Low Power Optimization for MPI Collective Operations

Yong Dong, Juan Chen, Xuejun Yang, Canqun Yang, Lin Peng
School of Computer, National University of Defense Technology, China
{yongdong, juanchen, xjyang, canqun, linpeng}@nudt.edu.cn

Abstract

*DVFS-available (Dynamic Voltage/Frequency Scaling) processors make it possible for a system to reduce the energy consumption by scaling down the frequency/voltage of the processors in high performance computing. For MPI collective operations, network communication time occupies the most of the whole time. Scaling down CPU voltage/frequency in non-critical path can effectively reduce energy consumption. This paper proposes **L**ow-**P**ower **M**PI_**G**ather algorithm (**LPMG**) and **L**ow-**P**ower **M**PI_**S**catter algorithm (**LPMS**) and extend them to almost all the MPI collective operations. We evaluate the effectiveness of our low-power MPI collective operation algorithm using Intel MPI benchmark IMB on 128-processor cluster system connected by a 1000Mbps Ethernet. Experimental results show that different MPI collective operations can achieve different energy saving. With 128 processes, average 45.9% and 55.7% energy savings can be reached through **LPMG** and **LPMS**, respectively. But MPI_Alltoall only gets 2.2% energy saving.*

Key words: MPI, collective operation, low power, dynamic voltage/frequency scaling.

1. Introduction

Although HPC (high-performance computing) tends to get higher peak performance, the development of HPC makes the problem of high power/energy consumption more and more serious [1] . For example, Earth Simulator requires about 12 megawatts of peak power, and Petaflops systems require 100 megawatts of power [2] . It is observed that on current trend, the power density of microprocessor will reach that of a nuclear reactor by the year 2010. Too high energy consumption brings about many problems, such as low reliability, bad stability, high costs, etc. Reducing the energy of parallel programs has become more and more important.

MPI [3] is one of the most popular parallel program interfaces, among which there exist large amount of MPI collective operations, such as MPI_Scatter, MPI_Gather, MPI_Reduce and so on. During these MPI collective operations, communication phases sometimes become the performance bottlenecks which make CPU is not on the critical path. In particular, when the number of nodes is up to a large one or the network is slow, the performance bottleneck is more obvious. Take MPI_Barrier as an example, when all 1024 nodes have to do the synchronization at one barrier, the difference of communication time among 1024 nodes is big which causes a large amount of computing idle time. So dynamically reducing CPU voltage/frequency during these communication phases in MPI collective operations can save power/energy consumption with a little performance loss.

In this paper, we designed and implemented low-power MPI_Gather algorithm (**LPMG**) and low-power MPI_Scatter algorithm (**LPMS**) by utilizing dynamic voltage/frequency scaling (DVFS). LPMG and LPMS reduce the processors' voltage/frequency whose loads are lighter than others through inserting dynamic voltage/frequency scaling points to reduce the power consumption.

The main contributions of this article are:

- Proposing a **L**ow-**P**ower **M**PI_**G**ather algorithm **LPMG**.
- Proposing a **L**ow-**P**ower **M**PI_**S**catter algorithm **LPMS**.
- Extending **LPMG** and **LPMS** to all the MPI collective operations and evaluating them.

We evaluated these two algorithms using Intel MPI Benchmarks-IMB [4] on a cluster system with 128 processor cores which are connected by 1000Mbps Ethernet network. Experimental results show that our algorithms can effectively reduce processors' energy consumption with a little performance loss.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 describes two low-power MPI collective operation algorithms. Section 4 shows the experimental platform and

978-0-7695-3398-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ICYCS.2008.500

experimental results. Section 5 concludes this paper and discusses the future work.

2. Related Works

DVFS is one of the most important low-power techniques [5-10] . Various DVFS techniques have been widely studied at operating system, micro-architecture and compiler levels. DVFS algorithms in MPI programs have absorbed more and more researchers' attention. This kind of algorithms is usually implemented in the MPI library. Kappiah et al. [11] developed a system--Jitter, in which the slack time among different nodes is mined and the voltage and frequency of nodes are scaled down for energy savings. Freeh et al. [12] focused on low-power, high-performance clusters, divided programs into phases and chose a suitable frequency for each phase. Lim et al. [13] presented a MPI runtime system that dynamically reduces CPU performance during communication phases in MPI programs. It dynamically identifies such phases and selects the CPU voltage/frequency in order to minimize energy-delay product. This article is also concerned optimizing the energy consumption of MPI programs, but it focuses on MPI collective operations which include a large amount of communication phases. Lim also considered utilizing communication phases to reduce voltage/frequency for energy savings. But he didn't design the detailed algorithms for each MPI collective operations.

3. Low-Power MPI Collective Operation Algorithms

MPI has become a popular parallel programming interface standard, which is widely used in many parallel scientific applications. Reducing the energy of parallel application needs focusing more work on MPI operations because these operations occupies a large part of the execution time of the whole program. The computing time of each process has the big difference which provides us the opportunities of energy optimization.

In this section, we start with a motivation example to describe our basic idea and then introduce our low-power MPI collective operation algorithms in detail.

3.1. Motivation

MPI collective operations can be divided into the following four kinds:

1) All-to-all: all the processors participate in MPI collective operation and get the final results from the operation, such as MPI_Allgather, MPI_Allreduce, MPI_Alltoall and MPI_Reduce_scatter.
2) All-to-one: all the processors participate in operation and only one gets final result, such as MPI_Reduce and MPI_Gather.
3) One-to-all: one processor sends data to all others, such as MPI_Bcast and MPI_Scatter.
4) Others: MPI_Barrier.

Except 1), the computing time of each process has big difference which makes it possible for us to scale down voltage/frequency for energy savings. How much energy saving can be achieved not only depends on the speed of network, but also depends on MPI collective operations algorithms themselves.

In this paper, we use MPICH2 [15] as the prototype to design and implement our algorithms. Take MPI_Scatter and MPI_Gather as examples, they both use binomial tree to execute the collective operations, in which every processes act as leaf nodes, middle nodes or root node. Data is sent from root node to middle and leaf nodes in MPI_Scatter and is send from leaf and middle nodes to root node in MPI_Gather.

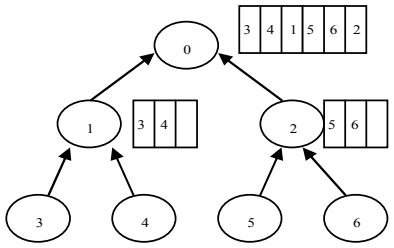


Figure 1. An example of MPI_Gather

Figure 1 shows an example of MPI_Gather which has seven nodes, among which node 3-6 are leaf nodes and node 0 is the root. In this operation, node 3-6 send data to node 1-2. Once node 1-2 finish sending data to the top-level nodes, they are idle to wait other nodes finishing operations or continuing other operations. If a forced barrier is added at the end of MPI_Gather, there is some slack time after node 1-2 sending data. Perhaps there is some performance loss because of a forced barrier operation, but DVFS from slack time can bring about a significant energy saving.

As the number of nodes increases, such energy saving is more and more. That is because more nodes make the height of binomial tree increase and the distance between leaf nodes and root nodes increase. Then leaf nodes need to wait longer time for root node receiving data. Besides, the speed of cluster network decides the amount of energy saving.

Then, we analyze the energy saving in terms of CMOS circuit power formulas. The processor's power

can be described in formula (1) in terms of CMOS circuit power. There are three parts in this formula: dynamic power, short circuit current power and leakage power, respectively.

$$P = \alpha_0 \cdot C \cdot V^2 \cdot f + t_s \cdot \alpha_0^{'} \cdot V \cdot I_{short} \cdot f + V \cdot I_{leak} \quad (1)$$

where α_0 is activity factor, C is capacitance, V is supply voltage, f is clock frequency, t_s is short-circuiting execution time, $\alpha_0^{'}$ is parameter, I_{short} is short circuit current and I_{leak} is leakage circuit current.

The dynamic power is always the main part in all these three parts in the 90nm processor technology. With the development of processor technology, the short circuit power is smaller and smaller, which is produced in switch of the circuit status by the current.

DVFS is one of the most important low-power techniques. The change of frequency comes from that of the voltage, and the relationship of voltage and frequency can be defined as:

$$f \propto \frac{(V - V_t)^{\gamma}}{V} \quad (2)$$

where $\gamma = 2$ and the frequency f is directly proportional to the supply voltage V. So,

$$P \propto f^3 \quad (3)$$

$$P_{\max} = k \cdot f_{\max}^3 \quad (4)$$

In this paper, we define β as the proportion of the power in the idle state to that in the active state.

$$P_{idle} = \beta \cdot P_{\max} \quad (5)$$

where P_{idle} represents the power in the idle state and $P_{\max}$ is the power in the active state. Although β can be vary in different system, we define $\beta = 80\%$ for simplicity and experience. Furthermore, we define η as the proportion of power in the shut-down state to that in the active state:

$$P_{shut-down} = \eta \cdot P_{\max} \quad (6)$$

where $P_{shut-down}$ is the power in the shut-down state and $\eta = 15\%$.

3.2. Low-Power MPI_Gather algorithms

In MPI_Gather operation, data is sent from all nodes to one node. The algorithm of MPI_Gather can be described as follows. Every node in MPI_Comm build a binomial tree according to the logical order. No.0 is the root of the tree; other nodes are No.1, 2, 3…, as breadth-first order. Both short and long messages are transported in this tree. For each node except the leaf nodes, a temporary buffer needs to be allocated to store the incoming message.

In MPI_Gather operation, when a node completes its data transfer to its parent node, it is in "idle" until the MPI_Gather operation finishes. From the perspectives of energy saving, we can shut down such kind of nodes or scale down the voltage/frequency of them. When MPI_Gather finishes, the shut-down nodes can be reactivated again. Obviously, it maybe delay the start time of the later operations and brings about the performance penalty. Our methods are different from Lim's [13] , which scales down the frequency of CPU in MPI communication phase but doesn't refer to the detailed algorithm implementation for each MPI collective operation. Figure 2 shows the Low-power MPI_Gather algorithm **LPMG**.

Algorithm: ***L**ow-**P**ower **M**PI_**G**ather Algorithm (**LPMG**)*
INPUT: MPI_Gather operation;
OUTPUT: Low-power MPI_Gather operation;

```
V=V_max;
get rank;
if (this is a leaf node) {
    send data to parent;
if (receive ack from parent)
      low power status;
}
if (this is neither leaf node nor root){
    get data from child;
    send data to parent;
    if (receive ack from parent)
      low power status;
}
if (this is root) {
    receive data from child;
    activate(all_nodes);
}
end.
```

Figure 2. Low-**P**ower **M**PI_**G**ather algorithm (**LPMG**)

In this algorithm, once each node finishes its data transfer, it is set to low-power status. Firstly, we construct the binomial tree according to breadth-first order; make the root node as No.0. Secondly, transfer data from leaves to the root. If a node is leaf, then it sends data to its parent. If it is neither leaf node nor the root, it prepares the temp buffer and receives data from child to the temp buffer and stores its data to it. When data is ready, the node sends its data to the parent. When the root node receives all data and reorders it, the gather operation finishes. Every node in this tree has no other operation once it receives the acknowledgement from the parent node. In this algorithm, we scale down the processor to the lowest voltage/frequency (called shut-down state) when it finishes its work in MPI_Gather operation. When MPI_Gather finishes, all the shut-down processors are reactivated.

In our algorithm, the performance loss comes from the following two reasons:

1) We shut down all processors after they send data to the parent node, and reactivate them

after the whole collective operation finishes. So that all possible asynchronous operations are forced to be synchronous. This perhaps makes the whole execution time longer.

2) During reactivating the shut-down processors, there is reactivate time penalty.

There are two methods to reduce such performance loss.

1) We shut down part of the processors instead of all the processors. For example, only shutting down the leaf nodes. This makes those nodes near to the root node keep the active status.
2) We take the pre-reactivate policy to ahead scale up the voltage/frequency of processors to avoid reactivate time penalty.

3.3. Low-Power MPI_Scatter algorithm

Algorithm: ***L**ow-**P**ower **M**PI_**S**catter Algorithm (**LPMS**)*
INPUT: MPI_Scatter operation;
OUTPUT: Low-power MPI_Scatter operation;
$V=V_{max}$;
get rank;
if (this is a leaf node) {
    recv data from parent;
    all leaf node barrier;
    activate(all nodes);
}
if (this is neither leaf node nor root) {
    get data from parent;
    send data to child;
    if(receive ack from child)
        low power status;
}
if (this is root) {
    send data to child;
    low power status;
}
end.

Figure 3. Low-**P**ower **M**PI_**S**catter algorithm (**LPMS**)

MPI_Scatter is a reverse operation of MPI_Gather. MPI_Scatter transfers data from one node to all the other nodes. MPICH2 also use the binomial tree to implement MPI_Scatter. Every process acts as leaf node, middle node or the root node. In MPI_Scatter, the root node sends a message to all others. Nodes other than leaf nodes are allocated a temporary buffer to store the incoming message. If the root is not rank 0, the send buffer is reordered in order of relative ranks, so as to make sure that every non-root node gets the right data.

Just like MPI_Gather, MPI_Scatter can also make use of the same idea. If a non-leaf node sends its data to child, it is set to low-power status with the lowest voltage/frequency until it is reactivated. Also there is some performance penalty. We chose the second way to alleviate such voltage switch time penalty which is the same as MPI_Gather. Figure 3 show the detailed algorithm description.

Compared to MPI_Gather, MPI_Scatter has its own characteristic. In MPI_Scatter, the leaf nodes get data from their parents, they need a barrier before they are activated.

3.4. Other MPI collective operations

Section 3.2 and 3.3 give low-power MPI_Gather and low-power MPI_Scatter algorithms, respectively. This section discusses other MPI collective operations.

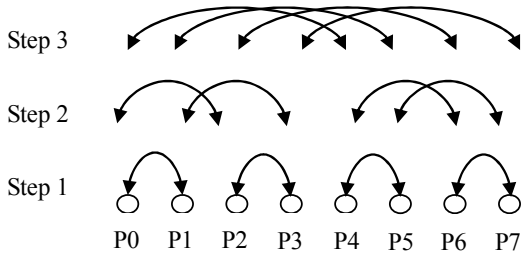


Figure 4. Recursive doubling for MPI_Allgather

Energy saving is obvious in all-to-one or one-to-all operation. But for all-to-all operations, energy saving is very small. Take MPI_Allgather as example. MPI_Allgather is an operation in which data is gathered on all processes and the result is sent back to all processes, which is different from MPI_Gather. So each node in MPI_Allgather participates in more times of communications than MPI_Gather. Figure 4 shows one recursive doubling algorithm for MPI_Allgather (from [14]).

Later experimental results also show energy saving of all-to-all operations is less than other operations.

4. Experimental Evaluations

4.1. Experimental environments

In our experiments, we use a 16-node Intel® Xeon® 5150 cluster connected by a 1000Mbps Ethernet. Each node has two CPUs and each CPU has four cores. Each core has 2.66GHz peak clock frequency. All nodes run the Linux kernel 2.6.18. All programs are compiled with MPICH2-1.0.7 [15] .

Intel MPI Benchmark (IMB) [4] is an easy-to-use set of MPI benchmark, and it evaluates the performance of various computing platforms or MPI implementations and can check many MPI communication patterns. Each program in IMB suite

only includes one MPI collective operation. Each MPI collective operation in IMB runs for many times with different sizes of messages and different number of processes. For example, when MPI_Scatter is running, we set the number of process 32, and IMB_MPI will execute MPI_Scatter with 2, 4, 8, 16 and 32 processes, in turn. The size of message will change from 0 to 4MB by default and we get tens of results.

Programs in IMB refer to all kinds of MPI collective operations described in the above content. In this article, we choose 13 programs to evaluate the effectiveness of our low-power MPI collective operation algorithms. We insert voltage scaling instruction in the proper place of each program and let it run with different number of processes (1-128 processes). The message length of each MPI collective operation is fixed on 4MB and the number of iterations is ten.

In the experiments, we also consider the reactivate time and energy overhead as (7)-(8) show.

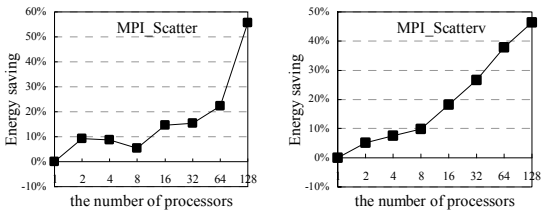
$$Time_{reactivate} = 1ms \quad (7)$$

$$Energy_{reactivate} = 10mJ \quad (8)$$

4.2. Experimental results

Figure 5 shows the experiment results of our low-power MPI collective operation algorithms. From this figure, we can see that when the number of processes is 128, low-power MPI_Gather can get 45.9% energy saving and low-power MPI_Scatter can get 55.7% energy saving compared to the original MPI_Scatter and MPI_Gather. With the number of processes increases, the energy saving is more and more. This is because when there are more processors, it need more time to construct the communication tree and finish the communicate operations, and there are more slack time within the processor computing which brings more opportunities to scale down the voltage/frequency of processors.

Compared to MPI_Gather and MPI_Scatter, MPI_Allreduce, MPI_Alltoall and MPI_Alltoallv only get 7.1%, 2.2% and 3.1% energy saving with 128 processes, respectively. That is because all nodes in these two operations send and receive data and there is no more slack time among them.



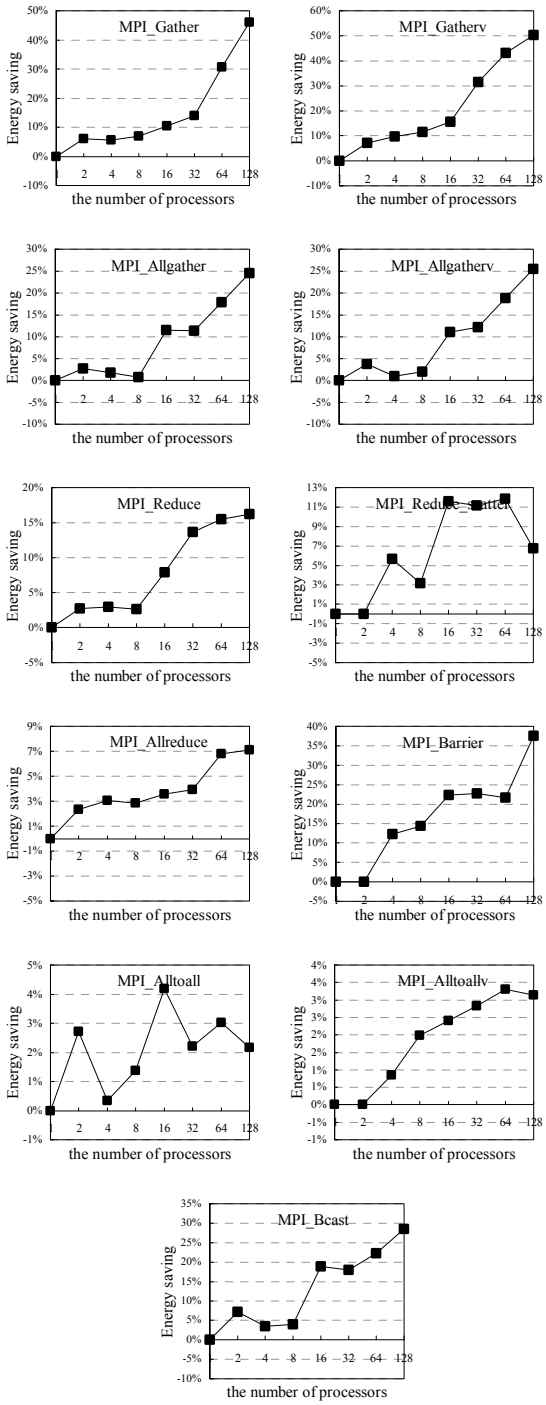


Figure 5. Energy optimization results for low-power MPI collective operations

Because each program only includes MPI collective operation, our low-power algorithm produces a little performance loss within 5%.

5. Conclusion and future work

In this article, we design and implement **L**ow-**P**ower **M**PI_**G**ather algorithm (**LPMG**) and **L**ow-**P**ower **M**PI_**S**catter algorithm (**LPMS**). Furthermore, we extend it to almost all the MPI collective operations. Our low-power MPI collective operation algorithms are evaluated on the cluster system with 128 processor cores using Intel MPI Benchmark. Experiment results show that (i) more processors will bring about more energy saving; (ii) with 128 processes, 45.9% energy saving can be achieved through **LPMG** and 55.7% energy saving is achieved through **LPMS**; (iii) MPI_Alltoall get the minimum energy saving which is only 2.2% with 128 processes.

In the future work, we will discuss other low-power MPI algorithm and integrate them into the implementation of MPI.

Acknowledgement

This work was supported in part by a grant from Nature Science Foundation of China (NSFC) project under Contract 60621003 and Nature Science Foundation of China (NSFC) project under Contract 60703074.

References

[1]Wu-chun Feng. The Importance of Being Low Power in High Performance Computing. August 2005. http://www.ctwatch.org/quarterly/articles/2005/08/the-importance-of-being-low-power-in-high-performance-computing/.

[2]D. H. Bailey. 21st Century High-End Computing. In invited Talk Application. In *Algorithms and Architectures workshop for BlueGene/L*.2002.

[3]The Message Passing Interface (MPI) standard. http://www-unix.mcs.anl.gov/mpi/.

[4]Intel MPI Benchmarks-Users Guide and Methodology Description. http://www3.intel.com/cd/software/products/asmo-na/eng/219848.htm.

[5]J. Pouwelse, K. Langendoen, H. Sips. Dynamic voltage scaling on a low power microprocessor.In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. p.251 – 259. July 2001.

[6]Fen Xie, Margaret Martonosi, Sharad Malik. Compile-Time Dynamic Voltage Scaling Settings: Opportunities and Limits.In *Proceedings of ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation (PLDI-03)*. San Diego, California, USA. ACM Press. p.49-62. June 9-11 2003.

[7]Chung-Hsing Hsu, Ulrich Kremer. Compiler-Directed Dynamic Voltage Scaling for Memory-Bound Applications. Technical Report. DCS-TR-498. Rutgers University, August 2002.

[8]Gang Qu. What is the Limit of Energy Saving by Dynamic Voltage Scaling?In *IEEE/ACM International Conference on Computer Aided Design*. p.560-563. November 2001.

[9]T. Burd and R. Brodersen . Design issues for dynamic voltage scaling.In *Proceedings of 2000 International Symposium on Low Power Electronics and Design (ISLPED' 00)*. July 2000.

[10]H. Saputra, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, J. Hu, C. -H Hsu, U. Kremer. Energy-conscious compilation based on voltage scaling.In *ACM SIGPLAN Joint Conference on Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems (LCTES/SCOPES' 02)*. June 2002.

[11]Nandini Kappiah, Vincent W. Freeh, David K. Lowenthal. Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs.In *Proceedings of the ACM/IEEE SC 2005 Conference on High Performance Networking and Computing (SC'05)*.2005.

[12]Vincent W. Freeh, Feng Pan, Nandini Kappiah, David K. Lowenthal. Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster.In *Proceedings of 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'05)*. p.164-173. June 15-17 2005.

[13]Min Yeol Lim, Vincent W. Freeh, David K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs.In *the Proceedings of SC'06*.2006.

[14]Rajeev Thakur, Rolf Rabenseifner, William Gropp. Optimization of Collective Communication Operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1). p. 49-66.2005.

[15]MPICH2-1.0.7. http://www.mcs.anl.gov/mpi/mpich2.