Comput Sci Res Dev (2014) 29:161–169
DOI 10.1007/s00450-013-0238-4

SPECIAL ISSUE PAPER

Performance estimation of high performance computing systems with Energy Efficient Ethernet technology

Shinobu Miwa · Sho Aita · Hiroshi Nakamura

Published online: 25 July 2013

Abstract Energy Efficient Ethernet (EEE) is an Ethernet standard for lowering power consumption in commodity network devices. When the load of a link is low, EEE allows the link to turn into a low power mode and therefore can significantly save the power consumption of a network device. EEE is expected to be adopted in high performance computing (HPC) systems a few years later, but the performance impact caused by EEE-enabled in HPC systems is still unknown. To encourage HPC system developers to adopt the EEE technology, it is required for the performance estimation of the non-existing HPC systems that would utilize the EEE technology. This paper presents the performance estimation method for EEE-supported HPC systems, which utilizes a novel performance model we propose. The model with a few profiles for HPC applications allows us to anticipate the performance of the systems not yet realized. The evaluation results show that the proposed model has the significant accuracy and that EEE is still promising for HPC applications.

Keywords Energy Efficient Ethernet · High performance computing · Performance estimation

S. Miwa (✉) · S. Aita · H. Nakamura
The University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan
e-mail: miwa@hal.ipc.i.u-tokyo.ac.jp

S. Aita
e-mail: aita@hal.ipc.i.u-tokyo.ac.jp

H. Nakamura
e-mail: nakamura@hal.ipc.i.u-tokyo.ac.jp

1 Introduction

Interconnection networks consume large amount of power in modern high performance computing (HPC) systems. The main contributors to the power consumption of interconnection networks are physical layer devices (PHYs) because they work whenever the network cable is connected. For high-performance and high-availability, the HPC systems require wide bandwidth and a lot of redundant network paths on interconnection networks. For example, TOFU interconnection network used in K computer has 6.25 GB/s in each link and 10 links in each node [1, 2]. Since a lot of PHYs are required for the HPC systems, their power consumption is not negligible. Some researcher reported that the power consumption of interconnection networks reaches 33 % of the total system power [12].

Energy Efficient Ethernet (EEE) is an Ethernet standard for lowering power consumption in commodity network devices. EEE allows a PHY to turn into a low power mode, which is called Low Power Idle (LPI), when the network load of the related link is low. During LPI mode, the PHY can save the power consumption by up to 70 % [5]. EEE is standardized as IEEE 802.3az in 2010 [9], and network devices compliant with IEEE 802.3az are gradually increased [6, 8, 13, 20].

EEE is expected to be used for interconnection networks in HPC systems, but few studies about it have been done.[1] The main reason is that EEE has not been standardized yet for high performance networks such as InfiniBand, so there does not exist EEE-supported devices for HPC systems. In

[1] Although the paper [17] that has a similar title to ours is submitted in ISPASS 2013, it had not been published in IEEE Xplore when we wrote this manuscript.

addition, EEE itself has not been studied thoroughly because it is a quite new technology. For example, it is still unclear that EEE can work well under the system where a parallel application runs. Only the power consumption of EEE-supported devices for ping-pong tests was evaluated [15, 16].

This paper shows a performance estimation method for HPC systems with EEE-supported devices. Our approach is to use a performance model for such systems, which we newly developed. The collaboration between the model and the network profiles for HPC applications allows us to estimate the performance of EEE-supported systems not yet realized.

The main contributions of this paper are summarized as follows.

- We propose a novel performance model for EEE-supported HPC systems. Using this model with the network profiles enables us to estimate the performance of the above systems.
- We uncover the power management scheme of an existing EEE-supported device. Although the power management scheme is not published by switch makers, we identify it through simple experiment.
- We present that EEE is also effective for parallel programs. This result would motivate HPC system developers to adopt the EEE technology.

The rest of the paper is organized as follows. First, we introduce EEE in Sect. 2, and Sect. 3 then summarizes hurdles for realizing EEE in HPC systems. Next, the proposed estimation method including the performance model is described in Sect. 4. Section 5 presents the evaluation result of the proposed method, and finally we summarize this work in Sect. 6.

2 Energy Efficient Ethernet

The power consumption of PHYs dominates that of network devices because PHYs consumes large amount of dynamic power. To confirm the connection of a network cable, PHYs that locate at the both edges of the cable are always activated and frequently communicate with each other. When no data should be transferred, a PHY creates a special packet, which is called IDLE, and it is periodically sent to the other edge. When the other PHY receives the packet, it returns an acknowledgement to the source PHY. The source PHY regards the link as live if it receives the acknowledgement.

Table 1 Specification of transition time in EEE

Protocol	Min T_s (μsec)	Min T_w (μsec)
100BASE-TX	200	30
1000BASE-T	182	16
10GBASE-T	2.88	4.48

To reduce the power consumption of PHYs, EEE was developed in the IEEE 802.3az task force. Figure 1 illustrates the concept of EEE. The basic idea of EEE is to reduce the number of IDLE transfers. When no data is transmitted, a PHY compliant with EEE sends an LPI packet instead of IDLE. If the receiver is also compliant with EEE, the source can receive an acknowledgement against the LPI packet. After that, both PHYs turn to be LPI mode where almost all parts in a PHY are shut down. This sleep process takes a little time (T_s) and consumes similar power to an active mode. After entering LPI mode, the power consumption of the PHYs can be dramatically reduced.

When a packet arrives to a PHY during LPI mode, the PHY starts a wake-up process. All hardware is powered on and whether the other PHY is ready for communication is confirmed. This wake-up process also consumes the almost same power as an active mode. After the wake-up process (which takes T_w) is finished, the PHY starts to send the packet.

To confirm the link connection during LPI mode, the refresh process is defined in EEE. When the pre-defined time (T_q) passes after entering LPI mode, both PHYs are woken up. For this purpose, they set the internal timers during the sleep process. In the refresh process, some packets are sent to confirm that the link partner is alive. After the confirmation is finished, both PHYs set the internal timers and enter LPI mode again.

The specifications of T_s and T_w in EEE [15, 16] are summarized in Table 1. In 1000BASE-T protocol, the sleep process takes hundreds of microseconds, while the wake-up

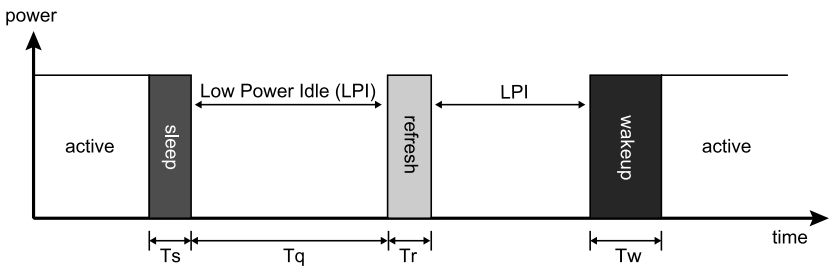

Fig. 1 Concept of Energy Efficient Ethernet

process takes dozens of micro seconds. This is 100 times faster than the reset process of an Ethernet controller [10]. Such a fast mode transition enables PHYs to sleep in short idle intervals.

Note that EEE does not define any power management policies such as timing of entering LPI mode. Since network switch makers can develop their original management policy, the power management scheme with EEE depends on network devices. Unfortunately, it is not published, so we cannot know how an EEE-enabled device works specifically.

3 Hurdles to realize EEE for HPC

Since EEE can dramatically reduce the power consumption of network devices, it is expected to be adopted in HPC systems. However, there exists the following issues to use EEE for HPC.

Development of EEE-supported devices Since EEE is an Ethernet standard, high performance interconnection networks such as InfiniBand are not compliant with EEE. EEE is available only in a part of 10GBASE-T and 1000BASE-T network devices. IEEE P802.3bj (which is a standard for 100 Gb/s backplane and copper cable) task force is now discussing about EEE [4, 7], but it would take a few years for the standardization. To utilize EEE in HPC systems, the immediate development of EEE-supported devices is desirable.

Power/performance estimation when using EEE To encourage HPC system developers to adopt the EEE technology, power and performance estimation of HPC systems with them is required. If EEE is also energy-efficient in HPC systems, it would motivate the developers to use the technology.

Establishment of power management scheme Power management scheme with EEE should be established for the further energy-efficient HPC systems. EEE is a part of the MAC layer protocol, so it does not define the power management policy of PHYs. The current policy implemented in an existing EEE-supported device is effective for LANs, but it may be ineffective for HPC networks. We need to clarify what power management scheme is suitable for HPC systems.

Since the first one is a matter of industry, this paper focuses on the rest. Above all, our concern is how to estimate the system performance of EEE-supported HPC systems. This is because the power consumption of such systems can be estimated by the combination of existing models [16]. Of course, the power management scheme should be carefully considered to develop the performance estimation method because the system performance highly depends on the management scheme.

4 Performance estimation method for EEE-supported HPC systems

EEE-supported interconnection network devices do not exist yet, but the benefit of introducing EEE into HPC systems should be estimated. To this end, we propose the performance model for EEE-supported HPC systems. The model with the network profiles on an existing HPC system enables us to estimate the system performance if the system adopted the EEE technology.

4.1 Power management scheme

IT devices such as CPUs, LCDs and HDDs use a simple policy for their power management: time-out control and on-demand wake-up [14, 19]. An IT device turns into a low power mode when constant time passes in an idle state where no request arrives. When a request arrives during a low power mode, the device starts to be powered up. Since the request must wait for the readiness of the device, its response time is exacerbated in this situation. In spite of this performance penalty, time-out control and on-demand wake-up are widely used in IT devices because of their simplicity.

As described later, this scheme is also suitable for EEE-supported devices of HPC systems. It is well-known that a lot of HPC applications have computation and communication phases. This means that the networks of HPC systems are used bursty and then keep idle. Therefore, if the time-out interval is appropriately set, an EEE-enabled device can keep an active state in a communication phase, while it can largely save the power in a computation phase.

When using time-out control and on-demand wake-up, the time-out interval and the mode transition time are the key factors to determine the system performance. The former determines how often a PHY enters LPI mode, while the latter determines how large the recovery time from LPI mode is. Proposed performance models should respond to their variability.

4.2 Performance model of EEE-enabled systems

Let i be a parallel application and j be the running threads per node of the application. We assume that the execution time T^{ij} of the application on an EEE-enabled system is defined as follows:

$$T^{ij} = T^{ij}_{base} + T^{ij}_{overhead} \tag{1}$$

where T^{ij}_{base} is the execution time when the application runs on an EEE-unsupported system, and $T^{ij}_{overhead}$ is the time overhead caused by EEE-enabled.

Here, we suppose $T^{ij}_{overhead}$ can be described below.

$$T^{ij}_{overhead} = n^{ij} * f\left(I^{ij}\right) \tag{2}$$

where n^{ij} is communication count per node; f is the function that expresses the time overhead per communication; and I^{ij} is the average idle interval of the network when the application runs.

If the network device utilizes time-out control and on-demand wake-up, ideally f forms a step function that shows the sudden rise at the time-out interval. However, since each idle interval is not completely equal to I^{ij}, actually f presents the gradual increase around the time-out interval. Therefore, we denote f as the following equation.

$$f\left(I^{ij}\right) = \begin{cases} 0 & (I^{ij} \leq t) \\ c * (1 - \exp(-\alpha * (I^{ij} - t_0))) & (I^{ij} > t) \end{cases} \quad (3)$$

where c and α are constant values that depend on the parameters of the network device; and t is a threshold. The time overhead can be classified into two cases depending on average idle intervals.

When the average idle interval is extremely short ($I^{ij} \leq t$), an EEE-supported device cannot enter LPI mode because the next packet comes into the device before the internal timer reaches the time-out interval. That is, all packets do not suffer from the performance penalty caused by the mode transition. The time overhead can be regarded as zero in this situation.

After the average idle interval is beyond a threshold ($I^{ij} > t$), some idle intervals exceed the time-out interval and the EEE-supported device often enters LPI mode. As previously mentioned, packets arriving the device during LPI mode are affected by the overhead of the mode transition and the system performance then degrades. The packet suffering from the above overhead increases as the network load decreases. Finally, every packet transfer is delayed by the mode transition time. We simply model this situation using an exponential function that is asymptotic to a constant value c.

Note that t, c and α depend on EEE-supported devices, especially their time-out intervals and mode transition time. In the performance estimation, some assumption is needed for them.

I^{ij} can be described as follows:

$$I^{ij} = \left(T_{base}/n^{ij}\right) - \left(S^{ij}/B\right) \quad (4)$$

where S^{ij} is the average data size of communication per node; and B is the network bandwidth per node. This assumes that all communication occurs periodically and transmits the same size of data. For example, supposing a program of which the execution time is 1 second on a system using a 1 Gb/s link and in which 1024 bit data is transferred 1000 times, its average idle interval is 999 microseconds.

Here, regarding the EEE-unsupported system as an existing HPC system, we can easily obtain the input parameters of the above models. If we would like to know T^{ij}_{base}, we should indeed execute the application i on the system while invoking j threads per node. n^{ij} and S^{ij} can be measured by a profiling tool such as TAU [18] and Scalasca [11]. Of course, B is already known. Thus, we can estimate the system performance if the system adopted EEE.

5 Evaluation

To evaluate the accuracy of the proposed model, an EEE-enabled device using time-out control and on-demand wake-up is needed. Fortunately, we were able to find out such a device, so we used it for the evaluation.

First, this section presents that the above device uses time-out control and on-demand wake-up through the simple experiment. Next, with a PC cluster system using this device, we will show how accurate the proposed model is.

5.1 Examination of power management scheme of existing EEE-supported devices

In order to examine the power management scheme of an EEE-supported device, a simple ping-pong test was carried out on an EEE-supported computing system. To our best knowledge, this is the first trial that clarifies the power management scheme of a commercial EEE-supported device.

5.1.1 Experimental methodology

If an EEE-supported device uses time-out control and on-demand wake-up for the power management of PHYs, the round trip time of a packet should be worsened as its send interval is lengthened. When a send interval is shorter than the time-out interval, a packet can be sent normally. On the other hand, when the former interval is longer than the latter interval, sending the packet is delayed by T_w because of waking the PHY up.

To confirm the above assumption, we constructed the evaluation platform shown in Fig. 2. The platform consists of one network switch and two compute nodes. The switch and the NIC of Node 0 support EEE, while the NIC of Node 1 does not. Therefore, only the link connected between the switch and Node 0 can enter LPI mode.

System configurations are listed in Table 2. Dell PowerConnect 5548 was used, which is a 48-port gigabit Ethernet switch compliant with IEEE 802.3az. For an EEE-supported node, HP ProLiant DL360p Gen8 with HP Flexible LOM 1 Gb 331FLR adapter was selected. Lenovo ThinkPad T400s with Intel 82567LM Ethernet adapter was used as an EEE-unsupported node. The nodes and the switch connected by CAT5e cable.

On the above system, we carried out a ping-pong test as changing send intervals of packets, and then measured their response time. *Ping* command was used for the above test,

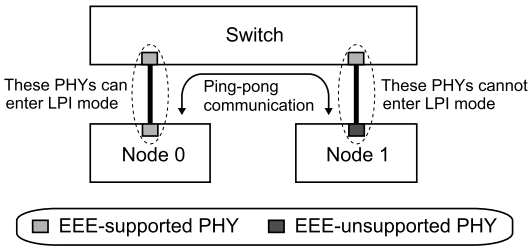

Fig. 2 Evaluation platform for ping-pong test

Table 2 System configuration of the ping-pong test platform

Device	Remarks
Switch	Dell PowerConnect 5548
Node 0	HP ProLiant DL360p Gen8 (NIC: HP Flexible LOM 1 Gb 331FLR)
Node 1	Lenovo ThinkPad T400s (NIC: Intel 82567LM Ethernet Adapter)

and the send intervals were varied from 0.5 to 4 milliseconds by using the "$-i$" option. For each send interval, 1000 packets were sent from one of the nodes to the other, and we measured the average of their response time. Each packet consists of 64 byte data including 8 bytes ICMP data.

Note that T_w of PowerConnect 5548 is defined as 60 microseconds. We confirmed it on the management console of PowerConnect 5548.

5.1.2 Experimental result

The result of the ping-pong test is shown in Fig. 3. The horizontal axis represents send intervals, while the vertical axis represents the response delay caused by EEE-enabled. The response delay means that the response time in EEE-disabled is subtracted from that in EEE-enabled. The line represents a trend of the response delay when Node 0 pings Node 1, while the dotted line represents vice versa.

The result indicates that PowerConnect 5548 uses time-out control and on-demand wake-up. In the figure, the response delay is nearly equal to zero under 1 millisecond send interval. This means that the PHYs did not enter LPI mode because packets were continuously sent within the time-out interval. As send intervals are lengthened, the response delay is close to the T_w (60 microseconds). This means that the PHYs turned to do the wake-up process every packet transfer.

The result also indicates that the time-out interval of PowerConnect 5548 is 1 millisecond. In the figure, at the 1 millisecond send interval, the response delay suddenly raises up by dozens of micro seconds. This means that the response time in EEE-enabled is delayed by the mode transition because the send interval exceeds the time-out interval.

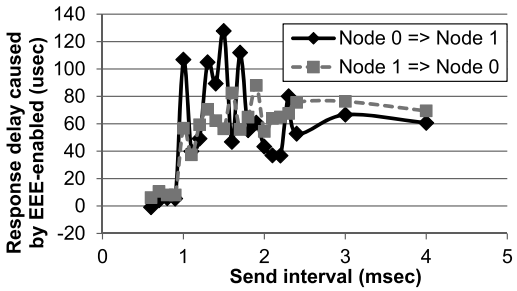

Fig. 3 Response delay caused by EEE-enabled

Note that the response delay is often beyond T_w. This is considered as the impact of T_s. When a packet arrives in the sleep process, PowerConnect 5548 seems to wait for the completion of the process. Since the wake-up process starts after the sleep process finishes, the response delay within this range shows around 100 microseconds.

5.2 Evaluation of performance estimation method

Using a PC cluster with PowerConnect 5548, we evaluated the effectiveness of our approach. The evaluation result is presented in this section.

5.2.1 Evaluation methodology

To evaluate the proposed method, we built the EEE-supported system shown in Fig. 4. The system consists of one EEE-supported switch and four compute nodes that have EEE-supported NICs. Each node is flatly connected to the switch via a CAT5e cable.

The switch and node configurations are summarized in Table 3. Dell PowerConnect 5548 was used as the switch, and HP ProLiant DL360p Gen8 was used as a node again. Each node contains two sockets of Xeon E5-2680 that has eight cores. To exclude the impact on the performance caused by something except for EEE, we disabled all functions that strongly affect the system performance, such as Hyper-Threading, Turbo Boost and *cpuspeed*. Moreover, with *cpufreq*, we set the clock frequency in all cores at its maximum (2.7 GHz). The main memory is DDR3-1333 and its capacity is 64 GB in each node. The switch and NICs support 1000BASE-T protocol, so the network bandwidth per node is 1 Gb/s.

To verify the accuracy of the proposed model, we made a simple program in which all threads uniformly communicate (Fig. 5). The program repeats *MPI_Alltoall* 100,000 times for the given array. To adjust the communication interval, we inserted the *usleep* function between MPI functions. In the evaluation, the array size varies from 256 B to 128 KB, and the sleep time changes from 0 to 1,000 microseconds.

In addition to the synthetic application, NAS Parallel Benchmarks (NPB) 3.3.1 [3] were used for the evaluation.

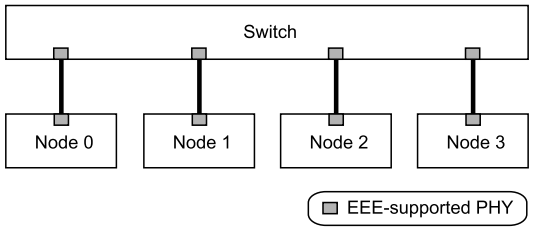

Fig. 4 Evaluation platform for the proposed model

Table 3 System configuration of the EEE-supported system

Device	Remarks
Switch	Dell PowerConnect 5548
Node	HP ProLiant DL360p Gen8 (NIC: HP Flexible LOM 1 Gb 331FLR, CPU: Xeon E5-2680 (2.7 GHz, 8-core), 2 CPUs, Memory: 32 GB (8 GB 2R×4 DDR3-1333 ×4))

```
#include <mpi.h>
#define ITERATION 100000

int main(int argc, char** argv) {
  /* initialization */
  for (i = 0 ; i < ITERATION ; ++i) {
    MPI_Alltoall(&data, numdata, MPI_INT, &out,
                 numdata, MPI_INT, MPI_COMM_WORLD);
    usleep(stime);
  }
  /* finalization */
}
```

Fig. 5 A synthetic application with uniform communication

All programs were compiled by OpenMPI-1.5.4 and gcc-4.4.6 with "-O2-funroll-loops". When executing a program, we invoked several MPI processes on each node. The number of processes per node varies as 1 and 4, so the total process count is 4 and 16, respectively. We used three problem classes: A, B and C. The program was executed 10 times for each process count and problem class. Then, we calculated the average of their execution time.

For the network profiling, we utilized *paraprof* command from TAU-2.22.2 [18]. The command with the "-dumpmpisummary" option provides us with the total transferred bytes and the total call count of MPI functions per process when an MPI program runs. Using these values, we calculated the average idle interval I^{ij} when the application i runs under j processes. We inputted the calculated value into the proposed model and compared the estimated performance to the actual performance.

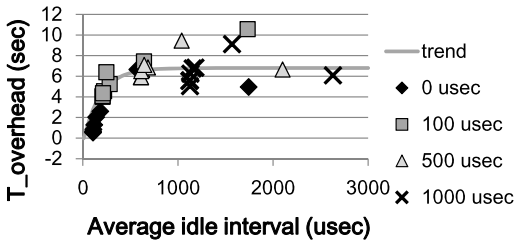

Fig. 6 The time overhead caused by EEE-enabled and the accuracy of its model (for 4-process execution)

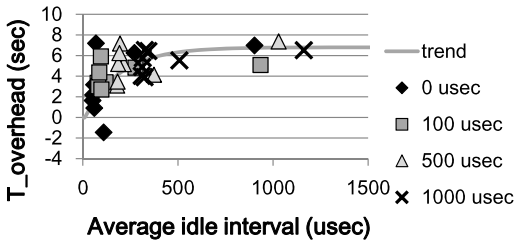

Fig. 7 The time overhead caused by EEE-enabled and the accuracy of its model (for 16-process execution)

5.2.2 *Evaluation result*

Figures 6 and 7 show the time overhead caused by EEE-enabled for the synthetic application. The horizontal axis represents the average idle interval calculated by Eq. (4), while the vertical axis represents the time overhead. A point in the figure indicates the result of a trial when the application runs with an array size and a sleep interval. The time overhead is calculated by the subtraction of the elapsed time in EEE-disabled from that in EEE-enabled. The line illustrates the trend calculated by Eq. (2), which is fitted to the measured data by the least square method.

Figure 6 indicates that our model can well express the trend of the time overhead. As shown in the figure, the most points converge on the trend line. Note that a few points that are far from the trend line are incurred by the fluctuation of the clock frequency. In fact, the difference of elapsed time between trials under the same configuration was up to 13.2 seconds. Since the firmware employed in HP ProLiant can control the core frequency independent of the operating system, it sometimes degrades the frequency. This mechanism affects the performance at these points.

As shown in the figure, the rising point of the trend line is different from the time-out interval shown in Fig. 3. This is because a part of idle intervals does not match their average. Although the average idle interval is smaller than the time-out interval, some intervals may exceed the time-out interval and then cause the system performance degradation.

The figure also presents that the average idle interval becomes longer as the array is larger. This is because time overheads except for data transfer are not considered in our

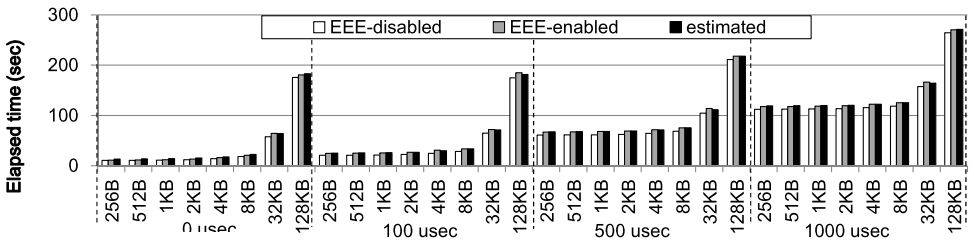

Fig. 8 Real elapsed time and the estimated time when the synthetic application runs (for 4-process execution)

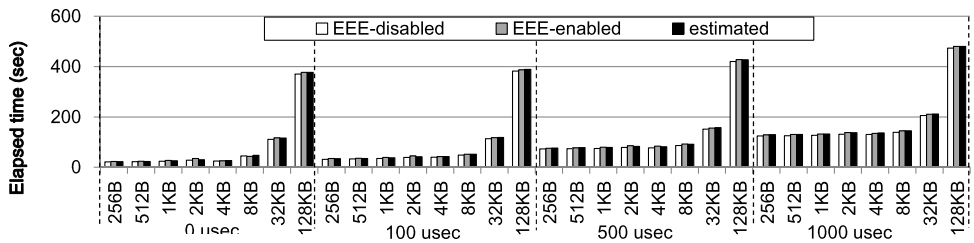

Fig. 9 Real elapsed time and the estimated time when the synthetic application runs (for 16-process execution)

model. Generally speaking, pre- and post-process of data transfer in an MPI function take much more time as the communication data is larger, but we did not model it. In fact, the elapsed time of an MPI function with 128 KB array is around 1.8 milliseconds as shown in Fig. 8, while the transmission time of actually transfered data in the function is much smaller. According to the *paraprof*'s result, the transfered data in the function is 1.73 KB data, so its transmission time through 1 Gbps link is only 1.73 microseconds. That is, a large fraction of time in the function is spent in pre- and post-process of data transfer.

In contrast to the 4-process execution, for the 16-process execution (Fig. 7), the proposed model presents a bit mismatch at the small average intervals. This is because our assumption where all MPI functions in a node are called periodically is not satisfied. Four processes running on a node concurrently call an MPI function. Then, they simultaneously sleep after the function call finishes. That is, the communication count is overestimated in terms of idle intervals, so the proposed model estimates that the average idle interval is smaller than the real. We will try to improve the model of the average idle interval in the future.

Figures 8 and 9 present the elapsed time in EEE-disabled, that in EEE-enabled and the time estimated by the proposed method. The former figure is the result of the 4-process execution, while the latter is the result of the 16-process execution. In each figure, the horizontal axis represents the array size and the sleep interval used for the execution, while the vertical axis represents the elapsed time. For the visibility, a part of Fig. 8 is enlarged as shown in Fig. 10.

Firstly, both figures show that EEE significantly worsens the system performance. In the worst case, where 4 KB data and 100 microseconds sleep are used in the 4-process execution, the performance degradation reaches 25.8 % (Fig. 10).

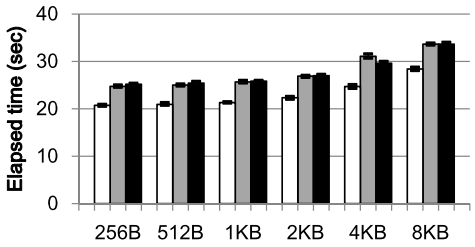

Fig. 10 The scaled version of Fig. 8 at 100 microseconds

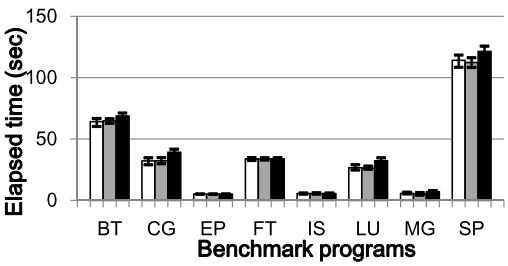

Fig. 11 Real elapsed time and the estimated time when NPB runs (16-process execution, class B)

Secondly, the figures indicate that the proposed model can well estimate the performance of an EEE-supported system. The error between the estimated time and the elapsed time in EEE-enabled achieves 2.63 % (4-process) and 1.11 % (16-process) on average, respectively. Even in the worst case, the error still presents 20.0 % and 15.3 % in each.

Figure 11 shows the elapsed time when NPB runs with 16 processes and the problem class B. We had to omit other

results for want of space. Since the communication count is significantly lower in the most configurations, the elapsed time in EEE-enabled is similar to that in EEE-disabled. The proposed model also presents the same tendency, but it overestimates the execution time in some cases. This seems to be incurred by the inaccuracy of the average interval model because a lot of trials by 16 processes show the overestimation. Thus, the further improvement of our model is needed.

6 Summary

EEE will be adopted in the future HPC systems, but its impact on the system performance was unclear because of lack of the performance estimation method for such systems. This paper presents the novel performance estimation method for EEE-supported systems, which utilizes the newly developed model and a few network profiles on existing systems. The evaluation result shows that the proposed model has the significant accuracy. Moreover, the result shows that the performance penalty caused by EEE-enabled is negligible in the realistic situation.

For the future work, we will try to carry out the further improvement of our model and confirm it under various applications and platforms. Especially, we are interested in the accuracy of the proposed model in the large diameter networks because the performance penalty caused by EEE-enabled increases in such networks. By adding the further evaluation, we would like to confirm that our approach is also effective under other conditions.

Acknowledgements This research was partially supported by MEXT and JST CREST projects for next-generation high performance computing systems.

References

1. Ajima Y et al (2012) TOFU: interconnect for the K computer. Fujitsu Sci Tech J 48(3):280–285
2. Ajima Y et al (2009) TOFU: a 6D Mesh/Torus interconnect for exascale computers. IEEE Comput 42(11):36–40
3. Bailey D et al (1994) The NAS parallel benchmarks. RNR Technical Report RNR-94-007
4. Bannet MJ (2011) Energy-efficient Ethernet for 100 G backplane and copper. IEEE P802.3bj task force
5. http://www.broadcom.com/products/features/energy_efficient_network.php
6. D-Link (2011) DGS-1100 EasySmart switches 16/24 port gigabit switches. Datasheet
7. Gustlin M (2011) 100 Gb/s Ethernet and EEE. IEEE P802.3bj task force
8. Hewlett-Packard (2011) HP E8200 z1 v2 switch series. Technical specifications
9. IEEE (2010) IEEE Std 802.3az-2010. IEEE Standards
10. Intel Corp (2012) Intel Ethernet controller I350 specification update. Revison 2.05
11. Geimer M et al (2010) The scalasca performance toolset architecture. Concurr Comput 22(6):702–719
12. Kogge PM (2008) Architectural challenges at the exascale frontier, simulating the future: using one million cores and beyond. Invited talk
13. Level One (2011) GEU-0820 8-port gigabit switch. Datasheet
14. Li K et al (1994) A quantitative analysis of disk drive power management in portable computers. In: USENIX winter, pp 279–291
15. Reviriego P et al (2012) An initial evaluation of energy efficient Ethernet. IEEE Commun Lett 15(5):578–580
16. Reviriego P et al (2012) An energy consumption model for energy efficient Ethernet switches. In: International conference on high performance computing and simulation, pp 98–104
17. Saravanan KP et al (2013) Power/performance evaluation of energy efficient Ethernet (EEE) for high performance computing. In: 2013 IEEE international symposium on performance analysis of systems and software
18. Shende S, Malony AD (2006) The TAU parallel performance system. Int J High Perform Comput Appl 20(2):287–311
19. Shye A et al (2009) Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In: Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture, pp 168–178
20. Trendnet (2011) 8-port gigabit GREEENnet switch. Datasheet

Shinobu Miwa was born in 1977 and received Doctor of Informatics degree from Kyoto University in 2007. He is now an assistant professor at the University of Tokyo from 2011. His research interests are computer architecture, high performance computing and embedded systems. He is a member of IEEE.

Sho Aita was received the B.E. degree in Department of Mathematical Engineering and Information Physics from the University of Tokyo in 2012. He is currently pursuing the Master's degree in Information Physics and Computing from Graduate School of Information Science and Technology at the University of Tokyo. He is interested in high performance computing systems. Sho Aita was received the B.E. degree in Department of Mathematical Engineering and Information Physics from the University of Tokyo in 2012. He is currently pursuing the Master's degree in Information Physics and Computing from Graduate School of Information Science and Technology at the University of Tokyo. He is interested in high performance computing systems.

Hiroshi Nakamura is a Professor at The University of Tokyo. He received the Ph.D. degree in Electrical Engineering from The University of Tokyo in 1990. His research interests include power-efficient computer architecture and VLSI design for high-performance and embedded systems. He is now leading the Normally-Off Computing Project supported by NEDO, New Energy and Industrial Technology Development Organization in Japan. He served IEEE ISLPED 2011 (International Symposium on Low Power Electronics Design) as a general chair. He is a senior member of IEEE and ACM.