J Supercomput
DOI 10.1007/s11227-009-0296-3

Performance analysis and optimization of MPI collective operations on multi-core clusters

Bibo Tu · Jianping Fan · Jianfeng Zhan · Xiaofang Zhao

Abstract Memory hierarchy on multi-core clusters has twofold characteristics: vertical memory hierarchy and horizontal memory hierarchy. This paper proposes new parallel computation model to unitedly abstract memory hierarchy on multi-core clusters in vertical and horizontal levels. Experimental results show that new model can predict communication costs for message passing on multi-core clusters more accurately than previous models, only incorporated vertical memory hierarchy. The new model provides the theoretical underpinning for the optimal design of MPI collective operations. Aimed at horizontal memory hierarchy, our methodology for optimizing collective operations on multi-core clusters focuses on hierarchical virtual topology and cache-aware intra-node communication, incorporated into existing collective algorithms in MPICH2. As a case study, multi-core aware broadcast algorithm has been implemented and evaluated. The results of performance evaluation show that the above methodology for optimizing collective operations on multi-core clusters is efficient.

Keywords Parallel computation model · Multi-core clusters · Memory hierarchy · MPI collective operations · Data tiling

B. Tu (✉) · J. Fan · J. Zhan · X. Zhao
Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
e-mail: tbb@ncic.ac.cn

J. Fan
e-mail: fan@ict.ac.cn

J. Zhan
e-mail: jfzhan@ncic.ac.cn

X. Zhao
e-mail: zhaoxf@ict.ac.cn

J. Fan
Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518067, China

Published online: 22 April 2009

1 Introduction

In fact, multi-core clusters have been the most popular platforms in parallel computing. In the new Top500 supercomputer list published in November 2007, about 87.6% of processors are multi-core processors and about 81.2% of supercomputers have used cluster architecture [1]. Intuitively, multi-core processors can speed up application performance by dividing the workload to different cores. However, compared with SMP or NUMA clusters, applications on multi-core clusters have not gotten optimal performance and scalability. Accordingly, it is crucial to have an in-depth understanding on characteristics of multi-core clusters and their effect on application behavior.

Initially, applications are likely to treat multi-core processors, also called Chip Multiprocessor (CMP), simply as conventional symmetric multiprocessors (SMPs). However, chip multiprocessors feature several interesting architectural attributes differently from SMPs. One such architectural feature is the design of multi-level cache hierarchies. The two broad strategies deployed in current multi-core processors exist in processors from Intel and AMD. Intel processors provide a shared L2 cache whereas AMD multi-core processors deploy HyperTransport links for quick data transfers. Also, these architectures employ efficient cache coherency protocols. However, irrespective of these different hierarchies, both the systems enable very fast sharing of data across the cores. Further, to scale the bandwidth available to the coherency traffic, the cores are connected either via multiple buses as in Intel or by 2-D mesh HyperTransport links. Apart from providing good data movement capabilities across the processing cores, the caching hierarchies are useful in reducing the pressure on the memory bandwidth [2].

Consequently, the above multi-core specific characteristics complicate memory hierarchies on multi-core clusters. Understanding their effect will potentially benefit the performance optimization of not only application algorithms, but also message passing. For many scientific applications, the cost of message passing (e.g. MPI communication middleware) greatly affects the overall execution time. Since MPI has still emerged as the primary programming paradigm for writing efficient parallel applications before the emergence of new programming paradigms suitable for multi-core/many-core platforms in the future, it is of practical significance to study performance analysis and optimization of MPI communication middleware on multi-core clusters at present. MPI programming paradigm provides a plethora of communication primitives with operations geared towards point-to-point and collective communications. When compared to the point-to-point performance, collective operation performance is often overlooked. However, profiling study [3, 4] showed that some applications spend more than 80% of a transfer time in collective operations. Thus, improving the performance of collective operations is the key to enable very high parallel speed-ups for parallel applications. Many parallel computation models are used to predict performance of collective operation on given parallel platform, and they are useful for the performance analysis of optimal collective operations. However, they are not suitable for new multi-core clusters. Concretely, complex memory hierarchy on multi-core clusters has not been modeled by previous models, so it cannot be used to accurately predict the performance of collective operations on multi-core clusters.

Memory hierarchy on multi-core clusters has twofold characteristics: vertical memory hierarchy and horizontal memory hierarchy. Vertical memory hierarchy has been modeled by previous work (e.g. memory logP, $\log_n$P and $\log_3$P, etc. [5–7]) to analyze middleware's effect on point-to-point communication with different message sizes and message strides. Horizontal memory hierarchy becomes more prominent due to the distinct performance among three levels of communication in a multi-core cluster: intra-CMP, inter-CMP and inter-node, which should adequately be considered. In this paper, we propose new parallel computation models, m$\log_n$P and its reduction 2$\log_{\{2,3\}}$P, to abstract memory hierarchy on multi-core clusters in vertical and horizontal levels, which are derived from $\log_n$P and $\log_3$P models. The results of performance evaluation show that new models can predict communication costs for message passing on multi-core clusters more accurately than $\log_3$P model. This is one of two main contributions in this paper, and it will provide the theoretical underpinning for the multi-core aware optimization of MPI collective operations.

The other contribution of this paper is that we give the methodology for optimizing MPI collective operations on multi-core clusters aimed at horizontal memory hierarchy. The optimal implementation of a collective for a given system mainly depends on virtual topology (e.g. flat-tree, binary tree, binomial tree, etc.) and message sizes. So exploiting hierarchy in different levels of communication (typically for three communication levels: intra-CMP, inter-CMP and inter-node, simply for two communication levels: intra-node and inter-node) on multi-core clusters to optimize collective operation performance, i.e. topology-aware algorithm for collective operations, will be essential. Furthermore, to select befitting segment sizes (data tiling approach) for intra-node collective communication can cater to cache hierarchy in multi-core processors (e.g. private L1 cache and shared L2 cache in Intel multi-core platform). The former emphasizes on the performance difference in different levels of communication, the latter tries to improve cache hit ratio in intra-node collective communication. Together, they construct the portable methodology over MPICH2 for optimizing collective operations on multi-core clusters, not modifying MPICH2 ADI layer, network drivers, even operating system. As a case study, we use 2$\log_{\{2,3\}}$P model to analyze the performance of multi-core aware broadcast algorithm and give its implementation and evaluation.

The rest of the paper is organized as follows. In Sect. 2, we summarize related work. In Sect. 3, we analyze twofold characteristics on memory hierarchy on multi-core clusters. In Sect. 4, we propose new parallel computation models suitable for multi-core clusters. Corresponding experiment results are given in Sect. 5. In Sect. 6, we give the methodology for optimizing collective operations on multi-core clusters. As a case study, multi-core aware broadcast algorithm and its performance evaluation are given in Sect. 7. Finally we conclude and point out future work directions in Sect. 8.

2 Related works

In recent years, many models have been proposed to predict and evaluate distributed communication performance and provide the theoretical underpinning for the optimal design of collective operations. For example, Thakur et al. [8] and Rabenseifner

et al. [9] use Hockney model to analyze the performance of different collective operation algorithms. Kielmann et al. [10] use PLogP model to find optimal algorithm and parameters for topology-aware collective operations incorporated in the MagPIe library. Park et al. [11] present an algorithm for constructing optimal broadcast trees for single-level networks based on terms similar to LogP. In this section, we give a brief survey of related work in two areas: parallel computation models and performance optimization for collective operations.

2.1 Parallel computation models

Previous work on parallel computation models may be classified as hardware-parameterized models and software-parameterized models.

Hardware-parameterized models ignore the increasing effect of memory communication (memory hierarchy) on communication cost with different message sizes and message strides. LogP model [12] approximates memory communication in parallel systems with a fixed overhead parameter (o), the reciprocal of the bandwidth between application and network buffers. LogGP [13] is an extension of the LogP model that additionally allows for large messages by introducing the gap per byte parameter, G, and ignores the effect of data distribution. There are also other varieties of LogP model, such as PLogP [14], LoPC [15], LoGPC [16], LoGPS [17], and so on, that have more or less ignored the important effect of memory communication.

With the ever-increasing speed gap between the CPU and memory systems, incorporating memory hierarchy into computational models has become unavoidable. Memory logP model [5] uses middleware parameters to describe the cost effect of message size and distribution when message passes through memory. Inspired by the memory logP model, $\log_n$P model and its reduction $\log_3$P model [6, 7] combine hierarchical memory performance with estimates of network communication cost, and accurately predict the performance of distributed point-to-point communication.

With the emergency of multi-core processors, memory hierarchy on multi-core clusters becomes more complicated. Apart from memory hierarchy in the vertical level, memory hierarchy in the horizontal level becomes more prominent. The above software-parameterized models, such as memory logP, $\log_n$P and $\log_3$P, only focus on vertical memory hierarchy. In this paper, the new models m$\log_n$P and 2$\log_{\{2,3\}}$P provide a general abstract for vertical and horizontal memory hierarchy so as to accurately predict and evaluate the performance of message passing on multi-core clusters.

2.2 Performance optimization for collective operations

Performance optimization of MPI collective operations has been an active area of research in recent years and some early works are discussed as follows.

Some work on collective communication focused on developing optimized algorithms for particular architectures, such as hypercube, mesh, torus or fat tree, with an emphasis on minimizing link contention, node contention, or the distance between communicating nodes [8, 9, 18, 19]. Otherwise, automatically tuned collective communication algorithms under different conditions (message size, number of processes) have been developed [20, 21].

Another work on collective communication focused on exploiting hierarchy in parallel computer networks to optimize collective operation performance. One is to optimize MPI collective communication for WAN distributed environments (e.g. MagPIe) [10, 22], the goal is to minimize communication over slow wide-area links at the expense of more communication over faster local-area connections. The other is to optimize MPI collective communication for LAN environments through modifying MPICH ADI layer, target for SMP clusters, e.g. MPI-StarT [23].

Utilizing shared memory for implementing collective communication has been a well-studied problem in the past. Some authors propose using remote memory operations across the cluster and shared memory within the cluster to develop efficient collective operations for clusters of SMPs [24, 25]. With the help of operating system and network drivers, minimizing the cost of memory copy to improve intra-node communication has also been applied in intra-node communication on more-core clusters recently [26].

Inspired by MagPIe and MPI-StarT, we adopt hierarchical virtual topology to exploit horizontal memory hierarchy on multi-core clusters to optimize collective operation performance. Furthermore, data tiling approach is applied to improve the intra-node communication performance for better utilization of high-efficient cache in multi-core processors, not modifying MPICH2 ADI layer, network drivers, even operating system.

3 Memory hierarchy on multi-core clusters

The ever-increasing speed gap between CPUs and memory systems and current multi-core technology complicate memory hierarchy. Memory hierarchy on multi-core clusters takes on twofold characteristics in the vertical and horizontal levels as shown in Fig. 1 as an example of Intel Dual-Core Xeon.

3.1 Vertical memory hierarchy

For message passing in a cluster system, distributed point-to-point communication requires moving data from the source process' local memory to the target process' local memory. Sends and receives are explicit communications accomplished using implicit communication mechanisms provided in middleware and operating system. Communication cost consists of the sum of memory and network communication times. Memory communication is the transmission of data to/from user buffer, kernel buffer, from/to the local network buffer. Network communication is data movement between source and target network buffers. For example, an MPI_Send() of a strided message describes a point-to-point transfer explicitly. To ensure that packed data is actually sent across the network, MPI middleware performs a series of implicit communications to complete the transfer (i.e., packing strided data at the source and unpacking data by stride at the target). Some transmissions occur in user space, others via the operating system in kernel space [6]. The message path of the above point-to-point communication shows memory hierarchy in a vertical level as shown in Fig. 1.

Improvements in memory speed will continue to lag behind improvements in processor and network interconnect technologies. Given current technological trends,

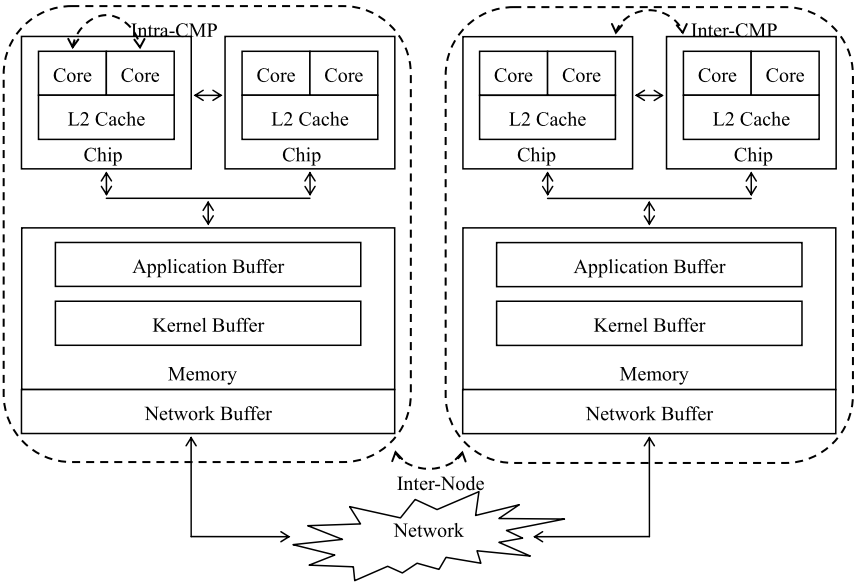


Fig. 1 Memory hierarchy on multi-core clusters

memory communication cost (e.g. MPI middleware cost) will increase in proportion to overall communication cost. So it is indispensable to incorporate middleware costs into analytical models for message passing.

Log_nP model and its reduction $\log_3\text{P}$ model can accurately analyze communication performance with memory hierarchy in the vertical level. Log_nP model is formalized as follows using five parameters [6, 7]:

l: the effective latency, defined as the length of time the processor is engaged in the transmission or reception of a strided or non-contiguous message over and above the cost of a non-strided or contiguous transfer. The system-dependent l cost is a function of the message data size (s) under a variable stride or distribution (d). We denote this function as $f(s, d) = l$, where variable s corresponds to a series of discrete message sizes in bytes, variable d corresponds to a series of discrete distributions or stride distances in bytes between array elements, and function f is the additional time for transmission in microseconds over and above the non-strided or contiguous cost for variable message data size s and distribution or stride d.

o: the overhead, defined as the length of time the processor is engaged in the transmission or reception of a non-strided or contiguous message. The system-dependent o cost is a function of the message data size (s) under a fixed unit stride or distribution ($d = 1$ array element). We denote this function as $f(s, d) = f(s, 1) = o$, where variable s corresponds to a series of discrete message sizes in bytes, variable $d = 1$ array element corresponds to the unit stride or distribution distance in bytes between adjacent array elements, and function f is the time for transmission in microseconds for variable message data size s and distribution or stride $d = 1$ element. This average, unavoidable overhead typically represents the best case for data transfer on a

target system. This cost is bounded below by the data size divided by the hardware bandwidth.

g: the gap, defined as the minimum time interval between consecutive message receptions at a processor. Without resource contention, assuming this parameter has no impact on communication cost, effectively using $o = g$.

n: the number of implicit transfers along the data transfer path between two endpoints. Endpoints can be as simple as two distinct local memory arrays or as complex as a remote transfer between source and target memories across a network.

P: the number of processes or processors.

In a word, vertical memory hierarchy behaves as follows: parameter n denotes the number of implicit communication along the message path and the memory communication cost varies with message size (s) and message distribution (d, strides). $\mathrm{Log}_n\mathrm{P}$ denotes the function $f(s, d)$ to express the effective latency (l) and overhead (o) in message transmission, i.e. $l = f(s, d)$ and $o = f(s, 1)$. Therefore, $\log_n\mathrm{P}$ estimates point-to-point communication cost as:

$$T = \sum_{i=0}^{n-1} \{o_i + l_i\} = \sum_{i=0}^{n-1} \big\{ f(s,1)_i + f(s,d)_i \big\}. \tag{1}$$

For simplicity, $\log_3\mathrm{P}$ model assumes $n = 3$ points of implicit communication. Equation (1) reduces to:

$$\begin{aligned} T &= \sum_{i=0}^{2} \big\{ f(s,1)_i + f(s,d)_i \big\} \\ &= \big\{ f(s,1)_0 + f(s,d)_0 \big\} + \big\{ f(s,1)_1 + f(s,d)_1 \big\} + \big\{ f(s,1)_2 + f(s,d)_2 \big\} \end{aligned}$$

i.e.

$$T = \{o_0 + l_0\} + \{o_1 + l_1\} + \{o_2 + l_2\}. \tag{2}$$

Equation (2) describes three implicit communication points along the message path of distributed point-to-point communication as follows: (0) Middleware communication from user buffer to the network interface buffer, this includes the effect of hierarchical memory. (1) Communication across the network. (2) Middleware communication from the network interface buffer to user buffer, this includes the effect of hierarchical memory. Since packets are contiguous and fixed size for point $i = 1$, l_1 is assumed to be zero. More precisely and simply, the costs at point $i = 0$ and $i = 2$ are regarded as equal. Assuming:

middleware overhead $= o_{\mathrm{mw}} = \{o_0 + o_2\}$,
middleware latency $= l_{\mathrm{mw}} = \{l_0 + l_2\}$,
network overhead $= o_{\mathrm{net}} = o_1$.

Thus, the $\log_3\mathrm{P}$ model can be expressed semantically as:

$$T = \{o_0 + o_2\} + \{l_0 + l_2\} + \{o_1 + l_1\} = o_{\mathrm{mw}} + l_{\mathrm{mw}} + o_{\mathrm{net}}. \tag{3}$$

In a word, $\log_n P$ and $\log_3 P$ models disclose the intrinsic characteristic of distributed point-to-point communication: software-parameterized memory communication (middleware costs) and hardware-parameterized network communication, i.e. memory hierarchy in the vertical level. So they are very suitable for conventional cluster computing platforms, including SMP and NUMA clusters.

3.2 Horizontal memory hierarchy

Apart from vertical memory hierarchy, multi-core clusters take on prominent characteristic of horizontal memory hierarchy. Horizontal memory hierarchy is mainly brought by distinct performance among three levels of communication in a multi-core cluster: intra-CMP, inter-CMP and inter-node communication as shown in Fig. 1. The communication between two processors on the same chip is referred to as intra-CMP communication in this paper. The communication across chips but within a node is referred to as inter-CMP communication. Intra-CMP and inter-CMP communications in the same node are also referred to as intra-node communication. And the communication between two processors on different nodes is referred to as inter-node communication.

Due to multi-level cache hierarchies, CMPs offer unique capabilities that are fundamentally different from SMPs [27]:

- The inter-core communication performance (bandwidth and latency) on a CMP can be many times better than that is typical for a SMP.
- Cache hierarchy organization (Intel processors provide a shared L2 cache whereas AMD multi-cores deploy HyperTransport links for quick data transfers) on multi-core processors makes inter-core memory access far faster than that is typical for a SMP, only with shared main memory.

So it does, horizontal memory hierarchy on multi-core clusters becomes more prominent and brings the following features differently from conventional SMP clusters, which can also be seen from recent studies [28, 29]:

Prominent intra-node communication performance: For multi-core clusters, the performance (bandwidth and latency) of intra-node communication is many times better than inter-node, and intra-CMP has the best performance because data can be shared through L2 cache as the example of Intel Quad-core Xeon. However, for SMP clusters, the performance of intra-node communication only with shared main memory cannot absolutely excel that of inter-node communication, because of memory copy cost of intra-node communication and some zero-copy and high performance schemes of inter-node communication, e.g. InfiniBand and RDMA-enabled interconnects [30]. Otherwise, poor scalability of memory interconnection limits the number of intra-node CPUs, which degrades the importance of intra-node communications on SMP clusters.

Equal chance between intra-node and inter-node communications: For SMP clusters, the most of message distribution is distributed point-to-point communication (inter-node communication). However, on average about 50% messages in a multi-core cluster are transferred through the intra-node communication [28]. With the

ever-increased number of computing cores in a CMP, especially for future many-core processors, the chance of intra-node communication will be more dominant.

Consequently, horizontal memory hierarchy on multi-core clusters should be enough considered to incorporate into previous parallel computing models, only modeling vertical memory hierarchy on SMP clusters.

4 New models suitable for multi-core clusters

Vertical memory hierarchy on multi-core clusters implies overlooked memory communication costs of point-to-point message passing varied with different message sizes and message strides, while horizontal memory hierarchy implies distinct performance of point-to-point message passing among different communication levels on multi-core clusters. This paper proposes new models mlog_nP and $2\text{log}_{\{2,3\}}\text{P}$ to unitedly abstract memory hierarchy in vertical and horizontal levels to accurately analyze communication performance on multi-core clusters, derived from log_nP and log_3P models.

4.1 mlog_nP model

Extending log_nP model, mlog_nP model adds a new parameter 'm' to define different communication levels, which depicts the characteristic of horizontal memory hierarchy in Sect. 3.2. While parameter 'n' is redefined to denote the number of implicit transfers along the message path in different communication levels, which still depicts the characteristic of vertical memory hierarchy in Sect. 3.1. mlog_nP model is formalized as follows using six parameters:

m: the number of different communication levels. Typically for three communication levels: intra-CMP, inter-CMP and inter-node. Non-uniform memory hierarchy and large-scale cascading network may bring more communication levels.
l: the effective latency, defined as the length of time the processor is engaged in the transmission or reception of a strided or non-contiguous message over and above the cost of a non-strided or contiguous transfer.
o: the overhead, defined as the length of time the processor is engaged in the transmission or reception of a non-strided or contiguous message.
g: the gap, defined as the minimum time interval between consecutive message receptions at a processor. Without resource contention, assuming this parameter has no impact on communication cost, effectively using $o = g$.
n: the number of implicit transfers along the message path in different communication levels.
P: the number of processes or processors.

According to (1), mlog_nP estimates point-to-point communication costs in different communication levels on multi-core clusters as:

$$T_i = \sum_{j=0}^{n_i-1} \left\{ o_j^i + l_j^i \right\} = \sum_{j=0}^{n_i-1} \left\{ f(s,1)_j^i + f(s,d)_j^i \right\} \quad \left(i \in \{0,1,\ldots,m-1\} \right) \tag{4}$$

where T_i ($i \in \{0, 1, \ldots, m-1\}$) denotes point-to-point communication cost in different communication levels, n_i denotes the number of implicit transfers along the message path in the ith communication level, and o_j^i, l_j^i denote the overhead and latency of the jth message transfer in the ith communication level. Compared with (1), (4) shows that the number of implicit transfers along the message path in different communication levels is different, but also their latencies and overheads are also different.

4.2 $2\log_{\{2,3\}}$P model

For simplicity and practical use, $2\log_{\{2,3\}}$P model is proposed to simplify $m\log_n$P model, which is suitable for current COTS multi-core clusters. In $2\log_{\{2,3\}}$P model, we assume two communication levels ($m = 2$): intra-node and inter-node. In intra-node communications, we ignore the inter-CMP communication, which has two reasons:

- Because of ever-enlarging number of intra-CMP computing cores (especially for future many-core CPUs) and the limited number of intra-node CMPs in real systems (awkward performance and poor scalability of intra-node CMP interconnection), the chance of inter-CMP communications will be small, which can also be seen in Sect. 3.2.
- Performance parameters for inter-CMP communication are hard to measure, which will be not beneficial to the simplicity and practical use of new model.

In $2\log_{\{2,3\}}$P model, the intra-node communication has $n = 2$ points of implicit communication without network transfer, while inter-node communication has $n = 3$ points of implicit communication. Because of distinct memory communication costs (middleware costs) in different communication levels, the point-to-point communication costs in $2\log_{\{2,3\}}$P model will be estimated over again.

For $m = 2$ and $n = 2$ or 3, (4) is reformulated as:

$$\begin{cases} T_0 = o_{\text{mw}}^0 + l_{\text{mw}}^0 (o_{\text{net}} = 0, n = 2), & \text{for intra-node communication,} \\ T_1 = o_{\text{mw}}^1 + l_{\text{mw}}^1 + o_{\text{mw}}^1 (n = 3), & \text{for inter-node communication.} \end{cases} \tag{5}$$

We use the function $f(s, d)$ to rewrite (5) as:

$$\begin{cases} T_0 = 2f^0(s,1)_0 + 2f^0(s,d)_0, & \text{for intra-node communication,} \\ T_1 = 2f^1(s,1)_0 + 2f^1(s,d)_0 + f^1(s,1)_1, & \text{for inter-node communication.} \end{cases} \tag{6}$$

For the same message size and stride, the middleware communication costs ($f^0(s,1)_0$ and $f^1(s,1)_0$, $f^0(s,d)_0$ and $f^1(s,d)_0$) between intra-node and inter-node communications are distinct. Differently from $\log_3$P model expressed by (3), $2\log_{\{2,3\}}$P model expressed by (5) and (6) discloses not only vertical memory hierarchy, but also horizontal memory hierarchy on multi-core clusters.

For simplicity, thereafter $o_{\text{mw}} + l_{\text{mw}}$ and $o'_{\text{mw}} + l'_{\text{mw}} + o'_{\text{net}}$ are respectively used to denote the intra-node and inter-node point-to-point communication costs in (5), assuming $o_{\text{mw}} = o_{\text{mw}}^0$, $l_{\text{mw}} = l_{\text{mw}}^0$, $o'_{\text{mw}} = o_{\text{mw}}^1$, $l'_{\text{mw}} = l_{\text{mw}}^1$ and $o'_{\text{net}} = o_{\text{mw}}^1$.

5 Experimental results

Since the importance of memory communication as to vertical memory hierarchy in point-to-point communication has been verified in previous work [6, 7], our objective will verify that the effect of horizontal memory hierarchy on point-to-point communication on multi-core clusters should adequately be considered. Furthermore, we will verify $2\log_{\{2,3\}}\mathrm{P}$ model is more suitable for multi-core clusters than $\log_3\mathrm{P}$ model.

Our evaluation system consists of eight nodes connected by Gigabit Ethernet. Each node is equipped with two sets of quad-core 2.0 GHz Intel Xeon processor, i.e. eight computing cores per node. Four cores on the same chip share 8M L2 cache. The operating system is Linux 2.6, MPI version is mpich2-1.0.6p1. In all our experiments, we use *sched_affinity* system call to ensure the binding of process with processor. Without loss of generality, we only select strided message to measure performance, because strided message must be packed into a contiguous buffer (like non-strided message), and then sent across memory or network to its destination. In the following tests, message size is classified as 1 Kbyte, 4 Kbyte and 16 Kbyte and message stride is classified as 8 byte, 64 byte and 512 byte. Messages greater than 16 Kbyte are not considered, because handshakes or acknowledgements are required for long messages. A strided message is expressed as "*K*S", as shown in the x-axis in the following figures. For example, "4K8S" means that message size is 4 Kbyte and message stride is 8 byte.

5.1 Middleware costs of point-to-point communication

We use blocking MPI_Send and MPI_Recv calls to get roundtrip times (RTT) so as to measure memory communication costs, which is suggested by related papers [6, 14, 31]. Firstly, we use PRTT method [31] to get the network overhead (o'_{net}) of inter-node communication. Subsequently, we begin by obtaining the roundtrip times of contiguous and non-contiguous message transfers for intra-node and inter-node communications. According to (5) and (6), there are the following equations to get individual middleware and network costs:

$RTT(s) = 2o_{\mathrm{mw}}$ (contiguous transfers for intra-node point-to-point communication);
$RTT(s,d) = 2(o_{\mathrm{mw}} + l_{\mathrm{mw}})$ (non-contiguous transfers for intra-node point-to-point communication);
$RTT'(s) = 2(o'_{\mathrm{mw}} + o'_{\mathrm{net}})$ (contiguous transfers for inter-node point-to-point communication);
$RTT'(s,d) = 2(o'_{\mathrm{mw}} + l'_{\mathrm{mw}} + o'_{\mathrm{net}})$ (non-contiguous transfers for inter-node point-to-point communication).

Figure 2 shows middleware costs of point-to-point communication on our evaluation system. The left part in Fig. 2 shows the performance of the intra-node communication, and the right part shows the performance of the inter-node communication. From the figure, we can see middleware overhead (o_{mw} or o'_{mw}) and middleware latency (l_{mw} or l'_{mw}) increase with message sizes and message strides, and middleware latency dominates memory communication costs with increased message sizes and

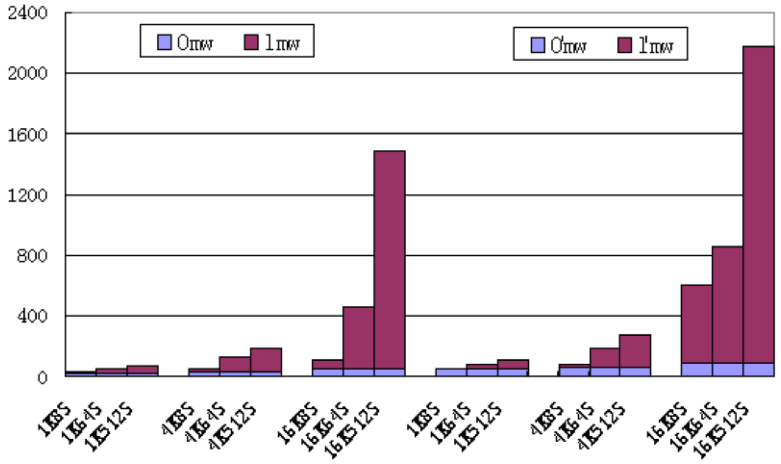


Fig. 2 Middleware costs of point-to-point communication on a multi-core cluster. The x-axis is message size and stride and y-axis is time in microseconds

message strides. For the same message, middleware overhead of intra-node communication is on average 49.2% less than that of inter-node communication. The difference of middleware latencies between intra-node and inter-node communications is small when message size and stride is small, but the largest difference of middleware latencies is approaching 87.9% with increased message sizes and strides. The whole middleware cost (middleware overhead and latency) of intra-node communication is on average 41% less than that of inter-node communication. Thus, it can be seen that the difference of middleware costs between intra-node and inter-node communications on the multi-core cluster is prominent, and it is indispensable to incorporate horizontal memory hierarchy into parallel computation models.

5.2 Cost prediction of collective communication

In this section, we use $2\log_{\{2,3\}}$P model to analyze common collective communication algorithms, linear broadcast algorithm and binomial tree broadcast algorithm, compared with $\log_3$P model.

The linear broadcast algorithm is based on point-to-point communication and used to broadcast the message from the first process in a group to all other processes, in which $(P-1)$ individual consecutive MPI_Sends are used at the source/root process to transfer data to each remaining process, where P is the number of processes.

Using $\log_3$P, the cost of the linear broadcast algorithm is $P \times (o'_{\mathrm{mw}}/2 + l'_{\mathrm{mw}}/2) + o'_{\mathrm{net}}$, where $P \times (o'_{\mathrm{mw}}/2 + l'_{\mathrm{mw}}/2)$ is the middleware overhead and middleware latency occurring at the source process for sending data to other $(P-1)$ processes, and o'_{net} is the network overhead.

Using $2\log_{\{2,3\}}$P model, the cost of the linear broadcast algorithm is divided into intra-node and inter-node, expressed as:

$$(P/n-1) \times (o_{\mathrm{mw}}/2 + l_{\mathrm{mw}}/2) + \big(P - (P/n-1) - 1\big) \times \big(o'_{\mathrm{mw}}/2 + l'_{\mathrm{mw}}/2\big) + o'_{\mathrm{net}},$$

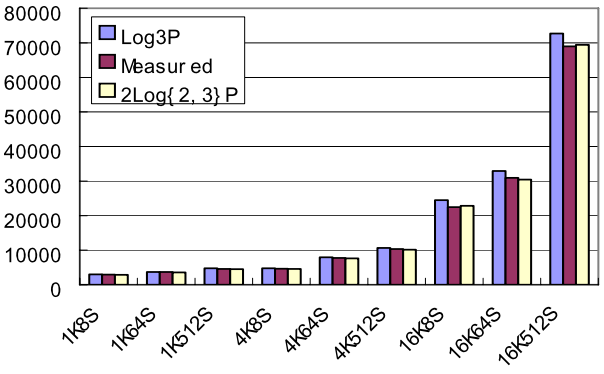


Fig. 3 Cost prediction of the linear broadcast ($8 \times 8 = 64$ processes). The x-axis is message size and stride and y-axis is time in microseconds

where n is the number of nodes, assuming all intra-node communication are started at first.

Figure 3 shows costs predicted by $\log_3$P and $2\log_{\{2,3\}}$P models and measured cost for the linear broadcast algorithm, respectively. The size of the linear broadcast algorithm in Fig. 3 is 64 processes (8 processes per node).

From Fig. 3 we can see that $2\log_{\{2,3\}}$P prediction is more accurate than $\log_3$P for all measured message sizes and message strides. For the data points measured, maximum relative error of $\log_3$P is 15.32% and average relative error is 5.64%; while the maximum relative error of $2\log_{\{2,3\}}$P is 3.61% and average relative error is 1.59%.

The binomial tree broadcast algorithm is commonly used in MPICH. In the first step, the root sends data to process (root $+ P/2$). This process and the root then act as new roots within their own subtrees and recursively continue this algorithm (P denotes the number of processes). This communication takes a total of $\log_2 P$ steps (the height of the tree).

Therefore, the times taken by the binomial tree broadcast algorithm are $(o'_{\text{mw}} + l'_{\text{mw}} + o'_{\text{net}}) \times \log_2 P$ (using $\log_3$P) and $(o_{\text{mw}} + l_{\text{mw}}) \times n + (o'_{\text{mw}} + l'_{\text{mw}} + o'_{\text{net}}) \times (\log_2 P - n)$ (using $2\log_{\{2,3\}}$P, n denotes the number of intra-node transfers along the longest distance to a leaf node in $\log_2 P$ steps and its value is determined by the location of the root).

Figure 4 shows costs predicted by $\log_3$P and $2\log_{\{2,3\}}$P models and measured cost for the binomial tree broadcast algorithm, respectively. The size of the binomial tree broadcast algorithm in Fig. 4 is 64 processes (8 processes per node).

From Fig. 4 we can see that for small message sizes and small strides, $\log_3$P prediction is accurate, but for large message sizes and larger strides, $2\log_{\{2,3\}}$P prediction is more accurate than $\log_3$P. For the data points measured, maximum relative error of $\log_3$P is 10.53% and average relative error is 4.68%; while the maximum relative error of $2\log_{\{2,3\}}$P is 4.36% and average relative error is 2.03%.

Otherwise, from Figs. 3 and 4 we can see that $\log_3$P prediction consistently overestimates the communication time, because it overlooks high-efficient intra-node communication. In more cases, $2\log_{\{2,3\}}$P model underestimates the communication time, because it uses intra-CMP communication cost to predict inter-CMP communication cost due to ignored inter-CMP communication level in the model. With the increased message sizes and message strides, the error of $\log_3$P prediction increases,

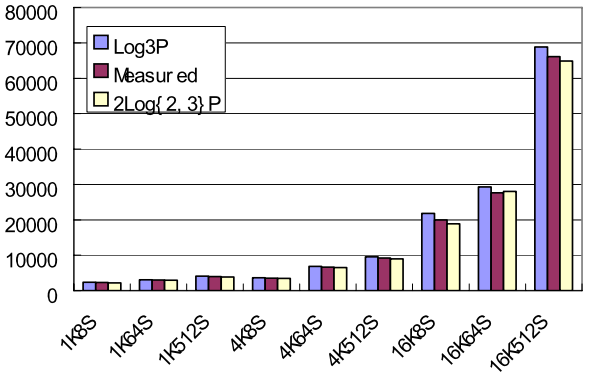


Fig. 4 Cost prediction of the binomial tree broadcast ($8 \times 8 = 64$ processes). The x-axis is message size and stride and y-axis is time in microseconds

while $2\log_{\{2,3\}}$P prediction does not follow this trend. In a word, the results in Figs. 3 and 4 show that $2\log_{\{2,3\}}$P model is more suitable for communication evaluation on multi-core clusters than $\log_3$P model.

6 Methodology for optimizing collective operations on multi-core clusters

For accurate analysis of communication performance on multi-core clusters, it is indispensable to incorporate horizontal memory hierarchy to parallel computation model. Furthermore, it is also the key to exploiting horizontal memory hierarchy to optimize collective operations on multi-core clusters. Concretely, the performance difference among different communication levels on multi-core clusters and cache utilization in intra-node collective communication mainly brought by horizontal memory hierarchy must be adequately considered for optimal MPI collective operations.

The optimal implementation of a collective for a given system mainly depends on virtual topology (e.g. flat-tree, binary tree, binomial tree, etc.) and message sizes. MPI collective operations can be classified as either one-to-many/many-to-one (asymmetrical virtual topology, the root process exists) or many-to-many (symmetrical virtual topology, all processes are equal peers) operations. For example, Broadcast, Reduce, Scatter(v), and Gather(v) follow one-to-many communication pattern, while Barrier, Alltoall, Allreduce, and Allgather(v) employ many-to-many communication. Generalized version of the one-to-many/many-to-one type of collectives can be expressed as (i) receive data from preceding node(s), (ii) process data (optional), (iii) send data to succeeding node(s). The data flow for this type of algorithms is unidirectional. Based on some parameters, including message sizes, number of processes and the location of the root node (where applicable), different virtual topologies can be used to determine the preceding and succeeding nodes in the algorithm.

To optimize collective operations on multi-core clusters, our methodology focuses on the performance difference among different communication levels on multi-core clusters and cache utilization in intra-node collective communication, which is presented as follows:

Hierarchical virtual topology: Our algorithms are designed for a two-layer hierarchy: inter-node communication and intra-node communication, respectively. Concretely, asymmetrical algorithms perform two macro steps. In a one-to-many operation, the root first sends its data to head processes of other nodes (the first process in every node), and second, the head process locally sends to other processes in the same node. In the many-to-one case, processes in the same node first send to their head process and then all head processes in different nodes send to the root. Symmetrical algorithms perform three macro steps. First, processes in every node send to their head process. Second, all head processes perform an all-to-all exchange, and third, every head process sends to their other processes.

Cache–aware intra-node collective communication: In order to improve the performance of intra-node collective communication, an easy software technique related to message size is used to adapt well cache-shared multi-core system: data tiling approach for improving utilization of private L1 cache and shared L2 cache. Data tiling is a cache-blocking technique and tries to allow data to stay in the shared cache while being processed by data loops. By reducing the unnecessary cache traffic (fetching and evicting the same data throughout the loops), a better cache hit rate is achieved. Possible candidates for the cache blocking technique include large data sets that are operated-on many times. By going through the entire data set and performing one operation, followed by another pass for the second operation, and so on, if the entire data set does not fit into the cache, the first elements in the cache will be evicted to fit the last elements in. Then, on subsequent iterations through the loop, loading new data will cause the older data to be evicted, possibly creating a domino effect where the entire data set needs to be loaded for each pass through the loop. By sub-dividing the large data set into smaller blocks and running all of the operations on each block before moving on to the next block, there is a likelihood that the entire block will remain in cache through all the operations.

Some previous optimization algorithms involved with virtual topology and message sizes have successfully been implemented in MPICH2, while our methodology is constructed on top of them and portable. In fact, existing MPICH2 algorithms are still reserved in intra-node and inter-node collective communications, respectively. But our methodology is only to reorder its data flow.

7 Case study: the optimization of broadcast operation

As a case study, multi-core aware broadcast algorithm is proposed. In MPI_Bcast in MPICH2, a so-called root process sends a message to all other processes. This operation is the simplest case of an asymmetrical, one-to-many algorithm. MPI_Bcast uses multiple algorithms based on the message size. For small messages, binomial tree algorithms are widely employed. Large message broadcast involves a more complex approach of doing a scatter followed by an allgather. According to our portable optimization methodology over MPICH2, we need not utterly rewrite the above MPI_Bcast algorithm, only reorder its data flow.

7.1 Multi-core aware broadcast algorithm

Pseudo-code for multi-core aware broadcast algorithm, called MCC_MPI_Bcast, is presented in Fig. 5. First, all processes in the same node are grouped into a new communicator, thus, every node exists a communicator (e.g. Node_Comm[i] in Fig. 5), respectively. All head processes (process with rank 0) in every node communicator are grouped into a new communicator (e.g. Head_Comm in Fig. 5) for inter-node collective communication. Second, all head processes start inter-node broadcast communication. Thus, the root sends message to all nodes. At last, in every node, message is segmented to start intra-node broadcast communication in a loop.

In intra-node collective communication, segment size is determined by data tiling approach for improving L1 cache and L2 cache utilization. In MCC_MPI_Bcast algorithm, for small message, to split message into segments that can easily fit L1 cache (fitL1CacheSize) may hold segment in high performance L1 cache for a long time in one-to-many communication. For large message, message will be split into some segments that can fit into shared L2 cache (fitL2CacheSize) and get operated on many times. Thus, both L1 and L2 cache hit rates get improved.

```
int MCC_MPI_Bcast (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
{
/* define all node communicators and head processes communicator */
MPI_Comm Node_Comm[MaxNodeNum], Head_Comm;

/* all processes in the same node are grouped into new communicator, i.e. every node has a communicator, respectively. */
for (i=0; i < Node_Num; i++)
    MPI_Comm_split(comm, isSameNode(), 0, Node_Comm[i]);

/* all head processes (first process with rank 0 in every node) are grouped into a new communicator */
MPI_Comm_create(comm., isFirstRank(myrank), Head_Comm);

/*Step 1: inter-node collective communication */
if (isFirstRank(myrank))
    MPI_Bcast(buffer, count, datatype, root, Head_Comm);

/* Step 2: intra-node collective communication */
for (i=0; i < Node_Num; i++) {
    if (isLocalComm(myrank, Node_Comm[i]) {
/* in a loop broadcast call, message is divided into several segments in order to fit in cache */
        if (messagesize < fitL1CacheSize) /* no segmented */
            MPI_Bcast(buffer, count, datatype, 0, Node_Comm[i]);
        else if (messagesize < fitL2CacheSize) { /* to fit in L1 Cache */
            for (j = 0; j < messagesize/fitL1CacheSize; j++)
                MPI_Bcast(buffer, fitL1CacheSize, datatype, 0, Node_Comm[i]);
        }
        else { /* to fit in L2 Cache */
            for (j = 0; j < messagesize/fitL2CacheSize; j++)
                MPI_Bcast(buffer, fitL2CacheSize, datatype, 0, Node_Comm[i]);
        }
    }
}}
```

Fig. 5 Pseudo-code for multi-core aware broadcast algorithm (MCC_MPI_Bcast)

7.2 Model analysis

We use $2\log_{\{2,3\}}\mathrm{P}$ model to analyze the cost of broadcast operation for its optimal design. Without loss of generality, we take the example of commonly used binomial tree algorithm for broadcast in MPICH2. In the first step, the root sends data to process (root + $P/2$). This process and the root then act as new roots within their own subtrees and recursively continue this algorithm (P denotes the number of processes). This communication takes a total of $\log_2 P$ steps (assuming P is the power of 2). The amount of data communicated by a process at any step is m. Therefore, the time taken by this algorithm is:

$$T_{\text{tree}} = T(m) \times \log_2 P. \tag{7}$$

In (7), $T(m)$ is the time of point-to-point communication with m bytes message shown in (5) in Sect. 4.2.

In $\log_2 P$ steps, some are intra-node point-to-point communications, others are inter-node point-to-point communications. In order to reduce the cost (T_{tree}) of broadcast operation, one possible approach is to decrease the number of inter-node communications. MCC_MPI_Bcast algorithm with hierarchical virtual topology in Sect. 7.1 has the least number of inter-node communications. Assuming that the number of cluster nodes is expressed by N, hierarchical virtual topology brings better performance and the time is:

$$T'_{\text{tree}} = T_{\text{intra}}(m) \times \log_2(P/N) + T_{\text{inter}}(m) \times \log_2 N. \tag{8}$$

In (8), $T_{\text{intra}}(m)$ denotes the time of intra-node point-to-point communication with m bytes message, and T_{inter} denotes the time of inter-node point-to-point communication with m bytes message.

Furthermore, in T'_{tree}, the cost of intra-node collective communication may be reduced through data tiling approach shown in Fig. 5. To split message m to some befitting segments to loop operations may improve cache hit rates. Thus, the value of $T_{\text{intra}}(m) \times \log_2(P/N)$ will be reduced. Assuming that the number of loop operations is expressed by n, the cost of intra-node collective communication is:

$$\sum_{i=1}^{n} \big(T_{\text{intra}}(m/n) \times \log_2(P/N)\big),$$

and then the cost of optimal broadcast operation is as follows.

$$T''_{\text{tree}} = \sum_{i=1}^{n} \big(T_{\text{intra}}(m/n) \times \log_2(P/N)\big) + T_{\text{inter}}(m) \times \log_2 N. \tag{9}$$

To make ensure that $T''_{\text{tree}} < T'_{\text{tree}} < T_{\text{tree}}$, this model analysis will provide the support for finding optimal broadcast algorithm on multi-core clusters, and the performance evaluation in next subsection will verify it.

7.3 Performance evaluation

We evaluated the performance of cache-aware intra-node broadcast communication and multi-core aware broadcast algorithm, compared with MPI_Bcast in MPICH2. In our evaluation system in Sect. 5, every cluster node has 8 computing cores, and four cores on the same chip share 8 M L2 cache and each core has 32 KB private L1 cache.

In the test, message sizes are selected from 8 KB to 8 MB, and segment sizes are selected from 4 KB to 2 MB, in accordance with cache capacity of Intel quad-core Xeon processor. Figures 6 and 7 show performance results of cache-aware intra-node broadcast communication (8 processes). In order to get optimized performance, segment size and the number of broadcast loops need the tradeoff. From Fig. 6, we can see that: when segment size is 8 KB, the performance of intra-node broadcast communication is the best for small message (less than 512 KB); when segment size is 1 MB, the performance of intra-node broadcast communication is the best for large message (more than 1 MB). Consequently, for our evaluation system, when parameters fitL1CacheSize = 8 KB and fitL2CacheSize = 1 MB, the performance of cache-aware intra-node broadcast communication is optimal.

Hierarchical virtual topology and cache-aware intra-node collective communication improve the performance of broadcast operation (MCC_MPI_Bcast) on multi-core clusters, compared with MPI_Bcast in MPICH2, which is seen from Fig. 8. Broadcast operation with 64 processes (8 × 8, 8 processes every node, totally 8 nodes) has been tested with different message sizes. The root process is selected at random. From the figure, we can see that the performance difference between MCC_MPI_Bcast and MPI_Bcast is increasing with message sizes, and the percentage of performance improvement is from 16.8 to 24.7%.

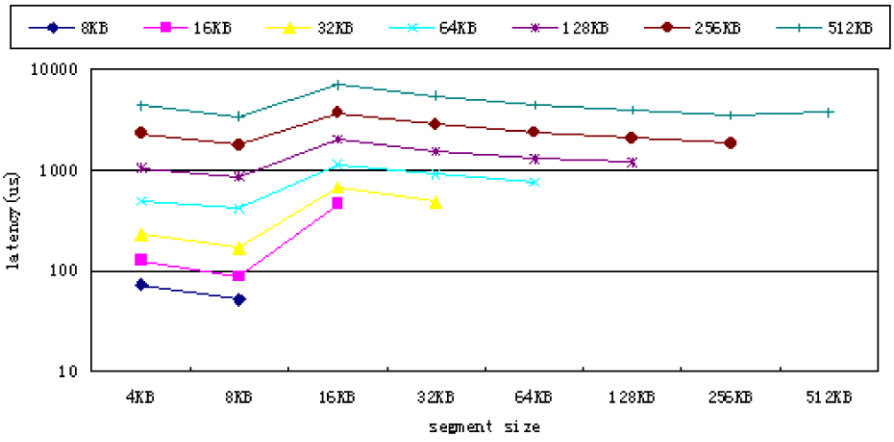


Fig. 6 The costs of cache-aware intra-node broadcast communication (8 processes) in microseconds when message size is less than 512 KB

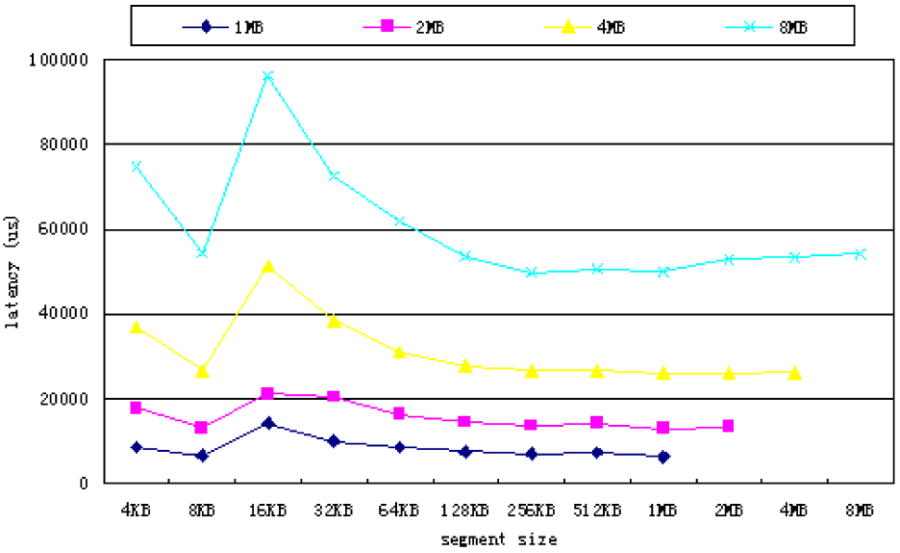
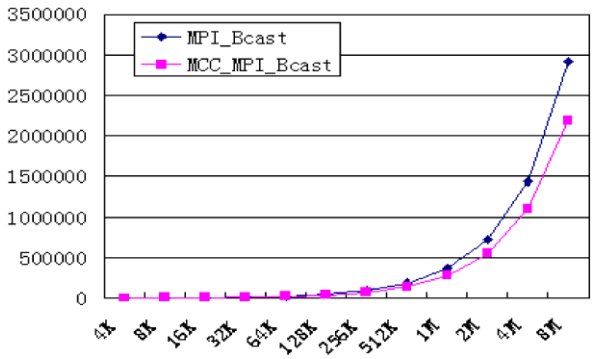


Fig. 7 The costs of cache-aware intra-node broadcast communication (8 processes) in microseconds when message size is more than 1 MB

Fig. 8 The costs of multi-core aware broadcast algorithm (64 processes, 8 processes every node) compared with MPI_Bcast in MPICH2 in microseconds



8 Conclusions and future work

Differently from conventional SMP clusters, horizontal memory hierarchy on multi-core clusters behaves more prominently, interweaved with vertical memory hierarchy. Vertical memory hierarchy implies overlooked memory communication costs of point-to-point message passing varied with different message sizes and message strides, while horizontal memory hierarchy implies distinct performance of point-to-point message passing among different communication levels on multi-core clusters. Derived from $\log_n P$ and $\log_3 P$ models, new parallel computation models $m\log_n P$ and $2\log_{\{2,3\}} P$ are proposed to unitedly abstract vertical and horizontal memory hierarchy for accurate performance analysis of MPI collective operations on multi-core clusters.

It is also the key to exploiting horizontal memory hierarchy to optimize collective operations on multi-core clusters. With the help of $2\log_{\{2,3\}}$P model, we propose the corresponding portable optimization methodology constructed over MPICH2, including hierarchical virtual topology and cache-aware intra-node collective communication. Conforming to it, optimized broadcast algorithm has been implemented and it showed a good performance.

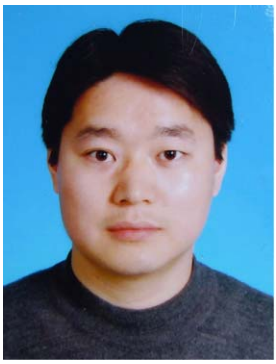
For our future work, we have two research directions. (i) Though our models are general and applicable to current multi-core clusters, multi-core processors also complicate hierarchical parallelization, our models may require extension (e.g. OpenMP, multi-threaded programming model [32]) in the future. (ii) Using $2\log_{\{2,3\}}$P model, we continue to carry out more in-depth design and evaluation of optimal algorithms for other important collective operations.

Acknowledgements We would like to thank the reviewers for their insightful comments. This work was supported by the National High Technology Research and Development Program of China (863 Program) under grant No. 2006AA01A102.

References

1. TOP500 Team, TOP500 Report for November 2007, http://www.top500.org
2. Mamidala AR, Kumar R, De D, Panda DK (2008) MPI collectives on modern multicore clusters: performance optimizations and communication characteristics. In: 8th IEEE international conference on cluster computing and the grid (CCGRID '08)
3. Rabenseifner R (1999) Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512. In: Proceedings of the message passing interface developer's and user's conference, pp 77–85
4. Pjesivac-Grbovic J, Angskun T, Bosilca G et al (2005) Performance analysis of MPI collective operations. In: Proceedings of the 19th IEEE international parallel and distributed processing symposium (IPDPS'05)
5. Cameron KW, Sun X-H (2003) Quantifying locality effect in data access delay: memory logP. In: Proceedings of IEEE international parallel and distributed processing symposium (IPDPs 2003), Nice, France
6. Cameron KW, Ge R (2004) Predicting and evaluating distributed communication performance. In: Proceedings of the 2004 ACM/IEEE supercomputing conference
7. Cameron KW, Ge R, Sun X-H (2007) $\log_n$P and $\log_3$P: accurate analytical models of point-to-point communication in distributed systems. IEEE Trans Comput 56(3):314–327
8. Thakur R, Gropp W (2003) Improving the performance of collective operations in MPICH. In: Dongarra J, Laforenza D, Orlando S (eds) Recent advances in parallel virtual machine and message passing interface. Lecture notes in computer science, vol 2840. Springer, Berlin, pp 257–267
9. Rabenseifner R, Traff JL (2004) More efficient reduction algorithms for non-power-of-two number of processors in message-passing parallel systems. In: Proceedings of EuroPVM/MPI. Lecture notes in computer science. Springer, Berlin
10. Kielmann T, Hofman RFH, Bal HE, Plaat A, Bhoedjang RAF (1999) MagPIe: MPI's collective communication operations for clustered wide area systems. In: Proceedings of the seventh ACM SIGPLAN symposium on principles and practice of parallel programming. ACM Press, New York, pp 131–140
11. Park J-YL, Choi H-A, Nupairoj N, Ni LM (1996) Construction of optimal multicast trees based on the parameterized communication model. In: Proc int conference on parallel processing (ICPP), vol I, pp 180–187
12. Culler DE, Karp R, Patterson DA, Sahay A, Santos E, Schauser K, Subramonian R, von Eicken T (1996) LogP: a practical model of parallel computation. Commun ACM 39:78–85
13. Alexandrov A, Ionescu MF, Schauser K, Scheiman C (1995) LogGP: incorporating long messages into the LogP model. In: Proceedings of seventh annual symposium on parallel algorithms and architecture, Santa Barbara, CA, pp 95–105

14. Kielmann T, Bal HE (2000) Fast measurement of LogP parameters for message passing platforms. In: Proceedings of the 15 IPDPS 2000 workshops on parallel and distributed processing, pp 1176–1183
15. Frank MI, Agarwal A, Vernon MK (1997) LoPC: modeling contention in parallel algorithms. In: Proceedings of sixth symposium on principles and practice of parallel programming, Las Vegas, NV, pp 276–287
16. Moritz CA, Frank MI (1998) LoGPC: modeling network contention in message-passing programs. In: Proceedings of SIGMETRICS '98, Madison, WI, pp 254–263
17. Ino F, Fujimoto N, Hagihara K (2001) LogGPS: a parallel computational model for synchronization analysis. In: Proceedings of PPoPP'01, Snowbird, Utah, pp 133–142
18. Barnett M, Littlefield R, Payne D, van de Geijn R (1993) Global combine on mesh architectures with wormhole routing. In: Proceedings of the 7th international parallel processing symposium, April
19. Scott D (1991) Efficient all-to-all communication patterns in hypercube and mesh topologies. In: Proceedings of the 6th distributed memory computing conference, pp 398–403
20. Vadhiyar SS, Fagg GE, Dongarra J (1999) Automatically tuned collective communications. In: Proceedings of SC99: high performance networking and computing, November
21. Faraj A, Yuan X (2005) Automatic generation and tuning of MPI collective communication routines. In: Proceedings of the 19th annual international conference on supercomputing, pp 393–402
22. Karonis NT, de Supinski BR, Foster I, Gropp W et al (2000) Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In: Proceedings of the 14th international parallel and distributed processing symposium (IPDPS'2000), pp 377–384
23. Husbands P, Hoe JC (1998) MPI-StarT: delivering network performance to numerical applications. In: Proceedings of the 1998 ACM/IEEE SC98 conference (SC'98)
24. Tipparaju V, Nieplocha J, Panda DK (2003) Fast collective operations using shared and remote memory access protocols on clusters. In: International parallel and distributed processing symposium
25. Wu M-S, Kendall RA, Wright K (2005) Optimizing collective communications on SMP clusters. In: ICPP' 2005
26. Chai L, Hartono A, Panda DK (2006) Designing high performance and scalable MPI intra-node communication support for clusters. In: The IEEE international conference on cluster computing
27. Asanovic K, Bodik R, Catanzaro BC et al (2006) The landscape of parallel computing research: a view from Berkeley. Electrical Engineering and Computer Sciences, University of California at Berkeley. Technical Report No: UCB/EECS-2006-183, p 12
28. Chai L, Gao Q, Panda DK (2007) Understanding the impact of multi-core architecture in cluster computing: a case study with intel dual-core system. In: Seventh IEEE international symposium on cluster computing and the grid (CCGrid'07), pp 471–478
29. Alam SR, Barrett RF, Kuehn JA, Roth PC, Vetter JS (2006) Characterization of scientific workloads on systems with multi-core processors. In: International symposium on workload characterization
30. Liu J, Wu J, Panda DK (2004) High performance RDMA-based MPI implementation over InfiniBand. Int J Parallel Program
31. Hoefler T, Lichei A, Rehm W (2007) Low-overhead LogGP parameter assessment for modern interconnection networks. In: Proceedings of IEEE international parallel and distributed processing symposium (IPDPS'2007)
32. Curtis-Maury M, Ding X, Antonopoulos CD, Nikolopoulos DS (2005) An evaluation of OpenMP on current and emerging multithreaded/multicore processors. In: IWOMP



Bibo Tu received the B.Sc. degree in Computer Science and Technology from Wuhan University of Science and Technology, China, in 1999. He also received the M.Sc. degree in Computer Software and Theory from Huazhong University of Science and Technology, China, in 2002. He received the Ph.D. degree in Computer Architecture from Institute of Computing Technology, Chinese Academy of Sciences, in 2008. Since 2002, he has been in a research staff at Institute of Computing Technology, Chinese Academy of Sciences. His research interests include high performance computing and parallel processing.

Jianping Fan received the Ph.D. degree in Computer Software and Theory from Institute of Software, Chinese Academy of Sciences, in 1990. Since 1990, he has worked at Institute of Computing Technology, Chinese Academy of Sciences, as Associate Professor and Professor. Now, he works at Shenzhen Institute of advanced technology, Chinese Academy of Sciences, as Professor and the Dean. His research interests include high performance computing, cluster and grid computing and parallel processing.

Jianfeng Zhan received the Ph.D. degree in Computer Software and Theory from Institute of Software, Chinese Academy of Sciences, in 2002. Since 2002, he has worked at Institute of Computing Technology, Chinese Academy of Sciences, as Associate Professor. His research interests include high performance computing, parallel and distributed systems.

Xiaofang Zhao received the Ph.D. degree in computer architecture from Institute of Computing Technology, Chinese Academy of Sciences, in 1999. Since 1999, he has worked at Institute of Computing Technology, Chinese Academy of Sciences, as Associate Professor and Professor. His research interests include high performance computing, information security and computer architecture.