SIMD-Emulations of Hypercubes and Related Networks
(extended abstract)

Fred S. Annexstein
Department of Computer Science
University of Cincinnati
Cincinnati, OH 45221-0008

Abstract

This paper addresses a problem in emulation theory. We show how processor-array networks with simple topologies can efficiently emulate the computations of complex topologies. This is possible by trading off parallelism for time. Such emulations are advantageous since processor-array networks of simple topologies are cost-effective to build on a large-scale. The challenge is to perform these emulations optimally, without the loss of too much parallelism.

We present emulations of generic computations programmed in a SIMD fashion which are all optimal (up to constant factors). Specifically, we present emulations of the order-n cube-connected-cycles network and the order-n shuffle-exchange network by an n-node ring-connected processor array. Also, we present an emulation of an order-n hypercube network by an $n/\log n$-node linear processor array.

1. Introduction

Our first motivation arises from the problem of how to efficiently *emulate* a large number of parallel, interconnected tasks by using a physically simple processor array. Our second motivation asks how such emulations can be programmed in a restricted SIMD model of computation.

Many parallel applications, e.g., physical simulations and image processing, uniformly process large data sets. Such applications can often be effectively programmed for SIMD architectures with interconnection topologies based on the hypercube, or its bounded-degree relatives, the shuffle-exchange and cube-connected-cydes (see, [2, 7, 8]).

In this paper we show that general computations programmed for SIMD hypercubes and related networks can be efficiently emulated by processor-array networks of simple topology. Our results suggest that when large amounts of data are processed uniformly on a small parallel machine, the choice topology of the underlying interconnection network is moot, i.e., a linear array is as efficient as any other network topology. Specifically, we show that for each n, the order-n cube-connected-cycles network and the order-n shuffle-exchange network can be emulated by an n-node ring-connected processor array. Also, we show that there is an emulation of an order-n hypercube network by an $n/\log n$-node linear processor array. These three results are shown to be *optimal* on general computations. We use the term optimal in the following two senses: (a) no larger array of the same topology can produce a faster emulation, and (b) no array of any topology (of the same size) can produce a faster emulation.

Results in emulation theory are offered in the work of Koch et al. [5], Bhatt et al. [1] and Obrenić et al. [6]. Our motivations relate to some results in [5] which do not apply directly to SIMD models of computation. Our challenge in demonstrating SIMD-emulations is to orchestrate memory accesses uniformly across the entire network-array. Our emulation technique augments traditional graph embedding approaches by employing "uniform" assignments of the processor-states to memory blocks.

2. The Formal Framework

Computation Model: We employ a very restricted model of synchronous parallel computation which we call a *pure* SIMD model. The hardware abstraction consists of a processor array – a set of interconnected processing elements (PEs) linked via broadcast bus to a controller. Each PE has (potentially large) local memory.

A program is executed by the controller broadcasting instructions and operands to the set of PEs. The set of PEs execute the same instruction on the same

operands and their own local data. Data addresses are decoded globally. Indirect addressing and other forms of independent memory access are not permitted. In a single time step, the set of PEs must process local data with the same address, which we call a *plane of memory.*

In this model, inter-PE communication is supported by an underlying interconnection network via message passing. At most one communication port per PE may be active during any step of the computation. This restriction deserves particular attention when the number of ports is potentially large, e.g., in hypercube networks.

Networks of Interest: For the host (emulator) we consider n-PE linear or ring-connected arrays. The nodes[1] of a ring-connected array are indexed by the integers $0 \leq i \leq n-1$; there is an edge-connection between each pair of consecutively indexed nodes i and $(i+1) \bmod n$. In a linear array the edge connecting nodes $n-1$ and 0 is absent.

The emulated guest networks are hypercubes and derivative networks. The *order-n hypercube* $\mathcal{Q}_n$ has 2^n nodes indexed by all n-bit strings. Each edge of $\mathcal{Q}_n$ is associated with one of n dimensions: An edge of dimension $0 \leq d \leq n-1$ connects a node $w = a_{n-1} \cdots a_0$ to the node $\text{Comp}_{(d)}(w) = a_{n-1} \cdots \bar{a}_d \cdots a_0$. The *order-n cube-connected-cycles* $\mathcal{C}_n$ has $n2^n$ nodes indexed by the set $\{0, 1, \ldots, n-1\} \times \{0,1\}^n$. The edges of $\mathcal{C}_n$ come in two flavors: *cycle* edges connect nodes of the form (i, w) with $(i \pm 1 \bmod n, w)$, and *cross* edges connect nodes of the form (i, w) with $(i, \text{Comp}_{(i)}(w))$ — where $\text{Comp}_{(i)}(w)$ denotes w with the i^{th} bit changed.

SIMD-Emulations: We seek an emulation of a generic SIMD computation of a "large" guest network $\mathcal{G}$ on a "small" host network $\mathcal{H}$. In part, this is accomplished via a *graph embedding* specified by: a (many-to-one) *assignment* $\alpha : \text{Nodes}(\mathcal{G}) \rightarrow \text{Nodes}(\mathcal{H})$; and a *routing* ρ of each edge (u, v) of $\mathcal{G}$ along a distinct path in $\mathcal{H}$ connecting nodes $\alpha(u)$ and $\alpha(v)$.

This standard definition of embedding does not address the central issue of memory management necessary for SIMD emulations. To this end, we logically partition the local memory of each $\mathcal{H}$-node h into a contiguous sequence of memory blocks or *extents*, denoted $\text{ME}(h)$. We assume each extent is large enough to store the state of a single $\mathcal{G}$ node.[2] Given $h \in \mathcal{H}$ and a bit-string w, we let $M\langle h, w\rangle$ denote the unique memory extent local to node h with base address w.

We augment the definition of embedding to include for each $h \in \mathcal{H}$ a (one-to-one) *process-to-memory mapping* μ_h of the nodes of $\mathcal{G}$ comprising the pre-image of h under α to the set of memory extents— $\mu_h : \{\alpha^{-1}(h)\} \rightarrow \text{ME}(h)$.

We assume that the number of extents for each node h is at least $|\{\alpha^{-1}(h)\}|$. Each extent will be addressed by a bit-string of length $\log(|\{\alpha^{-1}(h)\}|)$.[3] Let $\mu = \bigcup_h \{\mu_h\}$, and let the triple (α, ρ, μ) define this augmented embedding which we call a $\mathcal{M}$-embedding.[4]

Fundamental to our representing emulations by $\mathcal{M}$-embeddings is our assessing the *delay* incurred by a emulation. We now briefly describe our strategy for minimizing the delay. An emulation proceeds by sequencing through time the executions of $\mathcal{G}$-nodes mapped by α to a single $\mathcal{H}$-node. The sequencing we use is determined by the specifics of the patterns of communication. Sets of communicating processes must be sequenced so that their images under μ are memory planes. Specifically, the base address of memory extents identified with sets of processes constituting sources of messages must be identical. This should hold true for the destination-processes as well. This orchestration should be made clear with the following concrete example.

3. Emulating the C-C-C

The 2^n cycles comprising $\mathcal{C}_n$ present us with a natural projection mapping to the ring of n nodes. However, if we naively map each $\mathcal{C}_n$ node (i, w) to the memory extent $M\langle i, w\rangle$ we will not be able to accomplish the emulation of communication across the *cross* edges efficiently. This is due to the fact that message destinations become $M\langle i, \text{Comp}_i(w)\rangle$ which are dependent on processor indices. We proceed by manipulating the memory mapping to eliminate this dependence.

Let (α, μ) be the $\mathcal{M}$-embedding that maps each $\mathcal{C}_n$ node of the form (i, w) to the memory extent $M\langle i, \text{Shift}^{-i}(w)\rangle$, where $\text{Shift}^{-i}(w)$ denotes the bit-string w cyclicly shifted right i times. Let $w = a_{n-1} \ldots a_1 a_0$, then $\text{Shift}(w) = a_{n-2} \ldots a_1 a_0 a_{n-1}$ — exponents have the obvious interpretation. This memory mapping yields the projection to the ring we need to carry out the following SIMD-emulation program.

[1] We use the term node to refer to the combination of PE and its local memory.

[2] When emulating the hypercube it will be convenient to allow two states to be stored in a single memory extent.

[3] An address for a $\mathcal{G}$-node memory can be translated by simply concatenating it with the base address of the memory extent of its image under μ.

[4] Hereafter we omit ρ since the routings are trivial on a line.

SIMD-Emulation of C-C-C {
For $w := 0$ to $2^n - 1$ do — *for each extent*
  For all PEs $i := 0$ to $n - 1$ do — *in parallel*
    Load state of memory extent w
    Execute next instruction/computation step
    Generate 3 messages — *one for each edge*
    Send message 1 — *emulate cycle edges*
      to $M\langle i + 1 \bmod n, \text{Shift}^{-1}(w)\rangle$
    Send message 2 — *emulate cycle edges*
      to $M\langle i - 1 \bmod n, \text{Shift}(w)\rangle$
    Send message 3 — *emulate cross edges*
      to $M\langle i, \text{Comp}_0(w)\rangle$
  End;
End; }

Proof of correctness: Fix an n-bit string w. Consider the plane of memory extents given by $M\langle i, w\rangle$. The set of pre-images of this plane under μ is the set of nodes of $\mathcal{C}_n$ of the form $(i, \text{Shift}^i(w))$. *Cycle* edges emanating from this set of nodes are destined to nodes of the form $(i \pm 1 \bmod n, \text{Shift}^i(w))$. Note that the image of this set of nodes under μ are memory extents of the form $M\langle i \pm 1 \bmod n, \text{Shift}^{\pm 1}(w)\rangle$. Hence, the requirements for delivery of messages in parallel from $M\langle i, w\rangle$ to $M\langle i \pm 1 \bmod n, \text{Shift}^{\pm 1}(w)\rangle$ are simply communicating one step in the ring, followed by writing the new state to the extent $\text{Shift}^{\pm 1}(w)$—an address specified independent of the PE index.

To show the efficiency of *cross* edge communication, again consider the plane of memory extents given by $M\langle i, w\rangle$. The set of pre-images of this plane under μ are the set of nodes of $\mathcal{C}_n$ of the form $(i, \text{Shift}^i(w))$. *Cross* edges emanating from this set of nodes are destined for nodes of the form $(i, \text{Comp}_{(i)}(\text{Shift}^i(w)))$. The image of these nodes under μ are memory extents of the form $M\langle i, \text{Comp}_0(w)\rangle$. Hence, the requirement for delivery of messages from $M\langle i, w\rangle$ to $M\langle i, \text{Comp}_0(w)\rangle$ is simply writing the new state to the extent $\text{Comp}_0(w)$, in parallel. □

It follows that the preceding provides an emulation which runs in time $O(2^n)$. It is not difficult to derive the fact that no larger ring can provide a faster emulation, since $\mathcal{C}_n$ networks can route arbitrary permutations in time $O(n)$. The theorem follows.

Theorem 1. *There is an optimal SIMD-emulation of the order-n cube-connected-cycles on an n-node ring-connected array.*

4. Emulating Hypercubes

We define a $\mathcal{M}$-embedding of the 2^n nodes of $\mathcal{Q}_n$ into an n-node linear array with the following strategy: given an n-bit string w, defining a node of a $\mathcal{Q}_n$, remove a particular set of $\log n$ bits from w. Treat these removed bits as a number i_w between 0 and $n - 1$; let $\alpha(w) = i_w$. By slightly modifying the remaining bits we obtain w'; and define the $\mathcal{M}$-embedding as $(\alpha, \mu) : w \mapsto M\langle i_w, w'\rangle$.

The method we use to choose out those $\log n$ bits is detailed in the following:[5] Find the longest run of zeros in w, allowing such a run of zeros to wraparound, i.e., it may span the low and high-order bits.[6] After identifying this unique *zero-run*, remove $\log n$ consecutive bits from w—without touching the low-order bit—by choosing them (a) to the right of the zero-run, if possible without removing the low-order bit, or (b) if not, then to the left of the zero-run if possible, or (c) if not, then choose the last $\log n$ bits of the zero-run. Thus defining i_w.

Now, insert "10" or "01" at this point of removal insuring that the remaining bit-string has a unique longest run of zeros. These remaining $(n - logn + 2)$-bits define w'.

Our intention for this $\mathcal{M}$-embedding is to minimize communication requirements. We now demonstrate this. First, it is not difficult to show that each memory extent $M\langle i, w'\rangle$ will have either 0, 1, or 2 nodes of $\mathcal{Q}_n$ mapped to it depending on the position of the unique zero-run of w'. (The pre-image of $M\langle i, w'\rangle$ under μ will be empty if there is no unique run of zeros in w'.) Second, for each $0 \leq d \leq n - 1$, most values of w' result in only constant time communication overhead. We call a bit-string a *bad address (with respect to d)* if this is not the case. We show (Theorem 2.) that the ratio of *bad addresses* to the total address space is small $O(\log n/n)$. This $\mathcal{M}$-embedding yields the following SIMD-emulation program.

SIMD-Emulation of Hypercube {
Input: $0 \leq d \leq n - 1$ — *dim to communicate across*
For $w' := 0$ to $2^{(n - \log n + 2)} - 1$ do — *for each extent*
  For all PEs $i := 0$ to $n - 1$ do
    Load state of memory extent w'
    Execute next instruction/computation step
    Generate a message — *to pass across dim-d*
    If w' is a *bad address*
      then Send message
        via global routing algorithm — *time* $O(n)$
      else Send message
        to $M\langle i, Comp_{(d')}(w')\rangle$ — *d and w' determine d'*
    End;
End; }

[5] This method was adapted from an argument in [5] originally used in the context of embedding the shuffle-exchange.

[6] Break ties by choosing the left-most run.

Proof of correctness: Fix an $(n - \log n + 2)$-bit string w'. Consider the plane of memory extents given by $M\langle i, w'\rangle$. When w' is not a bad address communications can be done in constant time, since the source and destination PE-indices are identical, and the memory extent address is specified independent of PE-index. However, the total emulation time is dominated by the communication time for bad addresses. For each such address we charge a worst case time proportional to n, which is sufficient time to implement a generic global routing of n messages on the linear array. The following theorem shows that this does not occur too often.

Theorem 2. *For any $0 \leq d \leq n-1$, the number of* bad addresses (with respect to d) *is bounded by $O(2^n \log n/n^2)$.*

Therefore, an SIMD-emulation of the 2^n-node hypercube can be done by an n-node linear array in time $O(2^n \log n/n)$.

It follows from a wonderful result of Kahn et al. [3] that any embedding of an order-n hypercube into a line (of any size) must incur a delay of at least $\Omega(2^n \log n/n)$. Hence, we can derive an optimal emulation by simply compressing (in a balanced manner) our original n-node $\mathcal{M}$-embedding into an $n/\log n$-node linear array. The number of bad addresses may grow by a $\log n$ factor, but the communication time for routing is reduced by the same factor. We have the following corollary.

Corollary 4.1. *There is an optimal SIMD-emulation of the order-n hypercube by an $n/\log n$-node linear array.*

Proof sketch (Theorem 2): A bad address is a bit-string whose pre-image has a neighbor across dimension d which either has a zero-run in a different position, or has a different configuration of the $\log n$ bits immediately after (or before) that zero-run.

To bound the number of bad addresses we partition all bit-strings into disjoint sets called *necklaces*. A necklace is a set of bit-strings that are equivalent up to shifting. It follows that the length of the zero-run for all bit-strings in a necklace is the same.

The are very few bad addresses that are elements of necklaces with long zero-runs, as the following lemma shows (for a proof, see Leighton [4]).

Lemma 1. *The number of length-$(n - \log n + 2)$ bit-strings with a run of zeros more than $2\log n$ is $O(2^n/n^2)$.*

In any necklace with a longest run of zeros at most $2\log n$, only $O(\log n)$ strings are bad addresses. Since there are $O(2^n/n^2)$ distinct necklaces, in total there are $O(2^n \log n/n^2)$ bad addresses. □

Extensions

There is an optimal SIMD-emulation of the order-n shuffle-exchange by an n-node linear array. The proof will appear in a full paper.

Our results can be easily extended to show that there are optimal SIMD-emulations of butterfly and deBruijn networks by linear arrays. Also, by using the direct-product structure one can verify the existence of optimal SIMD-emulations of hypercubes and related networks on meshes and tori.

Acknowledgment

Thanks to Bojana Obrenić for her helpful suggestions.

References

[1] S.N. Bhatt, F.R.K. Chung, J.-W. Hong, F.T. Leighton, B. Obrenić A.L. Rosenberg, E.J. Schwabe (1991): Optimal emulations by butterfly-like networks. *J. ACM*, to appear.

[2] W.D. Hillis and G.L. Steele, Jr. (1986): Data parallel algorithms. *C. ACM 29*, 1170-1183.

[3] J. Kahn, G. Kalai, N. Linial (1988): The influence of variables on boolean functions. *29th IEEE Symp. on Foundations of Computer Science*, 68-80.

[4] F.T. Leighton (1983): *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks.* MIT Press, Cambridge, Mass.

[5] R. Koch, F.T. Leighton, B. Maggs, S. Rao, A.L. Rosenberg, E.J Schwabe (1990): Work-preserving emulations of fixed-connection networks. Submitted for publication; see also, *21st ACM Symp. on Theory of Computing*, 227-240.

[6] B. Obrenić, M.C. Herbordt, A.L. Rosenberg, C.C. Weems, F.S. Annexstein, M. Baumslag (1991): Using emulations to construct high-performance virtual parallel architectures. Tech. Rpt. 91-40, Univ. Massachusetts.

[7] F.P. Preparata and J.E. Vuillemin (1981): The cube-connected cycles: a versatile network for parallel computation. *C. ACM 24*, 300-309.

[8] D.B. Skillicorn (1988): A taxonomy for computer architectures. *Computer 21* (11) 46-57.