Designing Processor-cluster Based Systems: Interplay Between Cluster Organizations and Broadcasting Algorithms *

Debashis Basak and Dhabaleswar K. Panda
Department of Computer and Information Science
The Ohio State University, Columbus, OH 43210-1277
Email: {basak,panda}@cis.ohio-state.edu

Abstract– *Past research on designing processor-cluster based parallel systems has focused mainly on studying the packaging technologies affecting the inter-cluster network. To make such a design approach more attractive, there is a strong need to understand the details about the topology inside the cluster, its memory organization, and the impact of this organization on system performance. In this paper we analyze the communication costs for accessing inter-cluster and intra-cluster memories under different cluster organizations. The merits of these organizations are evaluated based on the performance of a commonly used U_mesh broadcast algorithm. Our results indicate that tightly coupled cluster organizations with shared access to memory offer faster intra-cluster communication. This leads to such organizations to outperform loosely coupled cluster organizations. We also demonstrate that such faster intra-cluster access in clustered systems can be exploited to design better collective communication algorithms. We propose a new broadcasting algorithm on clustered meshes -* clus_mesh *which outperforms existing u_mesh on clustered systems by up to 20%.*

1 Introduction

With advancements in VLSI and packaging technologies it has become cost-effective to integrate multiple processing elements into a multi-chip or board module [1]. This is leading to the development of parallel systems using such processor-clusters as building blocks instead of single processors, thus allowing a modular and hierarchical approach to building large systems. Prominent examples of processor-cluster based systems are the Stanford DASH, Intel Paragon, and Cray-T3D. Typically, the interconnections connecting the processor-clusters of these systems are scalable meshes, tori, or multistage networks.

In recent years research related to designing processor-cluster based systems has emphasized mostly on packaging and cost-efficiency aspects [1, 3]. These results are geared towards deriving optimal inter-cluster topologies under packaging and pinout constraints. While deriving such optimal topologies researchers have primarily emphasized on the load on network bisection and evaluated different topologies with respect to average message latency and throughput. However, the performance of a real system depends on several other factors like locality of reference, communication overheads, and performance of frequently-used communication patterns. In most of the above works the impact of *intra-cluster organization* on the performance of processor-clustered systems has been ignored.

The intra-cluster organization heavily depends on the programming model intended for a system. In order to support shared memory programming it has been shown to be beneficial to allow shared memory between processors inside a cluster [6]. However, it is not clear whether distributed memory systems need to provide shared-memory access between sibling processors (processors within a cluster)? From a naive point of view such sharing does not seem to be beneficial due to the message passing programming model. However, with a closer look it can be observed that in the absence of such sharing the benefits of clustering to exploit locality of reference are minimal. If four processors in a processor-clustered system are placed on a single board it is natural that faster communication between these processors can be achieved by providing some portions of shared memory between them (even though higher-level communications will be message-passing). This will lead to faster message passing between these processors and the applications can take advantage of locality. Otherwise processor-clustering may not lead to any performance gain. If we believe that such shared memory access are required the next questions are: 1) which intra-cluster organization will provide the best benefit? and 2) can existing collective communication algorithms take advantage of such memory sharing to deliver better performance?

In this paper we take on such a challenge to provide answers to the above questions. We study the interplay between intra-cluster organizations and the performance of collective communication operations, which are frequently used operations on distributed memory systems. We first analyze alternative intra-cluster organizations (star, bus, direct and crossbar) to study their respective capabilities to provide faster communication between sibling processors. We demonstrate that such faster communication can also lead to faster global operations like broadcasting. Our results indicate that bus and crossbar organizations can provide better performance improvement for broadcasting. We then demonstrate that existing collective communication algorithms (U_mesh, designed for non-clustered sys-

*This research is supported in part by NSF Grant MIP-9309627 and an Ohio State University Presidential Fellowship.

tems) may not be sufficient to take advantage of faster communication between sibling processors. We propose a new broadcast algorithm (Clus_mesh) to deliver better performance on processor-clustered systems capable of outperforming U_mesh by up to 20-25%. This study provides significant insight on how future processor-clusters should be designed for supporting distributed-memory paradigms.

2 Processor-cluster based systems

Processor-cluster based systems are two-level architectures as shown in Fig. 1. The processor-clusters are interconnected through a scalable inter-cluster network, e.g. meshes and tori. The cluster configuration can vary from a simple star connection as in the Cray T3D to a bus interconnection as in the Stanford DASH. These are discussed in detail in the next section. Currently the number of processors in a cluster is usually small, ranging from 2 to 4. However, with advancements in VLSI, larger clusters are expected to become common.

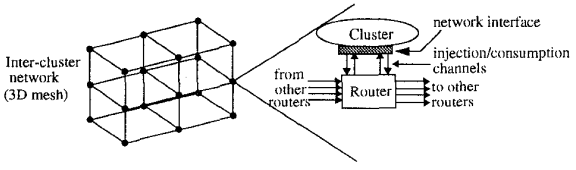

Figure 1: Typical example of a processor-cluster based system.

Current processor-clustered organizations connect each cluster to the inter-cluster network via a single interface, referred to as the network interface. This allows a system with a given total number of processors to be interconnected by using fewer network interfaces and network routers leading to a more cost-effective design [3]. The interface offers a number of channels to and from the network router. These channels (referred to as *injection/consumption channels* [2]) are used to inject/consume messages to/from other clusters. We use i to denote the number of injection or consumption channels per cluster. Currently, typical value for i is 1 to 2 [2].

3 Processor-cluster organizations and communication costs

In this section we present four different cluster organizations as shown in Fig. 2, two loosely coupled: star and direct-network and two tightly coupled: bus and x-bar. The inter-cluster network connecting clusters is kept the same when considering different cluster organizations. Let us analyze the communication delay to transfer m words to a processor's own memory from the memory of another processor. This basic operation is similar to the *shmem_get* primitive [4] supported on the Cray T3D system. In the following discussion we use parameters as summarized in Table 1. Explanation of these parameters and discussion on the different clustered organizations are detailed in [2]. The *shmem_get* transfer across clusters requires inter-cluster message communication. For popular wormhole routing a cost penalty of $(t_s + mt_p)$ cycles can be derived for this operation independent of the cluster organization. The time for a *shmem_get* operation from a memory inside the same cluster depends on the cluster organization. The communication delay for *shmem_get* for transferring m words in the four different cluster organizations can be derived as: star $(t_s + mt_p)$, direct $(t_s + mt''_p)$, bus $(t'_s + mt'_p)$, and crossbar $(t'_s + mt'_p)$. Detailed derivations and explanations for these expressions are presented in [2]. In this paper we assume typical values of $t_s = 200$ to 1000 cycles, $t'_s = 20$ to 100 cycles, $t_p = 10$ to 40 cycles/word, $t'_p \approx t''_p = 5$ to 20 cycles/word, and $m = 128$ words.

Table 1: Important Symbols Used in Paper.

b	Bus bandwidth in a bus-based cluster (words/cycle)
c	Size of each processor-cluster
i	Number of injection channels per cluster to router
t_s	Startup overhead for inter-cluster messages. In loosely coupled clusters (star and direct-network) it also represents startup overhead for intra-cluster messages
t'_s	Startup overhead for intra-cluster memory transfers in tightly coupled clusters (bus and x-bar)
t_p	Propagation delay per word in inter-cluster network and also in star intra-cluster network
t'_p	Propagation delay per word inside tightly coupled clusters
t''_p	Propagation delay per word inside direct-network based clusters
N	Total number of processors in a system
N_C	$(= N/c)$, total number of clusters in system

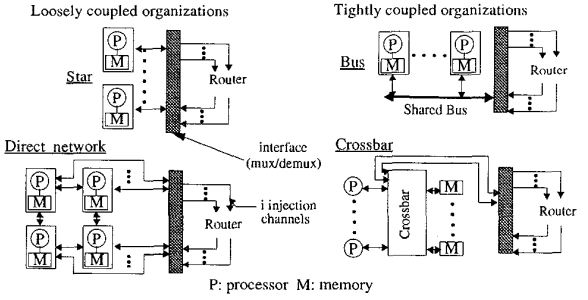

Figure 2: Four alternate processor-cluster organizations.

4 Evaluation of different processor-cluster organizations

In this section let us evaluate the above cluster organizations for efficiently supporting broadcasting, a collective communication algorithm which occurs frequently in applications. Similar analysis can be easily extended to

other collective communication operations such as multicast, gather, reduction, and barrier synchronization. The *U_mesh* algorithm has been proposed in [5] as an optimal algorithm to achieve multicasts and broadcasts in non-clustered mesh interconnected systems. To perform a multicast/broadcast to $(N-1)$ destinations the algorithm orders the $N-1$ destinations and source into a dimension-ordered chain and achieves the operation in $\lceil \log_2 N \rceil$ contention-free steps.

The U_mesh algorithm can be easily extended to clustered meshes to achieve broadcasting in contention-free steps in a two phase approach: an inter-cluster phase followed by intra-cluster phase in each cluster. In the inter-cluster phase the broadcast data is sent to one processor in each cluster. For a given system with N_C clusters, this phase requires $(\lceil \log_2 N_C \rceil)$ steps of inter-cluster message communication. Assuming the broadcast data consists of m words, the time for the inter-cluster phase can be derived as $(\lceil \log_2 N_C \rceil)(t_s + mt_p)$ [2].

The time for the intra-cluster phase is a function of the cluster organization. For the four different cluster organizations discussed earlier, Table 2 summarizes the time required for this phase. The basic approach for broadcasting inside a cluster is to employ, whenever possible, a U_mesh-like divide and conquer strategy. Detailed derivations of the expressions in Table 2 are presented in [2].

Table 2: Time required for intra-cluster phase of the U_mesh under different cluster organizations ($i \leq c$ and $b \leq c$ are assumed).

Cluster organization	Intra-cluster broadcast time (network cycles)
star	$(\lceil \log_2 i \rceil + \lceil c/i \rceil - 1)(t_s + mt_p)$
direct	$\lceil \log_2 c \rceil (t_s + mt''_p)$
bus	$(\lceil \log_2 b \rceil + \lceil c/b \rceil - 1)(t'_s + mt'_p)$
crossbar	$\lceil \log_2 c \rceil (t'_s + mt'_p)$

Figure 3 depicts plots comparing the time for broadcasting in a given system with a total of $N = 256$ processors with different cluster organizations. The impact of increasing cluster size ($c = 1$ to 16) processors and the number of injection channels per router ($i = 1$ and 4) was studied. From the plots it can be observed that the tightly coupled, bus and crossbar configurations, consistently outperform the loosely coupled, direct and star configurations, by at least a margin of 5% to 20% even for smaller cluster sizes. This demonstrates that tightly-coupled cluster configurations are indeed necessary to offer faster intra-cluster access and to fully exploit communication locality.

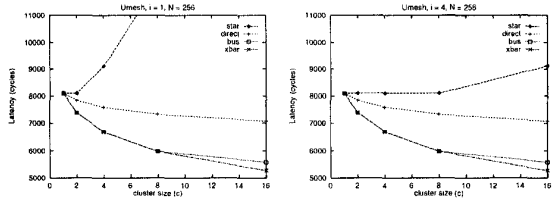

Figure 3: Impact of increasing cluster size on U_Mesh performance on four different clustered organizations.

5 Broadcasting on clustered meshes

Tightly coupled cluster organizations lead to faster intra-cluster communication. However, existing algorithms like U_Mesh[5] designed to be optimal under the assumption of *flat* communication cost may not remain optimal in clustered systems with *differential* intra- and inter-cluster costs and available multiple injection/consumption channels per cluster. In this paper we consider each cluster having two injection channels ($i = 2$) to the inter-cluster network router. Let us consider an example of the U_mesh algorithm for achieving a broadcast in a system with 9 clusters, $C0 - C9$, each having $c = 4$ processors, as shown in Fig. 4. U_Mesh requires 6 steps to complete the broadcast from processor 0 in $C0$. Let S denote a *slow* inter-cluster communication step and F denote *fast* intra-cluster communication step. Each arc representing a communication is numbered by the step in which it occurs and by S or F depicting inter- or intra-cluster communication, respectively. For sake of clarity only important intra-cluster steps have been depicted in Fig. 4. The completion time in this example, derived by tracing the longest arc-path to the last destination is $4S + 2F$ cycles. From Fig. 4 it can be also observed that U_mesh uses only *one injection channel* from each cluster thus not exploiting the multiple injection channels available in clustered systems effectively.

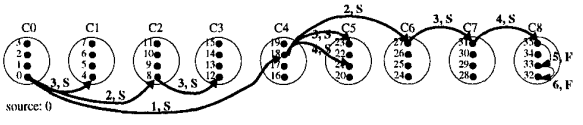

Figure 4: Example broadcasting with U_mesh algorithm.

5.1 A new algorithm - Clus_mesh

We propose a new broadcasting algorithm, Clus_mesh, for clustered systems which effectively exploits a) faster intra-cluster communication ($F < S$) and b) more injection channels from each cluster. The algorithm presented in this paper is designed for systems with two injection channels per cluster. However, a similar approach can be used to design an algorithm for systems with higher number of injection channels per cluster. The key idea behind Clus_mesh for a system with 2 injection channels, is as follows. The first processor in any cluster to receive the broadcasting data performs a fast intra-cluster communication to involve one more processor in the same cluster. The two processors then exploit the two injection channels to send

out two concurrent inter-cluster messages in two different directions to cover two more clusters. The potential gain arises from the faster step within the cluster which allows more clusters to be covered in lesser time.

Figure 5 depicts the communications performed to achieve broadcasting in 9 clusters. In the first step source processor 0 sends the data element to processor 16 in the center cluster (C4), which then performs an intra-cluster communication to involve another processor 17 in its cluster. In step 2 processors 16 and 17 perform inter-cluster communications to processors 4 (in C1) and 28 (in C7), respectively. This leads to two injection channels being effectively used in C4. Such steps are recursively repeated. The total broadcasting time as derived in Fig. 5 with Clus_mesh is $3S + 4F$ cycles. Comparing this to the $4S + 2F$ cycles for a similar broadcast using U_mesh, we derive that in this example Clus_mesh outperforms U_mesh if $S > F$. For broadcasting in larger systems, including 2D mesh interconnected systems, Clus_mesh has been shown to similarly outperform U_mesh [2].

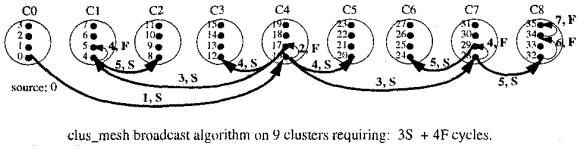

Figure 5: Example broadcasting with the new Clus_mesh algorithm.

5.2 Comparison of Clus_mesh and U_mesh performance

Figure 6 compares the time taken by Clus_mesh and U_mesh on a system having N_C clusters each of size c. In the two plots in top row of Fig. 6 the cluster size is kept fixed at $c = 2$ while studying the impact of increasing number of clusters from $N_C = 2$ to 1024. The experiment was performed for different relative values of S and F, $S/F = 2$ and 10. It can be observed that for $S/F = 2$, the Clus_mesh is not able to deliver significantly better performance than U_mesh. As S/F is increased to 10 the gains from Clus_mesh become apparent. The relative gain over U_mesh was observed to be up to 25% for a wide range of system sizes. We also observed that the gains are higher as the number of clusters is increased.

The impact of increasing cluster size while keeping the total number of processors in the system fixed, are presented in the bottom row of plots in Fig. 6. In each plot as the cluster size is increased the number of clusters in the system falls. This can lead to a fall [2] in the relative gains of Clus_mesh over U_mesh as depicted in the plots. However, we observed that for realistic cluster sizes and large system sizes as expected in the near future, Clus_mesh outperforms U_mesh.

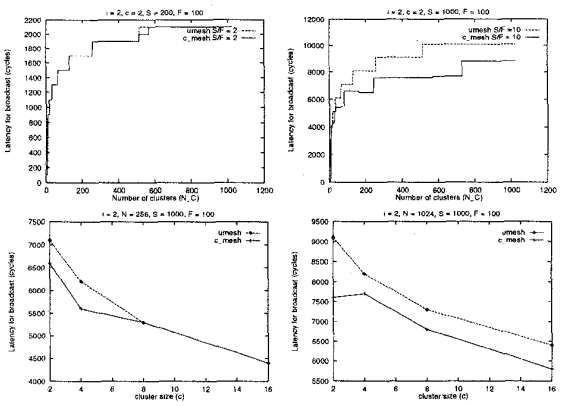

Figure 6: Comparing performance of Clus_mesh and U_mesh algorithms.

6 Conclusion

In this paper we have analyzed various cluster organizations for designing processor-cluster based multiprocessor systems. Using U_mesh broadcasting algorithm we have demonstrated the need for tightly coupled clusters to take advantage of higher integration available inside processor-clusters. With *differential* communication costs in the intra- and inter-cluster, we have also demonstrated that the U_mesh algorithm does not remain optimal on clustered systems. We have proposed a better algorithm, Clus_mesh, for broadcasting on clustered meshes. In future work we are studying the interplay of other collective communication algorithms like gather and complete exchange on clustered system design.

References

[1] D. Basak and D. K. Panda. Designing Clustered Multiprocessor Systems under Packaging and Technological Advancements. OSU-CISRC-11/95-TR51, 1995. To appear in IEEE TPDS.

[2] D. Basak and D. K. Panda. Designing Processor-cluster Based Systems: Interplay Between Cluster Organizations and Collective Communication Algorithms. OSU-CISRC-1/96-TR05, 1996.

[3] D. Basak, D. K. Panda and M. Banikazemi. Benefits of Processor Clustering in Designing Large Parallel Systems: When and How? In *Proc. of the IPPS*, pp. 286-290, 1996.

[4] Cray Reasearch Inc. *Cray T3D System Architecture Overview*, 1993.

[5] P. K. McKinley et al. Unicast-based Multicast Communication in Wormhole-routed Networks. *IEEE TPDS*, 5(12):1252–1265, Dec 1994.

[6] B. A. Nayfeh et al. The impact of shared-cache clustering in small-scale shared-memory multiprocessors. In *Proc. of the HPCA-2*, pp. 74–84, 1996.