Using Packet Information for Efficient Communication in NoCs

Prasanna Venkatesh Rengasamy  
PACE Lab, IIT Madras, Chennai, India 600036  
Email: pras@cse.iitm.ac.in

Madhu Mutyam  
PACE Lab, IIT Madras, Chennai, India 600036  
Email: madhu@cse.iitm.ac.in

Abstract—Multithreaded workloads like SPLASH2 and PARSEC generate heavy network traffic. This traffic consists of different packets injected by various nodes at various points of time. A packet consists of three essential components viz., source, destination and data. In this work, we study the opportunity to increase the efficiency of the underlying network. We come up with novel methods to share various components of the packets present in a router at any time. We provide an analysis of the performance gains over contemporary optimization techniques. We conduct experiments on a 64 node setup as well as a 512 node setup and record the energy consumption, IPC gains, network latency and throughputs. We show that our technique outperforms the contemporary Hamiltonian routing by 8.43% and VCTM routing by 7.7% on an average, in terms of IPC speedup.

I. INTRODUCTION

Chip multiprocessor (CMP) designs support parallel execution of more than one application. As these applications try to co-exist in a chip during their execution time, performance limitation occurs from places where parallelism gets limited. This is studied by Amdahl’s law [1]. In a CMP, there are a lot of inherent sharing. For example, independent processes share resources like shared cache blocks, network connecting distributed cache banks with the cores and the memory system. These shared resources hinder the performance of parallel applications. Also, sharing of data at the cache level causes a lot of coherence traffic to get produced. We can see in Sections III and VI that when the number of cores in a CMP becomes large, performance of a parallel application becomes influenced by the efficiency of shared resources. We provide key optimization at the network-on-chip (NoC) level to gain better performance over existing techniques. We start by listing the different parts of an NoC packet, that we will use in this paper for optimizing traffic flow.

- Source, Data: Multicast optimization via a dynamic common path.
- Destination: Packet concatenation for saving route-computation (RC) time and virtual channel (VC) space.
- Address\(^1\): Arresting a read request and replying if the router VC already has the response data.

Our methods read one or more of the items above from the packets in router’s buffers to achieve better network performance.

\(^1\)This is a special case of data sharing

II. BACKGROUND

The past works in literature are examined in three categories of NoCs in this section. NoCs [2] manage communication between various components in a chip. Congestion control algorithms aim at providing better bandwidth in heavy-traffic scenarios. Most of such works treat a packet as a black box. Our work aims to provide better bandwidth by understanding what a packet transports. So, combining both congestion control and packet-peeping would result in better network performance. Some initial works on congestion control include Odd Even routing [3] and West-first routing [4]. These techniques get the framework ready for exploring optimized congestion control methods in NoCs. Both these methods give the idea of choosing one out of many available productive paths for a packet.

Adaptive techniques learn and choose the best path for each packet from the available list of paths. DyAd routing [5] switches between static and dynamic route computation methods based on the occupation levels of input queues. Thus, this method marks the low and high injection phases for a network. But, it does not do much good when it has to decide on options between the available set of adaptive routes. RCA [6] uses both crossbar demand and VC occupation levels for its congestion metric to choose a link from a set of adaptive routes. It then communicates the congestion metric with other routers in every region. It has good performance gains with some multi-threaded benchmarks too.

DAR [7] is an advancement over RCA where each router estimates and corrects its routes dynamically. Each router maintains some book-keeping to update the recent paths taken by flits in its buffers. This router design is complex and it scales better with network size when compared to RCA-QUAD, the best performing RCA technique.

BARP [8] uses scout flits to warn sources of a congested link. It has no turn restrictions and so has better path redundancy. It avoids deadlocks by bifurcating the network into XY+ and XY- networks. Thus this network model is much simpler when it comes to route computation. The only overhead for this technique is the scout flit traversals. It suffers a set back in performance when the network congestion blocks the scouts from reaching the sources to take timely decisions. HPRA [9] uses dedicated nodes in every region of a CMP for predicting the traffic flow. It uses Artificial Neural Networks (ANNs) in each region to compute "hotspots" and warn the nodes in the region. Thus the effectiveness of this technique is dependent on the versatility of the offline learning for ANNs.
GCA [10] optimizes the communication of congestion values by packing them up into the empty bit fields in the data packets. Unlike the previous techniques, this technique does not need extra communication. It stores the communicated congestion values in every router using minimal storage bits. This is the latest technique for congestion control and it outperforms RCA.

A. Congestion Control in Three Dimensional NoCs

Path redundancy improves with improvement in router degree. Thus 3D NoCs introduced better path redundancy in networks from the advent of Through Silicon Vias [11] and a TDMA bus on Z-direction [12] to full fledged routers like the MIRA router [13], DimDe router [14] and Roce-Bush [15]. These network implementations enabled a variety of routing and optimization techniques to achieve better performance. AFRA [16] achieves a fault tolerant deadlock free routing in 3D NoCs. Every router stores the nearest Z-link and a backup Z-link for fault tolerance. It follows dimension order routing when the destination is in the same XY plane of the current router. Elevator First [17] routes the traffic through a deadlock free algorithm on a partially connected Z network in a 3D NoC. This is similar to AFRA but it achieves Z-traversals by masking the header with a new header to go to the nearest node linking the destination XY plane. There are recent techniques to reduce congestion by handling multicasts efficiently too.

B. Multicast Routing

In any typical CMP, multiple private caches are attached to a (group of) processor core(s). They have the possibility of holding a copy of the same block of data at any given time. These copies get validated for correctness by communicating updates through the interconnection network. Such update messages are often seen as a critical factor for performance. As the number of sharers increases for a given data block, every update to the shared data may cause a multicast to the sharers in the network. Also, there are directory based protocols that generalize these multicasts as broadcasts for simplicity. Network takes these multicasts/broadcasts as multiple single packet injections of the same data. So, techniques like EVC-T [18], VCTM [19] and Hamiltonian [20] provide a static path construction technique. The aim of this path construction is to make a single injection into the network. This single packet moves along a common path viable for all the multicast destinations. When the packet reaches a point where it is not possible anymore to hop to a common router, the packet forks. This forking again is optimized to happen during link traversals. Thus, the above techniques avoid multiple copies from flooding the network. This whole process of a single packet propagating to all destinations create a tree-like path. This tree gets fixed for every group of multicast destinations. Thus this deterministic nature provides less path diversity in case of congestion. All these trees are maintained in small CAM tables at every router. These techniques save route computations by having static lookup tables for destination sets. Adaptive routing for multicast tree management have been proposed in XHiNoC [21]. It also uses a lookup table like VCTM to store routes computed. These techniques optimize route computations using lookup tables. But to fill the tables, a control flit is sent apriori for each multicast packet destination

B. Multicast Routing

In any typical CMP, multiple private caches are attached to a (group of) processor core(s). They have the possibility of holding a copy of the same block of data at any given time. These copies get validated for correctness by communicating updates through the interconnection network. Such update messages are often seen as a critical factor for performance. As the number of sharers increases for a given data block, every update to the shared data may cause a multicast to the sharers in the network. Also, there are directory based protocols that generalize these multicasts as broadcasts for simplicity. Network takes these multicasts/broadcasts as multiple single packet injections of the same data. So, techniques like EVC-T [18], VCTM [19] and Hamiltonian [20] provide a static path construction technique. The aim of this path construction is to make a single injection into the network. This single packet moves along a common path viable for all the multicast destinations. When the packet reaches a point where it is not possible anymore to hop to a common router, the packet forks. This forking again is optimized to happen during link traversals. Thus, the above techniques avoid multiple copies from flooding the network. This whole process of a single packet propagating to all destinations create a tree-like path. This tree gets fixed for every group of multicast destinations. Thus this deterministic nature provides less path diversity in case of congestion. All these trees are maintained in small CAM tables at every router. These techniques save route computations by having static lookup tables for destination sets. Adaptive routing for multicast tree management have been proposed in XHiNoC [21]. It also uses a lookup table like VCTM to store routes computed. These techniques optimize route computations using lookup tables. But to fill the tables, a control flit is sent apriori for each multicast packet destination

This deterministic nature provides less path diversity in case of congestion. All these trees are maintained in small CAM tables at every router. These techniques save route computations by having static lookup tables for destination sets. Adaptive routing for multicast tree management has been proposed in XHiNoC [21]. It also uses a lookup table like VCTM to store routes computed. These techniques optimize route computations using lookup tables. But to fill the tables, a control flit is sent apriori for each multicast packet destination

C. Cache Coherence at NoC

In the previous section, we studied techniques that treated multicasts differently from unicasts. To know whether a request is unicast or multicast, they peek into the packet for its designated bit. We could also peek into the packet data further to gain better knowledge on the system behavior. Thus there are techniques that use cache coherence information to optimize network performance. INSO [22] provides snoop ordering from NoC level to extend snoop-based protocol for topologies other than bus. This reduces broadcasts and so utilizes the available bandwidth efficiently for transmitting useful information. Helia [23] provides better power management by exploiting NoC-cache combination to reduce tag-mismatches. There are further areas available for exploration at the data sharing level, since multicast optimization motivates us to exploit sharing. Thus in Sections III and IV, we will see novel usage of packet data to improve performance.

III. MOTIVATION

A. General Congestion Control

We study and come up with the following arguments to emphasize on the reasons behind the optimizations proposed.

Communication in SPLASH2 [24] and PARSEC [25] has been analysed in depth in [26]. This analysis forms the basis for establishing the motivation of our optimizations. The following observations are from the analysis.

1) Multithreaded applications communicate shared data.
2) The authors analyze only the producer-consumer type sharing as shared communication in the paper.
3) Shared writes constitute up to 55% of the total writes and shared reads constitute to 9% of the total reads.
4) Temporal locality among these producer-consumer patterns show that communications occur in heavy bursts during synchronization phases, but light and random otherwise.

Also, we study the sharers for any packet in its lifetime in the network as configured in Section V and get the pattern as shown in Fig. 1. Fig. 1a shows the mean number of sharers in multicast messages sent by the high injection benchmarks.
The mean ranges from 2.2 for Barnes and 3.4 for Radix. From point number 3 in the above observation, we can see that up to 9% of the total reads constitute shared read. Thus if we are using techniques like EVC-T, VCTM or Hamiltonian, static route assignment for specific multicast destination sets may under-utilize the network. In addition, Fig. 1a indicate that there can be up to 28 sharers in a 32 threaded application (87.5%). Thus, sending scout flits for route-discovery to assign destination sets with output ports like VCTM and XHiNoC will also become undesirable. The only way for the network would be to pick the least congested port for every destination in a multicast at each router. This means that there may be up to $0.875 \times \text{NumThreads}$ route computations at every router. This will make the route computation a bottleneck.

VCTM and Hamiltonian techniques route a multicast packet along a fixed path. We show adaptive route for multicast packets perform better than them. But, doing so may create a bottleneck at the RC phase of the pipeline. So, to reduce the delay of computing routes for all multicast destinations, we propose redundant RC units. Fig 1a shows that the number of sharers can reach up to 87.5% [28 out of 32] of the number of threads. We study the percentage of occurrence of such multicasts in the execution phase and populate the results in Fig. 1b. It shows that such maximums occur only for 3.2% on an average across the benchmarks under consideration. To decide on the number of RC units to use, we use the mean sharers per multicast packet. This value is 2.3, and so it is enough to have two RC units in place of one. This reduces most of the stalls caused by computing route for each destination at every hop.

This approach of dynamic RC for every destination in multicasts may still starve other packets in the router. Our aim is to reuse the RC output of a flit to other flits going to the same destination. This technique is a variation from Packet Chaining [27] where switch allocator (SA) memoizes the output for faster allocation. Here, we propose a destination lookup circuitry, that concatenates packets with same destination. These concatenated packets traverse as a single “Super-Packet” till the destination. Thus we save some RC time on unicast packets by allowing greater flexibility on multicast packets path. Such packet concatenations are usually done for the request packets. They are just one flit long and a VC of more than one buffers is allocated for storing the request. The reasoning behind the coincidental arrival can be better explained by the point number 4 in the above study.

We see that a producer-consumer pattern occur during synchronization phases. At these phases the cores make a lot of requests to the same block of data. These requests arrive at the same L2-bank. This causes the L2-bank to fetch the block and multicast it back to the requesting nodes. These phases provide three different optimization opportunities for us.

- There are different single-flit request packets traversing towards the same destination. So, if there is a bottleneck due to an occupied RC phase as described above, packet concatenation can help.
- Assume that a request for a block is on its way to reach the L2 bank at time $t$. And the L2 bank has sent the same block of data to some other node at time $t - \Delta t$. Now, if the request and response happen to meet in any router on their ways, we can arrest the request and send the reply back. We should note that the request still has to reach the home L2 bank for coherence. But L2 need not respond with a reply unless there is an update from its previous response.
- Synchronization phases with high-injection to the network tend to wax out and occur in bursts of time. So, there is little to gain on top a request-limited network (phases with few injections). In these phases, we send the requested word along the header of the five-flit response packet. This will enhance the response from the receiving core to get the further requests inject quicker. This improves our opportunity to get better network utilization.

IV. APPROACH

Though we propose three optimization techniques, we do not modify much at the base router microarchitecture level. First, we classify a packet as unicast or multicast packet using a separate control bit in the header. A packet may be a request to read or write a cache block, a cache block itself or a coherence message. For simplicity, we consider them to be a single flit packet or a five flit packet at the router level. The components of a packet at the network layer are header, body and tail.

A. Addressing: Bit field and Subnet mapping

For a single flit packet, the single flit itself will serve as all the components. But, a five flit packet has 64 bytes of data and another 16 bytes of header. As mentioned in GCA, this 16 bytes in header is used for communicating the source node-id, destination node-id(s), the memory address in transmission, along with 80 bits of optional control information. We use these 80 bits and the 4 bits for destination-id to denote multicast destination address. Thus this scheme supports till 84 nodes in a network.

We also use the same bit field based destination encoding to denote a group of nodes in a network with size more than 84 too. In this paper, we consider one bit to denote one $2 \times 2 \times 2$ cube for a 512 node network. We can optimize the addressing further to associate the bit fields to various groups of nodes.
like the groups discovered in VCTM and XHiNoC. This is left as an extension in the future to this work.

B. Critical Word First

In this optimization, for all unicast packets, we put the word (8 bytes) requested in the header flit itself. We send the word to the requesting core once the header flit arrives at the destination to speedup the execution process. This is a simple rearrangement of the block data, and we assume it to happen at the cache-controller before injection.

C. VC As Cache

For this technique we have a tag+source-comparator at each input as shown in Fig. 2a. It checks the tag and destination for an incoming read request with those of every multicast responses staying in the VCs. If there is a match, it marks a special bit on the read request to ask the receiver to not send any reply back along with the current timestamp. This does not prevent us from forwarding the request till the destination because of the need of the home L2 bank to know the current sharers. It also sets the requestors bit to one in the multicast address of the response, to make the request get a quicker response. This also enhances the network utilization as discussed in Section VI.

This step occurs when such a request is waiting in its VC for RC, SA and link traversal (LT) stages of the pipeline. Thus it does not consume any critical path delay. A read request can also miss this optimization if there are a lot of read requests in the VCs. We simulated to check only till 5 responses per request. This is to meet the clock cycle demands as discussed in Section VI-B.

D. Dynamic Multicast Tree

A multicast packet contains the destinations encoded in the header as bit fields. To build a multicast tree, we compute minimal odd-even route [3] for each destination and decide to fork a packet or not. In a 3D NoC, a minimal route computation at any router will give utmost three productive output ports. We start concatenating packets when we have a request packet to inject into a router and there is no free VCs in it. In that case we compare the destination of the request to inject with the requests in the VCs. If there is a match, and if the VC has a free buffer we concatenate the packets. For example, consider the following scenario. The RC units of a cache bank are busy with a multicast-reply packet and there are no free VCs available at an input channel. To handle the cache miss, the cache bank has to access the appropriate memory controller with a read request. Let us assume that one of the VCs already holds a memory read request to the memory controller. If there are more such read requests, we can concatenate them to form a longer chain of flits in the same VC. As the size of a request is one flit, this case is possible.

This implementation continues as an extension to the VC as cache method we saw earlier. The controller at the input level compares the source address bits for request packets in the previous optimization. We add a similar controller to match destination [Refer Fig. 2b]. Similar to the VC as cache comparator, we limit only 5 comparisons for maintaining cycle time limit.

Thus by utilizing the otherwise idle buffers, we also reduce the number of route computations required by unicast packets going to the same destination.

V. EXPERIMENTAL SETUP

The simulation setup consists of the following phases:

A. Benchmark Execution

We use Multi2Sim4.0.1 [28] simulator for benchmark execution. We use Booksim2.0 [29] for our NoC level simulations. We modify Multi2Sim to let an instance of Booksim to manage its network modelling. Booksim services the network requests and intimates the Multi2Sim once the service is complete. We also modify the FastForward setup in Multi2Sim to warm up the cache during the FastForward. This is to start sharing of the same data by different cores. We then executed the multithreaded benchmarks from SPLASH2 and PARSEC workloads using this setup. We configure the simulators with the parameters given in TABLE I.

1) Architecture for Executing Benchmark: We take a 64−node system-on-chip (SoC) consisting of 32−cores with private L1 − caches and 32 L2 − cache banks. Then, we arrange it on a 3D NoC as a chess board like structure [30]. This paper discusses in detail about the thermal benefits with such an arrangement. We use this to demonstrate our technique. We fix this arrangement for all the techniques compared. We could also adopt any other topology or NoC configuration as per the need and implement our optimizations on top of them without any modifications to them.

As we target multicast traffic, we use L1 - L2 - Memory as our cache hierarchy. The advantage we get is the density of traffic that flows through this network is higher when compared to more levels of cache. This hierarchy is still practical as we have contemporary examples of this hierarchy in SPARC Processor [31] and Intel Xeon Phi Coprocessor. Vantage [32] validates cache partitioning with banked L2 cache. McPAT [33] calls this banked L2 cache shared among CPU cores as "future high throughput computing".

We used the cycle accurate full system simulator to run all the SPLASH2 and PARSEC workloads for 100K cycles after an initial warm-up phase for removing cold cache misses. This warm-up phase is decided empirically after observing the

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>L1 CACHES</th>
<th>L2 CACHES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 1</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>CPU 2</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONFIGURATION</th>
<th>L1 CACHES</th>
<th>L2 CACHES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 1</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>CPU 2</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORKLOAD</th>
<th>SPLASH2</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 1</td>
<td>9000</td>
<td>8000</td>
</tr>
<tr>
<td>CPU 2</td>
<td>9000</td>
<td>8000</td>
</tr>
</tbody>
</table>

For the thermal simulations, we use the cycle accurate full system simulator.
TABLE I: Configuration used in our simulations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Cores</td>
<td>32/256</td>
<td>We use 256 nodes to demonstrate scaling</td>
</tr>
<tr>
<td>- Core Properties</td>
<td></td>
<td>2.5GHz Clock Speed</td>
</tr>
<tr>
<td>Number of L1 Caches</td>
<td>32/256</td>
<td>One per core - not shared, 2 way set associative,</td>
</tr>
<tr>
<td>- L1 Properties</td>
<td></td>
<td>128 Sets, 64B block size, 2 cycle latency, 2 ports.</td>
</tr>
<tr>
<td>Number of L2 Banks</td>
<td>32/256</td>
<td>Shared by all L1 caches, split address space, 16</td>
</tr>
<tr>
<td>- L2 Properties</td>
<td></td>
<td>way set associative, 1024 Sets, 64B block size,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 cycle latency, 4 ports.</td>
</tr>
<tr>
<td>Number of Memory Banks</td>
<td>8</td>
<td>Shared by all L2 caches, split address space,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The 8 corners of 3D NoC</td>
</tr>
</tbody>
</table>

B. Network Simulation

Booksim2.0 [29] has built-in C++ models for two stage pipelined router, virtual channels, switch allocators and other routing components. As mentioned in Section V-A, we send network injections from Multi2sim4.0.1 to a Booksim2.0 instance. So, the simulators interact with a common cycle accurate clock. With this setup they can send and receive request/response and acknowledgments in real time. As a full system in operation, the traffic does not handle message deadlines. It is essentially a network talking to the architecture. When the response from the architecture gets delayed, the network does nothing about it. Booksim2.0 processes the injections and respond back to Multi2sim4.0.1 using a common clock. This communication is possible by linking the clock increments to a TCP socket. This socket synchronizes both the simulators and network injections are sent through this channel. Thus cycle accuracy is maintained. The injections and responses to/from the network are marshalled and unmarshalled as fixed length strings. Thus, there is no data loss with this model. We also built a new network topology to support z-direction jumps, with a specific router model for supporting the jumps. These two components are integrated to form a base network model for our simulations.

C. Hardware Modelling

We model a comparator in Verilog for both source+tag comparison of read requests and compare the responses present in the VC buffers for its match. This comparator is 38 bits long and is implemented as two stages. The first stage XORs two numbers and the second stage checks for equality of the two numbers by OR-ing all the 38 bits and checks for zero. We also model a similar 6 bit destination id comparator for packet concatenation. We develop a Verilog module for a crossbar, a basic routing algorithm module, virtual channels, input-output ports and the comparators along each input channel buffers. The buffers are made using a matrix of T flip-flops to the required dimensions. We then used Synopsys-Synthesis tool with 65nm technology. We also encode the necessary parameters for the router design into Orion 2.0 [34] and integrate the model to our simulator stack. We did this by using a MySQL table as a buffer between booksim and orion executables. Booksim dumps per cycle per router vc injections into the table. A daemon checks and executes orion for each of the dumped injection value and collects stats on various energy parameters reported by orion back into the table. This way, we will able to get accurate cycle level energy consumption details for every router in the network. The results from this modelling are discussed in Section VI.

We do not develop hardware level designs for all the techniques compared in this paper. We compare our 2D designs area, power and energy consumption with the base router models 2D only. We do not introduce any change to the 3D router design at all. It can be implemented using TSVs or using a full fledged DimDe like router too.

VI. RESULTS

We compare our optimizations on the base router with two contemporary techniques, namely VCTM and Hamiltonian for a 4×4×4 network as explained in Table I. The results obtained are discussed below. routing for any multicast chain.

A. Real Traffic from a Full System Simulator

The results from full system simulations can be seen from Fig. 3. The full-system performance is obtained from the committed instructions per cycle for each simulation run. System speedup graph populated in Fig. 3b shows variations due to the proposed optimizations over the base odd-even routing. We also compare the performance speedup with state
of the art VCTM and Hamiltonian methods. Though the GeoMean shows step by step speedup improvement from the base, there are certain individual benchmarks where it does not report such performance gains. Barnes posts most gains from dynamic multicasting. With other optimizations over dynamic multicasting, it brings down the performance achieved by the dynamic multicasting. This is because, Barnes has a high-communication phase prevalent for most of the time in its execution. Thus by doing critical word first, we tend to make the destination node act on the received data before the body and tail flits retire. Since it is a heavy injecting phase, this causes the destination node to inject a new packet quicker and tail flits retire. Since it is a heavy injecting phase, this causes the destination node to inject a new packet quicker and the network becomes crucial. Thus, it starts recovering from the damage once concatenated super-packet to never split again, we tend to make the packets’ efficiency to reach to get limited by the number of concatenations.

But, this is not always the case. For example, in both Radix and Swaptions, Packet Concatenation alone improves IPC to 60% and 20%, respectively. Radix gains because of its dense multicasts. As we can see from Fig. 1a, the number of sharers peak to 28. So, it could potentially mean that we could gain performance from any of our optimizations. There is not much to gain with VC as cache in here because, the multicast data get no extra additional requests later. Swaptions loose performance when using dynamic broadcast. As Fig. 1a explains, Swaptions has a few number of sharers. Also, the workload is sensitive to the order of requests. Since with dynamic multicast, we tend to fork at each common point, we delay some injection. This stalls some threads and causes the loss in performance. Though the network is able to transfer more flits in the execution at a lower average latency, the workload’s sensitivity on the data becomes crucial. Thus, it starts recovering from the damage using Critical Word First. When there is a stall in injecting into a router, VC as cache becomes a natural option for the workload to show huge gains. IPC Speedup jumps by 12% and it also beats VCTM. With Packet concatenation further reduces injection stalls and gains up to 20% in terms of IPC Speedup.

Hamiltonian and VCTM also gain with VC as cache on an average. We could also apply any of the aforementioned optimization on top of these multicast techniques. But the analysis on the impact of performance of these techniques is not done for brevity.

B. Chip Area, Clock Delay and Control Lines

As mentioned in Section V-C, all the components of the router are modelled in Verilog. We do not find any area overhead by adding one extra RC unit. But by adding two extra RC units we were able to get a slight increase in the area.
of 0.013%. The area overhead for fitting in two comparators for VC as Cache and Packet Concatenation techniques were found to be 1% of the total area of the router. We need one extra line in the link to indicate multicast and unicast packets. We do not impose additional delays to the router pipeline. The comparators take 0.063\textit{ns} for one full comparison. Since our clock delay is 4\textit{ns}, we could fit 6.33 comparisons per cycle. But we limit our comparisons to five to provide some slack.

C. Energy Analysis

As mentioned in Section V, we use Orion 2.0 to interact with our full system simulator to gain fine grained dynamic energy at every cycle. The crossbars and the pipeline units consume less than 1% of the total energy. Clock consumes 0.3%, while the pipeline stages consume around 0.02% of the total router energy. The remaining 99.5% of the total energy is controlled by the buffers. Since we use the idle buffers in the baseline model to our advantage by adding more flits to them, our model utilizes the buffers more. Thus we show the per-flit energy consumption on an average at each router in Fig. 3. As we can see from the flits transferred by the network in Fig. 3c, our techniques tend to utilize the network better by transferring larger number of flits most of the time. Thus the dynamic energy consumption of our technique is slightly greater than the baseline router on an average. Our Base+CVDP consumes 5.4\textit{J} on an average per cycle while the base design consumes 4.75\textit{J}. But for a fair analysis and point out the advantage of our optimizations correctly, we take a look at the energy spent on a flit by a router. A multicast packet is still treated as a single packet with five flits only.

Fig. 4c shows that our optimizations reduce the energy spent by a router on a flit in most cases. This reflects our philosophy behind using the idle buffers. But in cases like Cholesky, ocean or Radix, our overall optimization shoots up the energy per flit by a router. It shows that Packet Concatenation is a bad idea when congestion seem to alleviate quicker than the average packet latency. To understand this data, we take two cases - Barnes and ocean. In Barnes (refer to Fig. 4a), we gain up to 22.85\% with respect to the energy spent on a flit by the base technique. Thus the optimizations we do benefit the system on the whole. Another thing to see is the erratic behavior of the energy line by Dynamic Multicast. Since Barnes is a tree access algorithm, the threads accessing a particular block of data tend to be close to each other in the beginning. In those places, our forking take place almost near the end of the multicast tree. This is very efficient and shows up to 17.34\% energy savings. But as the algorithm progresses, the threads accessing the same block tend to be located at opposite corners of the mesh, making the packet forking to occur prematurely. Thus the multicast transfers start congesting the network and other optimizations start kicking in. Unlike multicast, the other optimizations occur only when there is a congestion. So, we see a smooth gain in energy with CD, CVD and CVDP.

On the other hand, Fig. 4b shows ocean to behave in the exact opposite way. This makes Packet Concatenation loose to the base by about 1.8\%. This again boils down to the traffic behavior. The energy gain in multicast traffic is almost similar to that of Barnes. But the congestion alleviation happens so rapidly that doing Packet Concatenation drops the energy usage to 1.79\% higher than the base. Also, VC as cache consumes higher energy than plain multicasts at many places. This is mainly because of the futile matches made by the router. Though there is a good request - response meeting at the intermediate routers, the tags requested are different most of the time. So, the comparisons cause the overheads in degrading energy per flit when compared to dynamic multicast. But we do post better geometric mean than the base technique. We save 1.66\% of energy spent per flit by a router with respect to that of base. Our energy savings peak to 17.7\% when running Radiosity.

D. Scaling to 512 Nodes

As we discuss sharing, we also evaluated our model in a 512 node setup and the results are shown in Figs. 5 and 6. The speedups against base in the 64-node by CVDP, Hamiltonian and VCTM are 18.4\%, 10.01\% and 10.6\%, respectively. Now in 512 nodes, the speedups are 7\%, 5.8\% and 5\%, respectively. This shows the scalability of our technique when compared to Hamiltonian and VCTM. Also CVDP gains 9\% in this 512 node setup while it gained 11.6\% in the 64 node setup. This steady behavior is from the fact that concatenated packets travel longer distances with increase in network size. Thus there is a degradation in performance from CV to CVDP by 2\%. VCTM+V gains 5.5\% and that is 10\% improvement coming from VC as cache. Hamiltonian+V on the other hand gains only 4\%. This loss of 31.08\% is because, the responses take a longer path to move along the Hamiltonian route.
VII. CONCLUSION

We proposed a scalable way of handling multicasts and demonstrated it to outperform the best techniques available today in two different network sizes using and real multithreaded benchmarks. We analysed the other possible optimizations to support this implementation and we discussed the way to incorporate it with the existing techniques. Thus, we could derive much better performance of the whole system by utilizing the knowledge of data transmitted in an NoC of any CMP. It is also cautioned that Packet Concatenation can be beneficial only when there is no alternative path to take. Though this would occur often in case of heavy workloads like the experiments we conducted with, it may not be the case all the time. In such cases, path diversity will be better than getting stuck behind a queue of packets.

ACKNOWLEDGMENT

The authors would like to thank VIRGO team, IIT Madras for their kind support. This work is funded by DST, Govt. of India.

REFERENCES