle the network bandwidth.

Finally, the EIB effective bandwidth can be further reduced due to under-utilization resulting from less than full packet transfer size. The choice of optimizing the design for wide packets of 128 bytes can greatly diminish the utilization of allocated bandwidth when only a fraction of that packet size is actually used. For example, MMIO with its 4 or 8 byte transfers is the greatest culprit of bandwidth under-utilization. Fortunately, most transfers in the Cell BE are in units of 128 bytes, since the PPE uses 128 byte cache lines and the SPEs use 128 byte memory blocks. Under-utilization is a challenge for most network architecture designs. However, this issue is exacerbated in the EIB as poor packet efficiency also increases

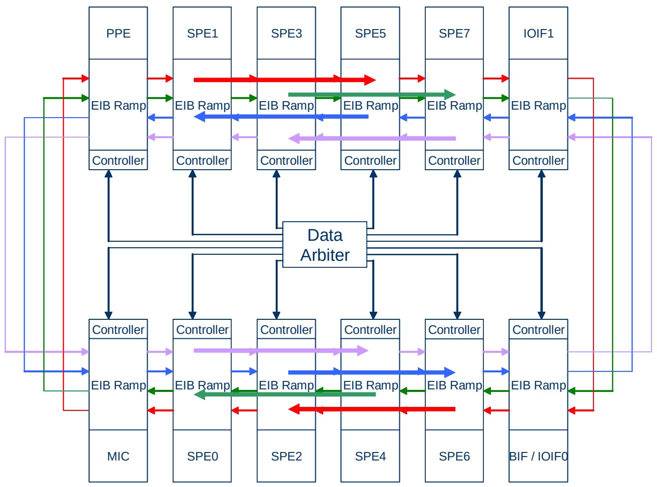

Fig. 4 – Non-conflicting traffic pattern for the high-efficiency transfer test.

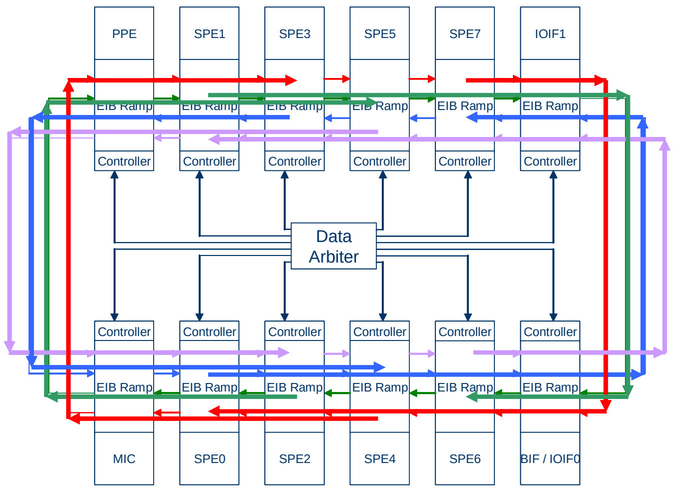

Fig. 5 – Conflicting traffic pattern for the low-efficiency transfer test.

data ring contention, which reduces the achievable effective bandwidth of the network.

V. An Alternative Characterization: The 5-Tuple Delay Model

The 5-tuple delay model proposed in [22] serves as an alternative method of characterizing network latency performance. Building from the best-case latency components given in Section IV.1, we assign values to the 5-tuple parameters as described below. Both the SPEs and the PPE take 5 bus cycles to start a DMA transaction, thus the Send Occupancy of the EIB is 5 cycles. Send Latency includes the time it takes the DMA controller to process the DMA, the entire command phase, and the data phase up to the point at which data is ready to be sent over the EIB data ring. This gives a total Send Latency of 59 bus cycles for coherent transfers and 47 bus cycles for non-coherent transfers. The EIB requires only a single cycle per hop between EIB elements, yielding a Network Hop Latency of 1 bus cycle. There is some difficulty in applying the 5-tuple delay model at the receiving side of the EIB. Receive Latency on the EIB can be 1 bus cycle if the DMA transaction has completed before the processing element accesses the data. That clock cycle is used to load the data from the Local Store into the processing element's register file. However, in the worst-case, the processing element may need to wait for an entire DMA to start, which can take over 90 bus cycles. Similarly, Receive Occupancy is zero cycles if the DMA has completed and the data is in memory since there is no additional penalty for using remote data found in the LS. If the data is not yet in the LS, then the Receive Occupancy cost would be significantly higher.

The level of abstraction of the EIB in matching operations to operands makes it difficult to compare it directly to true SONs. The EIB is designed to handle data at the system memory level, whereas SONs are designed to handle data at the operand level. Due to its higher level of abstraction, the EIB will naturally have higher 5-tuple costs than a typical SON. Nevertheless, using data from [22], [23], [40], Table 2 shows how the EIB compares with other types of networks. Not surprisingly, the EIB has a much larger 5-tuple delay cost than the superscalar bypass networks and SONs used to transport data operands. However, the EIB compares reasonably well with distributed shared memory and other message passing networks, both of which operate at the system memory level of abstraction. The high send latency cost of the EIB reveals a scalability issue with the EIB. While the data rings have an efficient 1 cycle per hop latency, the shared command bus takes a long time to synchronize all EIB interfaces. Given the longer latency, the EIB will have problems scaling beyond twelve nodes. In fact, it is likely that more nodes would require even more time during the command phase for synchronization, which would lead to an even higher Send Latency.

The 5-tuple delay model analysis also highlights the impact software has on EIB utilization. If a Cell programmer carefully plans out memory transfers, Receive Latency and Receive Occupancy can be much lower and more in line with those of the SONs. However, poor software programming could cause very large latency costs for the EIB if not tuned correctly. The effect of software on the EIB is discussed further in the next section.

TABLE 2 – SUMMARY OF THE 5-TUPLE DELAY MODEL FOR THE EIB IN COMPARISON TO OTHER NETWORKS

Tuple Costs	Cell BE EIB Coherent	Cell BE EIB Non-coherent	Typical Super-scalar bypass network	Distributed Shared Memory	Message Passing	M.I.T. Raw	M.I.T. Raw-Dynamic	Power4 bypass network	U.T. Austin TRIPS EDGE
Send Occupancy	5	5	0	1	3	0	0	2	0
Send Latency	59	47	0	14+commit latency	3+commit latency	1	2	14	0
Network hop latency	1	1	0	2	1	1	1	0	1
Receive latency	>= 0	>= 0	0	24	1	1	2	14	0
Receive occupancy	>= 0	>= 0	0	1	12	0	0	4	0

VI. SOFTWARE EFFECTS ON THE EIB

Estimating the packet latency and effective bandwidth values provide only part of the story when evaluating network architectures. Eventually, the applications that are to run on the network must be considered. Sections IV and V characterize the performance of the EIB only analytically as cycle-accurate simulation over the EIB cannot be performed using the publicly available Cell BE simulator [41]. Without considering the applications running on the Cell BE system, including its software optimizations, the EIB network cannot be fully evaluated.

Developing software for the Cell processor is the greatest challenge for Cell-based system design. This heterogeneous multicore processor requires a substantially different programming paradigm. IBM has written a five-part guide to provide an introduction to programming and compiling for the Cell architecture [42]. From the network perspective, the challenge for software is to balance the workload on the Cell processor with the strengths and limitations of the EIB. In general, EIB transactions have significant latency. To complicate matters further, the latency is not deterministic and varies based on contention and the transfer paths of the traffic. Thus, software development for the processor needs to take EIB latency and command bus bandwidth limitations into account. While software teams would like to treat all SPEs the same, Section IV shows that SPE transfer paths can drastically affect EIB performance.

There are three main application models for the Cell BE [43], [44]. The job queue model uses the PPE as the central processor which maintains a job queue and sends tasks to the SPEs. This model has lower EIB utilization than the other models since latency is of less concern. However, this model achieves only moderate performance as the SPEs are not fully utilized. The self-multitasking model is a distributed application model that operates across the SPEs. It is a high-performance application model, but it has increased synchronization requirements which place additional stress on the EIB. The last model, the stream processing model, is also SPE based. In this case, each SPE runs a single application and processes one part of the data stream. This model achieves high-performance as well, but it also highly utilizes the EIB. Of the three models, we believe analysis would show the stream processing model to be the best at balancing the latency limitations and bandwidth strengths of the EIB.

As mentioned above, EIB transactions have significant latency which has to be handled by software. By using the significant bandwidth of the EIB, Cell software can greatly reduce EIB latency. For example, by pre-fetching data, bandwidth can be traded to avoid the latency of unnecessary stalls. Software pipelining can also be helpful, but unpredictable access patterns may limit its effectiveness. Double buffering is used to stream data through the SPE LS, bypassing latency effects. Since the SPEs lack caches, a software cache can be used instead to handle workloads with temporal and spatial memory locality.

By using latency hiding techniques, software may be better able to utilize the abundant bandwidth provided by the EIB. However, there are some pitfalls that software should avoid to achieve optimal EIB performance, such as the under-utilization of packets. Cell uses a mailbox communication model with special registers that allow for communication between the PPE and the SPEs [45]. Some of these mailbox channels use MMIO which limit bandwidth utilization. In general, MMIO should be avoided as much as possible. Another problem area is SPE management. The SPEs are capable of context switches when necessary, but context switches take approximately twenty microseconds to complete, which is more than 33,000 bus cycles. The SPEs should always finish program execution to avoid this severe context switch penalty.

VII. FUTURE RESEARCH

One area of future research is to characterize typical Cell application code and its related EIB utilization using a cycle-accurate Cell simulator or actual Cell hardware. The Sony Playstation 3 and various Cell-based Linux distributions would be possible test platforms for such research. With such a platform, it would be possible to quantify the impact of software alluded to in Section VI. Without simulation or hardware testing, the effects of software can only be estimated, as is done here. Such tasks would require Cell code development, which implies another set of issues, but with the use of a test platform, the effectiveness of the EIB could be better understood.

Another area of research (possibly already in the works at IBM) is to consider design improvements for a second generation EIB. In [46], David Krolak points out that Cell is architected to allow the EIB to be easily replaced with a different network, such as a crossbar switch. He continues to say that a crossbar switch would take too much die space to be practical in the first Cell processor [46]. When IBM moves Cell to the next process generation, from 90 nm to 65 nm, it may be possible to include a richer network, such as a crossbar or higher-dimensional network. Improvements to end-to-end control of the network would certainly help, no matter what the network. Again, such design upgrades could be tested using a cycle-accurate simulator.

VIII. CONCLUSION

The Element Interconnect Bus is a network design that faces a number of challenges. This paper attempts to identify a number of these challenges by characterizing network performance using conventional and recently proposed methods based on analytical analysis. Network performance characterization in traditional terms of packet latency and effective bandwidth is a good starting point to understanding OCN design choices, but deeper analysis can be done in order not to evaluate the network in a vacuum. The entire system, including application characteristics and software optimizations, should be considered in future work to characterize the network more accurately.

Our findings reveal that the EIB suffers from latency and bandwidth limitations imposed by the mechanism used for end-to-end control. The complexity of the EIB design necessitates long Sending and Command Phases implemented over a shared command bus that provides limited concurrency. This greatly increases the packet latency and reduces the achievable effective bandwidth of the EIB. Although the data rings provide sufficient effective bandwidth, the command bus severely limits the use of that bandwidth. *This result points to the importance of not overlooking control and other end-to-end effects when designing multicore on-chip networks.*

Despite these limitations, the EIB is still capable of an effective bandwidth in excess of 100 GB/s, which compares favorably against current multicore processors. The 5-tuple delay model shows that the EIB is similar to other OCNs that transport memory data but that the EIB will not scale well beyond its twelve element design due to the command bus architecture. Design tradeoffs made by EIB designers require that software programmers strike a delicate balance when writing applications on Cell BE systems. In order to achieve the best performance, Cell software applications should make use of the relatively large bandwidth provided by the EIB to overcome the long latencies of the network, while being careful not to underutilize nor over prescribe that bandwidth. If properly utilized by Cell software, the EIB can achieve near superscalar bypass network latencies at the receive end of the network.

Acknowledgements

We acknowledge the help and advice of Scott Clark, Mike Kistler, and Fabrizio Petrini in clarifying various aspects of EIB design features and operation. This work was supported, in part, by NSF grants, CCR-0311742 and CCF-0541200.

References

[1] William J. Dally, “Interconnect limited VLSI architecture,” in Proceedings of International Interconnect Technology Conference, pp 15-17, May 1999.

[2] William J. Dally and Brian Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks,” in Proceedings of the Design Automation Conference, pp 684-689, ACM, June 2001.

[3] Luca Benini and Giovanni De Micheli, “Networks on Chip: A New SoC Paradigm,” IEEE Computer, Volume 35, Issue 1, pp 70-80, January 2002.

[4] Ahmed Hemani, Axel Jantsch, Shashi Kumar, Adam Postula, Johnny O’berg, Mikael Millberg, and Dan Lidqvist, “Network on a Chip: An Architecture for Billion Transitor Era,” in Proceedings of the IEEE NorChip Conference, November 2000.

[5] Marco Sgroi, M. Sheets, A. Mihal, Kurt Keutzer, Sharad Malik, Jan M. Rabaey, and Alberto L. Sangiovanni-Vincentelli, “Addressing the system-on-a-chip interconnect woes through communication-based design,” in Proceedings of Design Automation Conference, pp 667-672, June 2001.

[6] David Flynn, “AMBA: Enabling Resuable On-Chip Designs,” IEEE Micro, pp 20-27, July/August 1997.

[7] “IBM CoreConnect,” Technical Report, www.chips.ibm.com/products/powerpc/cores, 2000.

[8] Kanishka Lahiri, Anand Raghunathan, and Ganesh Lakshminarayana, “LotteryBus: A New High-Performance Communication Architecture for System-on-Chip Designs,” in Proceedings of the Design Automation Conference, ACM, June 2001.

[9] Rakesh Kumar, Victor Zyuban, and Dean~M. Tullsen, “Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads, and Scaling,” in Proceedings of the International Symposium on Computer Architecture, IEEE Computer Society, June 2005.

[10] Jian Liang, Sriram Swaminathan, and Russell Tessier, “aSoC: A Scalable, Singlechip Communications Architecture,” in International Conference on Parallel Architectures and Compilation Techniques, pp 37-46, October 2000.

[11] Pierre Guerrier and Alain Greiner, “A Generic Architecture for On-Chip Packet-Switched Interconnections,” in Proceedings of the Design Automation and Test in Europe, pp 250-256, March 2000.

[12] Adrijean Adriahantenaina, Herv ‘e Charlery, Alain Greiner, Laurent Mortiez, and Cesar~Albenes Zeferino, “SPIN: A Scalable, Packet Switched, On-Chip Micro-Network,” in Proceedings of the Design, Automation and Test in Europe Conference,, March 2003.

[13] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johnny O’berg, Kari Tiensyrja, and Ahmed Hemani, “A Network on Chip Architecture and Design Methodology,” in Proceedings of the IEEE Computer Society Annual Symposium on VLSI, pp 105-112, 2002.

[14] Jacob Chang, Srivaths Ravi, and Anand Raghunathan, “FLEXBAR: A Crossbar Switching Fabric with Improved Performance and Utilization,” in Proceedings of the IEEE CICC, pp 405-408, 2002.

[15] A. Brinkmann, J. C. Niemann, I. Hehemann, D. Langen, M. Porrmann, and U. Ruckert, “On-chip Interconects for Next Generation Systems-on-Chips,” in Proceedings of the 15th Annual IEEE International ASIC/SOC Conference, pp 211-215, September 2002.

[16] Timothy Mark Pinkston and Jeonghee Shin, “Trends Toward On-Chip Networked Microsystems,” Technical Report, *to appear in International Journal of High Performance Computing and Networking*, University of Southern California, CENG-2004-17, December 2004.

[17] Joan-Manuel Parcerisa, Julio Sahuquillo, Antonio Gonzalez, and Jose Duato, “Efficient Interconnects for Clustered Microarchitectures,” in Proceedings of 2002 International Conference on Parallel Architectures and Compilation Techniques, September 2002.

[18] Aneesh Aggarwal and Manoj Franklin, “Hierarchical Interconnects for On-chip Clustering,” in Proceedings of International Parallel and Distributed Processing Symposium, April 2002.

[19] Shubhendu S. Mukherjee, Peter Bannon, Steven Lang, Aaron Spink, and David Webb, "The Alpha 21364 Network Architecture," in Proceedings of Hot Interconnects 9, pp 113-117, August 2001.

[20] Gurindar S. Sohi, Scott E. Breach, and T. N. Vijaykumar, "Multiscalar Processors," in Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp 414-425, June 1995.

[21] Michael Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matthew Frank, Saman Amarasinghe, and Anant Agarwal," The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs," IEEE Micro, Volume 22, Issue 2, pp 25-35, March/April 2002.

[22] Michael Bedford Taylor, Walter Lee, Saman P. Amarasinghe, and Anant Agarwal, "Scalar Operand Networks," IEEE Transactions on Parallel and Distributed Systems, Volume 16, Issue 2, February 2005.

[23] Doug Berger, Steve Keckler, and the TRIPS Project Team, "Design and Implementation of the TRIPS EDGE Architecture," ISCA-32 Tutorial, pp 1-239, June 2005.

[24] Ramadass Nagarajan, Karthikeyan Sankaralingam, Doug Burger, and Stephen W. Keckler, "A design space evaluation of grid processor architectures," in Proceedings of the 34th Annual International Symposium on Microarchitecture, pages 40-51, December 2001.

[25] H. Peter Hofstee, "Power Efficient Processor Architecture and the Cell Processor," in Proceedings of the Eleventh International Symposium on High-Performance Computer Architecture (HPCA-11), IEEE Computer Society, February 2005.

[26] Kevin Krewell, "Sun's Niagara Pours on the Cores," Microprocessor Report, pp 1-3, September 2004.

[27] James M. Baker Jr., Brian Gold, Mark Bucciero, Sidney Bennett, Rajneesh Mahajan, Priyadarshini Ramachandran, and Jignesh Shah, "SCMP: A Single-Chip Message-Passing Parallel Computer," The Journal of Supercomputing, Volume 30, pp 133-149, 2004.

[28] Terry Tao Ye, Luca Benini, and Giovanni De Micheli, "Packetized On-Chip Interconnect Communication Analysis for MPSoC," in Proceedings of the Design, Automation and Test in Europe Conference, pp 344-349, March 2003.

[29] Peter N. Glaskowsky, "IBM rasies curtain on Power5," Microprocessor Report, Volume 17, Issue 10, pp 13-14, October 2003.

[30] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le, and Balaram Sinharoy, "POWER4 system microarchitecture," IBM Journal of Research and Development, Volume 46, Issue 1, pp 5-26, January 2002.

[31] Seon Wook Kim, Chong-Liang Ooi, Il~Park, Rudolf Eigenmann, Babak Falsafi, and T. N. Vijaykumar, "Multiplex: unifying conventional and speculative thread-level parallelism on a chip multiprocessor," in Proceedings of International Conference on Supercomputing, pp 368-380, June 2001.

[32] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the Cell multiprocessor," IBM Journal of Research and Development, Volume 49, Number 4/5, 2005.

[33] Thomas Chen, Ram Raghavan, Jason Dale, Eiji Iwata, "Cell Broadband Engine Architecture and its first implementation: A performance view," 29 Nov 2005, http://www-128.ibm.com/developerworks/power/library/pa-cellperf/

[34] Timothy M. Pinkston and Jose Duato, "Appendix E: Interconnection Networks" in Computer Architecture: A Quantitative Approach, 4th Edition, by John L. Hennessy and David A Patterson, pp 1-114, Elsevier Publishers, 2007.

[35] Mike Kistler, Michael Perrone, and Fabrizio Petrini, "Cell Microprocessor Communication Network: Built for Speed," IBM Austin Research Lab White Paper

[36] Mike Kistler, (Private Communication), June 2006

[37] Scott Clark, (Private Communication), December 2005

[38] Scott Clark, (Private Communication), June/July 2006

[39] David Krolak, "Just like being there: Papers from the Fall Processor Forum 2005: Unleashing the Cell Broadband Engine Processor: The Element Interconnect Bus," 29 Nov 2005, http://www-128.ibm.com/developerworks/power/library/pa-fpfeib/

[40] Michael B. Taylor. "Scalar Operand Networks for Tiled Microprocessors," invited talk at the *Workshop on On- and Off-chip Interconnection Networks*, Stanford University, December 2006, http://www.ece.ucdavis.edu/~ocin06/program.html.

[41] Power Architecture technology editors, developerWorks, IBM, "Meet the experts: The Mambo team on the IBM Full-System Simulator for the Cell Broadband Engine processor," 22 Nov 2005, http://www-128.ibm.com/developerworks/library/pa-expert7/

[42] Power Architecture technology editors, developerWorks, IBM, "An introduction to compiling for the Cell Broadband Engine Architecture," Parts 1-5, Part 1: 07 Feb 2006, http://www-128.ibm.com/developerworks/edu/pa-dw-pa-cbecompile1-i.html

[43] Power Architecture technology editors, developerWorks, IBM, "Just like being there: Papers from the Fall Processor Forum 2005: Unleashing the Cell Broadband Engine Processor: A programming model approach," 16 Nov 2005, http://www-128.ibm.com/developerworks/library/pa-fpfunleashing/

[44] Power Architecture technology editors, developerWorks, IBM, "Meet the experts: Alex Chow on Cell Broadband Engine programming models," 22 Nov 2005, http://www-128.ibm.com/developerworks/library/pa-expert8/

[45] Vaidyanathan Srinivasan, Anand Santhanam, Madhaven Srinivasan, "Cell Broadband Engine processor DMA Engines, Part 1: The little engines that move data,"06 Dec 2005, http://www-128.ibm.com/developerworks/power/library/pa-celldmas/

[46] Power Architecture technology editors, developerWorks, IBM, "Meet the experts: David Krolak on the Cell Broadband Engine EIB bus," 6 Dec 2005, http://www-128.ibm.com/developerworks/power/library/pa-expert9/

Other articles relating to Cell are available at the Cell BE Resource Center provided by IBM at http://www-128.ibm.com/developerworks/power/cell/articles.html