Optimizing Bus Energy Consumption of On-Chip Multiprocessors Using Frequent Values *

Chun Liu

Anand Sivasubramaniam

Mahmut Kandemir

Dept. of Computer Science and Eng.,
The Pennsylvania State University,
University Park, PA 16802.
{chliu,anand,kandemir}@cse.psu.edu

Abstract

Chip Multiprocessors (CMP) are a convenient way of leveraging from the technological trends to build high-end and embedded systems that are performance and power efficient, while exhibiting attractive properties such as scalability, reliability and ease of design. However, the on-chip interconnect for moving the data between the processors, and between the processors and memory subsystem, plays a crucial role in CMP design. This paper presents a novel approach to optimizing its power by exploiting the value locality in data transfers between processors. A Communicating Value Cache (CVC) is proposed to reduce the number of bits transferred on the interconnect, and simulation results with several parallel applications show significant energy savings with this mechanism. Results show that the importance of our proposal will become even more significant in the future.

Keywords: On-chip Multiprocessors, Value Locality, Power Optimization.

1. Introduction

Two important trends motivate the research to be presented in this paper. With higher levels of integration and the ability to pack millions of transistors on-chip, the important question to ask is how to employ these transistors. Numerous recent studies (e.g., [2, 10, 3, 7, 11]) show that it may be easier and more profitable to provision a number of small processing cores rather than a monolithic complex core. Several reasons for this argument include the wire delays, clock distribution, verification and overall system reliability. These multiple processing cores on the same chip can provide closely coupled on-chip parallelism, and there are many applications that can take advantage of such an

ability with thread-level parallelism. Using multiple (and perhaps slower) cores on the same die can turn out to be more rewarding than a single and powerful core [2]. The other motivating trend for this research stems from the power dissipation of these on-chip transistors, which becomes a serious concern with higher levels of integration and faster clock speeds.

Data communication becomes an important consideration when designing these chip multiprocessors (CMPs). The interconnect needs to provision communication channels between each processing core (termed a processor henceforth) and the memory subsystem, together with communication channels between the processors themselves. The latter is needed to enforce coherence and data transfer mechanisms between the processor caches that are needed to reduce the data access latencies to the memory subsystem. The on-chip interconnect between the processors, thus, plays a crucial role in determining the performance and scalability of these systems. Data buses that are used to implement these on-chip interconnects can also drain a considerable amount of dynamic power on each of these data transfers because of their high capacitances. For instance, the integrated switch in the Alpha 21364 [15] is estimated to consume 20% of the chip energy budget (125W), of which 33% is attributed to the data links [8]. These factors make it imperative to optimize the power drained by such on-chip interconnects.

Typically, an all-to-all interconnect is needed to facilitate communication between the processors, and also to the memory subsystem. In addition, coherence between the processor caches necessitates snooping capabilities to monitor accesses by other processors. A bus is typically employed because of its simplicity in design (compared to sophisticated interconnects), serializability (makes it easier to implement a stricter memory consistency model), broadcast capabilities (allows caches to snoop on other processor's requests), and the limited number of processing cores in use (8-16 is the range of focus in this paper). This bus carries the requests to the memory system for read/write misses, together with the corresponding responses (either from the memory subsystem or from some other processor cache), and the coherence control messages needed to keep the caches consistent. Although the evaluations of the idea to be presented use a bus architecture, the results are

---

* This research has been supported in part by NSF grants 0103583, 0130143 and Career Award 0093082.

equally applicable to other on-chip interconnects that can be employed.

There are two broad solution strategies for alleviating the power dissipated by the interconnect: (i) reducing the number of messages themselves, and (ii) reducing the bit switching activity by encoding the data sent on the buses [5, 14, 17, 18, 9]. The first one is a performance optimization technique and has been explored in depth in early research with different coherence protocols and message combining techniques [6]. The second solution strategy tries to encode patterns of ones and zeros in the exchanged messages to reduce the bit switching between successively sent flits on the buses. In this paper we present a different (and orthogonal) technique for addressing the interconnect power problem that can be used in conjunction with these two other solution strategies as well.

Our approach uses the principle of value locality [12, 20] for optimizing message traffic. We observe that there is considerable locality in the values for the data items that are communicated over the on-chip data bus (similar to what others have observed at the processor cache itself [12, 20]). We can exploit this locality by keeping a small content-addressable cache, called Communication Value Cache (CVC) at each processing core, that maintains the most commonly exchanged values. When a message is sent/received, the CVC is consulted to check if there are frequent values that are being communicated, and if so only the index of the location of these values (in the CVC) are actually communicated instead of the values themselves. We can get savings by keeping the size of the CVC relatively small (so that the index is small) as long as the program exhibits good value locality for such small sizes. This can lead to a reduction in the number of bits exchanged over the bus, and can provide power savings. Buses already provision abilities to shut down individual bit lines to benefit from this mechanism. While such techniques have been explored in the context of uniprocessor data exchanges between the cache and main memory [19, 5, 1], this is the first paper to investigate this idea in the context of multiprocessors (multiple processing cores on-chip). It is to be noted that the power consumption of the CVC itself should be discounted from the overall power savings, and our results include this overhead.

The next section gives an overview of the hardware and implementation of CVC. Section 3 discusses the experimental setup. The results from simulations are given in Section 4 and finally Section 5 summarizes the contributions of this work.

2. Hardware Details

2.1. CMP Overview

Our evaluations use an 8-way (8 processing cores and their L1 caches) CMP that employs a 36-line wide bus to interconnect these cores and to the rest of the memory subsystem (on-chip L2 and off-chip memory). We have aggressively assumed L2 to also be on-chip (as in [10]), though our ideas would automatically extend to the cases where it is off-chip (in which case it can provide even higher savings because off-chip capacitances are higher). A schematic of this architecture is given in Figure 1. Four lines of the bus are used for control signals (Read/write requests, asserting

shared lines, etc.). The remaining thirty two lines are used to carry address and data signals.

Figure 1. CMP Architecture Under Study

We use a 4-state MESI invalidation based protocol [6] that has been widely employed. In general, a cache block that is present in multiple L1 caches is in Shared (S) state, and moves to an Exclusive(E)/Modified(M) state once it is written by one of the processors, requiring a write signal on the bus which is snooped by the other caches to invalidate their copies. A write-back cache is used, where the contents of the block that is in Modified (M) state need to be propagated to the rest of the memory subsystem upon its replacement.

2.2. CVC Design

The idea behind optimizing the number of bits transferred between processor caches, or between an L1 cache and the memory, is based on exploiting the frequency of certain specific values that are communicated. The motivation for this idea stems from the results that are shown in Figure 2 which illustrate the fraction of data words communicated over the bus whose values are captured by the most frequent n values. Note that we are not performing any optimizations for address words and focus only on data value locality in this paper. The figures give the fractions for values exchanged between processor L1 data caches (L1-L1) and those between the L1 datacache and L2 (L1-L2) separately, in eight SPEC OMP2001 parallel applications (please refer to [16] for further information on these benchmarks). Similar results were observed for the other applications, and with other datasets for the same applications. We observe that even if we select the most frequent four values, these constitute as much as 85% of the data transfers (for swim), with the average being around 40%, between L1 caches. The L1-L1 value locality does turn out to be much higher than the L1-L2 value locality. Previous studies along similar ideas [19] have mainly looked at a single processor's cache and the rest of the memory subsystem, and our results here show that the value locality between the L1s is much higher for a CMP. Further, we also note that the value locality between L1-L2 is not as high for SPEC OMP2001 applications as compared to those reported in [19] for the Spec95 applications. The relative weightage of the number of bus transac-

(a) L1-L1

Category	n=4	n=8	n=16
swim	0.84	0.94	0.95
applu	0.43	0.54	0.67
galgel	0.28	0.28	0.28
equake	0.40	0.46	0.57
apsi	0.16	0.21	0.26
fma3d	0.49	0.54	0.62
art	0.08	0.54	0.81
ammp	0.09	0.09	0.11
average	0.35	0.45	0.53

(b) L1-L2

Category	n=4	n=8	n=16
swim	0.18	0.18	0.18
applu	0.30	0.34	0.36
galgel	0.44	0.44	0.44
equake	0.27	0.29	0.33
apsi	0.27	0.31	0.36
fma3d	0.28	0.28	0.30
art	0.43	0.47	0.50
ammp	0.12	0.12	0.14
average	0.28	0.30	0.33

An easy option of exploiting this value locality is to provision a Communicating Value Cache (CVC) at each processing core, and making sure that all the cores are in agreement on what goes into their CVCs together with exactly where they are placed within their CVCs. Consequently, a sender of a frequent value, can send the index of the location in the CVC of the destination rather than sending the value itself. Please note that we are trying to reduce the number of bits transferred on the bus (to reduce switching activity) rather than propose a narrower bus. The receiver can use this index to reassemble the value. There are several issues involved in the design of this CVC that impact its size. First, we do not want to make the CVC too large, or else we will incur a substantial energy cost per access (remember that this is a content-addressable structure), voiding any savings we may get from the bus. Second, the larger the CVC gets, more bits are needed to send the index which in turn eats away from the bus energy savings we are trying to optimize. Third, we also want to keep access times relatively low since the words need to be encoded/decoded using the CVC before/after they are sent/received on the bus. Since the encoding process can be done in parallel with the

considerable portion of the value locality with just 4 or 8 values (later results will show that larger CVCs are not very effective). Consequently, we opted to implement a CVC circuit that uses very few such frequent values.

In a CMP with n processors, we would have $n + 1$ CVCs overall - one at each of the cores, and one at the L2 end as shown in Figure 1. It is fairly straightforward to integrate the CVC into the architecture of bus based shared memory multiprocessors. When data needs to be sent out from a source L1 core (or from L2), we need to quickly look up the CVC and send out the data words as a sequence of either actual words (if they are not present in the CVC), or as an index in the CVC. There is one additional bit line of the bus (of the four control mentioned earlier) to indicate

which of these two categories a word belongs to and the energy for this bit is included in our experiments. Rather than perform all such lookups/encoding for a block before sending it out (which can incur a high latency), we pipeline the whole mechanism, i.e. the lookup for the second word of the block in the CVC is performed concurrently while the first word transfer goes on the bus. Similarly, at the receiver end, the lookups/decoding are pipelined. Consequently, the performance ramifications of a block transfer are not significantly affected. This mechanism may add two cycles (one at the sender and receiver each) overall per block for data communication, and with larger block sizes this overhead can become even less significant. The control/request messages do not incur any overheads because they do not go through the CVC.

2.3. Using the CVC

There are different ways of using this CVC structure. We could potentially have different contents (or at different locations within) for the CVC at the different processor cores. This can possibly optimize for cases where value localities are different across processors (or between pairs of processors). We noticed in our experiments that this was not a very significant issue, and decided to maintain the CVC as a structure that is mirrored across the cores (i.e. there is one CVC at each core, and the CVCs across the cores have the same values in the same locations).

We could opt to fill the values in the CVC dynamically based on execution characteristics, by tracking the frequent values during the run. For instance, one could use this as an associative cache and allow replacements from it. The advantage of this is that the optimizations can exploit dynamic changes in value locality. However, we found that such adaptation does not give better results than fixing the values statically, since the frequent values are more or less the same throughout the execution. Further, fixing the values can make the circuit much more efficient from both timing and power perspectives, and we consequently use the CVC as mainly a lookup mechanism and fix the values a priori.

These design choices make it easy to implement the CVCs and keep them coherent across the processor cores. In a bus-based system, it is possible for the CVCs to snoop the data transfers, and figure out what data the remote CVCs may contain and modify themselves appropriately. However, in a more general on-chip interconnect, the CVCs may not have any idea of what remote CVCs contain, and it may be necessary to explicitly propagate such information (which is quite costly) or maintain per processor-pair CVCs (which is not a very scalable design methodology). On the other hand, if we fix the contents (and their locations) of the CVC a priori (possibly by profiling the code), we can avoid these problems, making this mechanism general enough to work across diverse interconnects.

It should be noted that the CVC is orthogonal to the cache coherence mechanism itself. We are only optimizing the data values transmitted and are not changing the semantics of the messages themselves. One could hypothesize that if we are profiling and fixing the values, then it may be possible to recompile the code itself so that the indexes to the values get communicated instead of the values themselves in software. While this may be possible in some cases, the problem is that it may not be possible to statically identify

the references (or all of them) that correspond to these frequent values, while our hardware technique does this at run-time.

3. Experimental Setup

We have conducted detailed simulation studies of the bus energy behavior of CMP systems. Since we wanted to conduct complete system simulation studies, for real applications together with the operating system, we have used the Simics [13] simulation infrastructure. This infrastructure provides a memory system event generation mechanism, that has been used in conjunction with L1 and L2 simulations that we have written. The baseline (default) target hardware that have been used in the experiments is given in Table 1. Many of these parameters have also been varied to study their effectiveness on the energy savings. Similar to other related studies [1], the bus energy calculations are based on an average 50% bit switching activities per transaction and have been calculated using the formulations given in [21].

Process Technology	70 nm
Clock Speed	500 MHz
No. of CPUs	8
L1 Size	16 KB per CPU
L1 Associativity	4-way
L1 and L2 Block Size	32 bytes
CVC	4 entries, fully-associative
CVC Energy/Access	18.6pJ
Bus Width	32 + 4 lines
Bus Power/Access	11.6pJ/line
Bus Clock Speed	500 MHz

Table 1. Default Simulation Parameters

We have used several real-world applications from the SPEC OMP2001 suite [16] (reference input) that are given in Table 2. These are parallel applications using the OpenMP interface and are intended to evaluate the performance of shared memory multiprocessors. We have adapted these applications to run on the Simics environment. The simulation of these applications is extremely time-consuming, and we have collected statistics, after fast-forwarding over the first four billion instructions. The number of instructions and the number of bus transactions simulated are also given in Table 2.

In the following results, we are interested in the savings in bus energy, and also give the additional energy consumed by the CVC explicitly.

4. Results

As mentioned earlier, we are looking to pre-load the CVC with the values instead of dynamically tracking the frequent values. We could profile the code and compute frequent values, either at a global scale over the entire execution or within shorter epochs, and use such information to preload the CVCs. We refer to these as Globally Frequent and Locally Frequent values respectively, and investigate the benefits of these options first in the following discussion.

Figure 4. Energy Consumption of the Bus with CVC Compared to a Normal Bus. The results are given for (a) transactions between L1s, (b) transactions between L1s and L2, and (c) all transactions. The left two bars for each application are for Global Frequent Values with $n = 4$ (4,G) and $n = 8$ (8,G), and the right two bars (4,L) and (8,L) for each application are for Local Frequent Values. The lower part of each bar shows the CVC energy while the top part gives the bus energy. (Base Configuration)

4.1. Using Globally Frequent Values

Figure 4 shows the normalized energy that is obtained for the bus enhanced with the CVC compared to the normal bus. In these graphs, the first two bars for each application give the energy consumption of using the Globally Frequent Values for the CVC throughout the execution (i.e. they are pre-loaded before starting the application and the contents do not change) for CVC sizes of 4 and 8 entries respectively. These bars are in-turn broken down into two

parts - the bottom is the energy taken by the CVC itself and the top is that of the bus (typically the latter overwhelms the former showing that our scheme is not incurring significant energy costs). The three graphs in this figure correspond to the bus transactions (a) for L1 to L1 (L1-L1) transfers, (b) between L1 and L2 (L1-L2), and (c) for all transfers (L1 to L1 and between L1 and L2).

Corresponding to the value locality graphs shown earlier in Figure 2, we find the energy savings better for the L1-L1 transactions compared to the L1-L2 transactions. We

Benchmark	Description	Simulated Instns.	Bus Trans.
swim	water modeling	15720m	2099m
applu	parabolic/elliptic PDE	16696m	1802m
galgel	fluid dynamics	15080m	1492m
equake	earthquake modeling	35312m	2152m
apsi	pollution modeling	25600m	1783m
fma3d	finite-element crash sim.	95072m	2107m
art	neural network sim.	24800m	1998m
ammp	comp. chemistry	176344m	5661m

Table 2. Benchmarks (Instructions and bus transactions are in millions)

find savings as much as 70% (in swim) with average savings of 28% for $n = 4$ across the applications in L1-L1 transactions. The average saving for L1-L2 transactions is around 15%, which is still reasonably significant. When we examine the overall savings in energy across all transactions, we observe savings of around 20% on the average, showing the benefits of our CVC mechanism for reducing bus switching activity.

When we increase the size of the CVC from 4 to 8, we notice that there are no additional savings, and in fact, the energy consumption goes up in most cases (except art) because the number of index bits and per access CVC energy overshadow any savings from additional coverage of frequent values which we already observed earlier to be relatively small.

4.2. Using Locally Frequent Values

The previous set of experiments fixed the n frequent values throughout the run. It is possible that the value locality can change over the phases of a program, and a CVC that tracks such changes can provide even more energy savings. Consequently, we next explore the potential of dynamic value locality adaptation.

To explore this potential, we assume an oracle that has perfect knowledge of the frequent values during different epochs (defined to be 10,000 bus transactions in this study) of the execution, and it adjusts the CVCs of all processors to these frequent values before the epoch starts. We call this scheme Local Frequent Values. Note that this is a scheme which considers the limits/potential of the benefits of tracking locally frequent values though this may be difficult to implement in practice.

The graphs in Figure 5 plot the increase in hit ratio to the CVC by employing such a Locally Frequent Value CVC mechanism compared to a Globally Frequent Value CVC mechanism for each epoch in the execution, i.e. in each epoch i on the x -axis, the hit rate of the CVC with a Global Value CVC is subtracted from the hit rate of the CVC in another simulation which uses a Local Value mechanism. Higher the value of this difference, there is more scope for a locally frequent value mechanism to benefit from the CVC compared to fixing the values globally. These graphs are shown only for L1-L1 transactions in the interest of space. The actual energy consumption of this oracle-based Locally

Figure 5. Difference in hit rate between the Local Value CVC compared to Global Value CVC for each epoch shown on the x-axis. A larger value indicates that Local Value does much better than fixing the values globally for that epoch.

Frequent Value scheme is shown as the second set of two bars for each application in Figure 4.

We observe that in many applications (swim, galgel, apsi, ammp), the increases in hit ratio are not significant, and in fact the energy consumption for those applications in Figure 4 confirm that the Local scheme is not very different from choosing the Globally frequent values. In fma3d, we find that while the increases in hit ratio are significant at certain epochs (and these epochs display certain patterns) because of phase changes in programs, the consequent impact on energy in Figure 4 does not still seem to be significantly different from the Globally Frequent Value case. Finally, applications such as applu, equake and art show significant differences in hit rates in many epochs (and there are not decipherable patterns in these differences either). Even in these applications (particularly applu and equake), we do not see much differences in energy savings. This is because the epochs where these differences occur are themselves not very good exhibitors of value locality and choosing a locally good set of values does not make significant differences. Only in art (and that too only for $n = 4$ where the Local mechanism is able to select a better set of values than Global) do we find significant differences between Local and Global mecha-

nisms.

In general, we find that there is a significant fraction of epochs where the value locality is good and Global works equally well across these epochs (to give the energy savings in Figure 4). In the remaining epochs, the locality is anyway not very good and the energy behavior of Locally Frequent is not significantly different from Globally Frequent. This suggests that a simple policy of profiling the code and fixing the CVC to contain the globally most frequent values (across the run) is sufficient.

4.3. Sensitivity Analysis

In the interest of space, we focus on Globally Frequent Values and the fma3d application in the rest of this discussion (the results are representative of those observed in the other applications). When changing any one parameter, the others are fixed at their default values given in Table 1.

Number of Frequent Values (n) We have conducted experiments with varying number of entries for the CVC. We have already discussed the pros (better value locality) and cons (larger index bits, higher CVC energy cost per access) of larger CVCs and Figure 6 shows these trade-offs for fma3d. We find that even with a one entry CVC, we get good energy savings and the overheads start becoming more significant with larger n . In fact, the results presented until now (with a four entry CVC) are thus conservative, and one could get even more savings with a one entry CVC. Typically, zero turns out to be a frequent value in most applications (6 of the 8), and it is possible to simply hardwire this value to get good savings though this may defeat the flexibility of adjusting this value for the application at hand. On the other hand, the CVC can be customized for a specific application.

Figure 6. Sensitivity to the number of CVC entries (n). Energy is normalized with respect to the bus energy without CVCs. The lower part of each bar is the CVC energy and the top part is the bus energy.

L1 Size With improving technology, we expect L1 sizes to get larger. Figure 7 shows the energy savings of our mechanism across different L1 sizes, demonstrating that this scheme is likely to be equally effective in the future.

Figure 7. Sensitivity to the L1 Size. Energy is normalized with respect to the bus energy without CVCs. The lower part of each bar is the CVC energy and the top part is the bus energy.

Number of CPUs As the number of on-chip CPUs increases, which will be another trend for the future, the importance of bus energy optimization will increase. While L1-L2 traffic would grow at most linearly with the number of CPUs, the L1-L1 traffic would grow faster (possibly quadratically), making the savings with our mechanism even more pronounced. Figure 8 validates this observation. Note that our mechanism extends to other on-chip interconnects as well, if one wants to opt for a more scalable interconnect with larger number of CPUs.

Figure 8. Sensitivity to the number of CPUs. Energy is normalized with respect to the bus energy without CVCs. The lower part of each bar is the CVC energy and the top part is the bus energy.

5. Concluding Remarks

As integration densities increase, on-chip multiprocessors are likely to gain more prominence for both high-end

and embedded systems because of their attractiveness from the design, implementation, power, verification and scalability angles. At the same time, the importance of the interconnect that serves as the backbone for on-chip processing will be increasingly felt from both performance and power angles. This paper has presented a novel approach to optimize the energy consumption of these on-chip interconnects by exploiting the locality in values exchanged between processors. The Communicating Value Cache (CVC) can be used in conjunction with other bus power optimization techniques such as low swing buses, message reduction techniques and bus encoding techniques to amplify the savings.

We have experimentally demonstrated the benefits of this mechanism using several applications from the SPEC OMP2001 suite. We obtain over 40% energy savings in some applications with 20% savings of the total bus energy on the average across the applications. If we examine the L1 to L1 transactions, the savings are even higher. This effect is likely to benefit from larger number of CPUs, suggesting that our approach will be even more effective in the future. We have also shown that a simple strategy of profiling the code to pick the most frequent values and fixing this for the rest of the execution suffices to provide most of the energy savings. Since we need very few entries in the CVC, it would be possible to implement this structure in RAM without significant overheads compared to a CAM.

References

- [1] K. Basu, A. Choudhary, and M. Kandemir. Power protocol: reducing power dissipation on off-chip data buses. In *Proc. the 35th Annual International Symposium on Microarchitecture*, Istanbul, Turkey, November 2002.
- [2] L. Benini and G. DeMicheli. Networks on chip: A New Paradigm for Systems on Chip Design. In *Proc. DATE*, pp. 418-419, March 2002.
- [3] D. Bertozzi, L. Benini and G. DeMicheli. Low Power error resilient encoding for on-chip data buses. In *Proc. DATE*, pp. 102-109, March 2002.
- [4] J-H. Choi, J-H. Lee, S-W. Jeong, S-D. Kim, and C. Weems. A low-power TLB structure for embedded systems. *IEEE Computer Architecture Letters*, Volume 1, January 2002.
- [5] D. Citron and L. Rudolph. Creating a wider bus using caching techniques. In *Proc. IEEE Symposium on High Performance Computer Architecture*, January 1995, pp. 90-99.
- [6] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann, 1999.
- [7] DAC'02 Sessions: "Design methodologies meet network applications" and "System on chip design", New Orleans, LA, June 2002.
- [8] W. Dally, P. Carvey, and L. Dennison. Architecture of the Avici Terabit Switch Router. In *Proc. Hot Interconnects*, 1998.
- [9] W. Fornaciari, D. Sciuto, and C. Silvano. Power estimation for architectural exploration of HW/SW communication on system-level buses. In *Proc. International Workshop on Hardware/Software Co-design*, 1999.
- [10] L. Hammond et al. The Stanford Hydra CMP. *IEEE MICRO Magazine*, March-April 2000.
- [11] I. Kadayif, M. Kandemir, and U. Sezer. An integer linear programming based approach for parallelizing applications in on-chip multiprocessors. In *Proc. Design Automation Conference*, New Orleans, LA, June 2002.
- [12] M. H. Lipasti. Value Locality and Speculative Execution. *Ph.D. Thesis, Technical Report CMU-CSC-97-4*, Department of Electrical and Computer Engineering, Carnegie Mellon University, May 1997.
- [13] P. Magnusson et al. Simics: A Full System Simulation Platform. In *Computer*, February 2002.
- [14] H. Mehta, R. M. Owens, and M. J. Irwin. Some issues in Gray code addressing. In *Proc. 6th IEEE Great Lakes Symposium on VLSI*, 1996.
- [15] S. S. Mukherjee et al. The Alpha 21364 network architecture. *IEEE Micro*, 22(1), 2002.
- [16] SPEC OMP2001 (OpenMP) Benchmark Suite. <http://www.spec.org/omp2001/>
- [17] M. R. Stan and W. P. Burleson. Bus invert coding for low-power IO. *IEEE Transactions on VLSI Systems*, March 1995.
- [18] M. R. Stan and W. P. Burleson. Low power encodings for global communication in CMOS VLSI. *IEEE Transactions on VLSI Systems*, December 1997.
- [19] J. Yang and R. Gupta. FV encoding for low-power data I/O. In *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, Huntington Beach, CA, August 2001.
- [20] J. Yang, Y. Zhang, and R. Gupta. Frequent value compression in data caches. In *Proc. 33rd IEEE/ACM International Symposium on Microarchitecture*, Monterey, CA, December 2000.
- [21] H. Zhang and J. Rabaey. Low-Swing Interconnect Interface Circuits. In *Proceedings ISLPED*, pages 161-166, Monterey, CA, 1998.