Dynamic core binding for load balancing of applications parallelized with MPI/OpenMP

Masatoshi Kawai[1], Akihiro Ida[2], Toshihiro Hanawa[3], and Kengo Nakajima[3,4]

[1] Nagoya University, Japan kawai@cc.nagoya-u.ac.jp
[2] Research Institute for Value-Added-Information Generation (VAiG), Japan
[3] The University of Tokyo, Japan
[4] Riken, Japan

Abstract. load imbalance is a critical problem that degrades the performance of parallelized applications in massively parallel processing. Although an MPI/OpenMP implementation is widely used for parallelization, users must maintain load balancing at the process level and thread (core) level for effective parallelization. In this paper, we propose dynamic core binding (DCB) to processes for reducing the computation time and energy consumption of applications. Using the DCB approach, an unequal number of cores is bound to each process, and load imbalance among processes is mitigated at the core level. This approach is not only improving parallel performance but also reducing power consumption by reducing the number of using cores without increasing the computational time. Although load balancing among nodes cannot be handled by DCB, we also examine how to solve this problem by mapping processes to nodes. In our numerical evaluations, we implemented a DCB library and applied it to the lattice $\mathcal{H}$-matrixes. Based on the numerical evaluations, we achieved a 58% performance improvement and 77% energy consumption reduction for the applications using the lattice $\mathcal{H}$-matrix.

Keywords: Dynamic Core Binding, Dynamic Load Balancing, Power-aware, Hybrid Parallelization, Simulated Annealing, Lattice $\mathcal{H}$-matrix

1 Introduction

Load balancing is one of the most important issues in parallelizing applications for massively parallel processing. Load imbalance results in poor application performance and power wastage due to the long waiting for barrier synchronizations. Nevertheless, there are many algorithms for which load balancing is difficult, such as the particle-in-cell (PIC) method, lattice Boltzmann method, and hierarchical matrix ($\mathcal{H}$-matrix) method. In the case of the PIC and lattice Boltzmann methods, the amount of computation per unit area exhibits drastic variations with respect to time. In the H-matrix method, it is difficult to estimate the load bound to each process before creating the matrix. Moreover, hybrid programming by MPI (process) and OpenMP (thread) is widely used for parallelization, and the load must be equal at both the process and thread levels. Considering load balancing on multiple levels might be very complex work.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_30

Dynamic load balancing is a well-known concept for reducing such load imbalance. Task parallelism emphasizes a parallelized nature in the applications, but it may cause poor performance due to the large overhead. To achieve good performance, a specialized approach for each application is used. The disadvantage of the specialized approach is required cumbersome jobs for optimization because of low portability. Therefore, we need an effective load balancing approach with low overhead, low implementation costs, and good portability.

In this study, we propose a dynamic core binding (DCB) approach that supports effective and simple load balancing. Whereas in the standard MPI/OpenMP hybrid parallelization environment, an identical number of cores is bound to all processes, in the DCB environment, the number of cores bound to each process is changed to absorb load imbalance among processes inside of a node. DCB can achieve improving the parallel performance of reducing power consumption by the different core binding policies. To improve the performance of the applications, it binds all cores to processes so that the loads per core are similar. Depending on the process with the maximum load, it reduces the cores mapped to the other processes; as a result, the energy consumption can be reduced without changing the computation time. This approach is suitable for many-core architectures, which have recently become more general. Load balancing by DCB has low overhead and high versatility as it only involves changing the number of bound cores. For supporting the DCB environment, an appropriate system call(not required administrator permission) is required. Then we implement the DCB library to black box this system call.

In addition, to maintain the effectiveness of the DCB approach, we consider the load imbalance among nodes. The DCB library only supports load balancing inside a node. Therefore, loads among nodes must be balanced using another method. The load balancing problem among nodes is similar to the job scheduling problem [12], which is a type of combinatorial optimization problem (COP). Job scheduling problems are categorized as nondeterministic polynomial-time complete (NP-complete) [12], and it is difficult to find the optimal solution to these problems with many jobs and machines. Studies on job scheduling problems discussed the use of simulated annealing (SA) to find appropriate approximate solutions [1, 6] and have reported good results. Therefore, we also use the SA to absorb the load imbalance among nodes. In order to use the results obtained from the SA, the DCB library has a mechanism to generate a new MPI communicator, which is balanced loads among nodes.

The originality of this study is that it proposes a single approach (changing the cores bound to each process) to reduce the computation time and/or energy consumption of applications based on load balancing. In addition, we consider load imbalance among nodes using a metaheuristic approach to improve the effectiveness of DCB. Two existing studies [2, 4] proposed ideas similar to ours. By changing the number of cores, one study [2] achieved performance improvement of hybrid parallelized applications, while the other study [4] reduced the energy consumption of applications parallelized using OpenMP. Our proposed approach integrates these studies [2, 4] and also considers load balancing among nodes. Various studies achieved a reduction in the computation time of hybrid parallelized applications by dynamic load balancing. In one study [14], the imbalance among threads caused by communication hiding was solved by the task parallelization of OpenMP. Another study [5, 11] proposed load balancing with task parallelization by original libraries in MPI/OpenMP or MPI-parallelized applications. Our proposed approach fundamentally differs from the aforementioned studies, and performance improvement can be achieved using a simple interface. Studies have also been conducted on power awareness [3, 13, 15] by throttling core and memory clocks to reduce energy consumption. In particular, one study [13] focused on applications parallelized with MPI/OpenMP. The difference between our proposed approach and these studies is that we achieve a reduction in energy consumption by changing the cores bound to each process; furthermore, our approach supports performance improvement and reduction in energy consumption simultaneously.

We discuss the effectiveness and usability of the DCB library based on a sample implementation and numerical evaluations with a practical application using a lattice $\mathcal{H}$-matrix [9, 10]. The lattice $\mathcal{H}$-matrix method is utilized to approximate naive dense matrices and focuses on reducing communication overhead during parallel computation. However, this improvement increases load imbalance among the processes. Therefore, the lattice $\mathcal{H}$-matrix is a suitable target for evaluating the effectiveness of the DCB library.

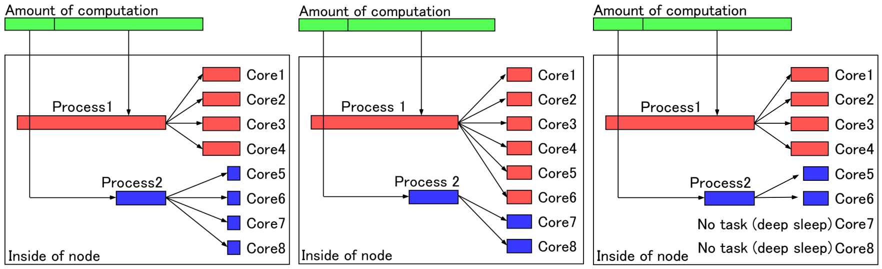
2 Dynamic Core Binding(DCB)

In this section, we introduce a method for improving parallel performance and/or reducing energy consumption using DCB. In addition, we indicate an overview of the interfaces of the DCB library that provide the DCB approach and describe its use.

The basic concept of the DCB approach is to equalize loads among cores by changing the number of cores bound to each process. The target of the DCB is parallelized applications based on OpenMP/MPI hybrid. In the general hybrid parallelization environment, the number of cores bound to each process is identical. Balancing the loads at the core level leads to reducing computational time and/or energy consumption. Then, as shown in Fig. 1(a), if ”Process 1” has a three times larger amount of computation than ”Process 2”, the allocated amount of computation to cores 1∼4 is also three times larger than cores 5∼8. By the basic concept of DCB, we expect to reduce the load imbalance among the cores.

In OpenMP, there is a function ”*omp_set_num_threads*” for changing the number of threads. However, this function does not change the number of usable cores for each process. When users increase the number of threads such that they exceed the number of cores, some threads are bound to the same core. When the number of threads is less than the number of cores, the threads can be moved among the cores by controlling the operating system. Changing the number of cores bound to each process requires a system call such as ”*sched_set_affinity*” or

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_30



(a) Without DCB (General environment) (b) DCB-RC (Reduce computational time) (c) DCB-PA (Reduce energy consumption)

Fig. 1: Policies of dynamic core binding(DCB) library for core binding

alternative functions with unique arguments. The DCB library internally calls these system calls, allowing users to easily benefit from an environment with an unequal number of cores bound to each process. The permission of the system calls inside of the DCB library is not limited to the super-user, and the DCB library can be used with normal permissions.

The DCB library determines the number of bound cores based on a parameter received from the user. The DCB library implements two core-binding policies to reduce computational time and/or energy consumption. In Section 2.2, we introduce these two policies. The DCB library focuses on load balancing inside each node, and load imbalance among nodes must be addressed using a different approach. In Section 2.3, we describe the use of SA to solve load imbalance among nodes.

2.1 Concept of DCB

2.2 Binding policy

In the DCB library, we implement two policies for improving parallel performance or reducing the energy consumption of applications. The policies of the DCB library change how the cores are bound to each process.

One policy of the DCB library focuses on reducing the computation time of the application (referred to as the RC policy). In the RC policy, all cores are bound to a process based on the load.Here we explain how to determine the number of cores bound to each process using the RC policy. The bound cores are determined by an argument l_p, which denotes the load executed by processes p, passed to the DCB library by the user. The optimal l_p depends on the application and is assumed to include the amount of computation and memory transfer for each process. Then, we consider a system whose nodes have c cores, and the IDs of nodes constructed the system are denoted as set N_a. The IDs of all processes denotes as set P_a, and the IDs of the processes launched on the node n as set P_n. The DCB library with the RC policy determines the number of cores c_p bound to process p launched on node n as follows:

$$c_p = \left\lfloor \frac{l_p}{\Sigma_{\tilde{p}\in P_n} l_{\tilde{p}}} (c - |P_n|) \right\rfloor + 1 \qquad (1)$$

In this equation, the term $c - |P_n| + 1$ guarantees binding more than one core to all processes. If there are remaining cores ($c - \sum_{p \in P_n} c_p \neq 0$), the DCB library bounds one more core ($c_p = c_p + 1$) in order to the value $\beta = (l_p / \Sigma_{\tilde{p} \in P_n} l_{\tilde{p}})\, c - c_p$ until there are no more cores remaining. When we apply the RC policy in the state Fig. 1(a), six cores are bound to "Process 1" and two cores are bound to "Process 2" as shown in Fig. 1(b). In this situation, the values of (1) are $l_1 = 3$, $l_2 = 1$, $c = 8$, $P_1 = 2$, respectively. The expected minimum computation time by using DCB is as follows:

$$t_{dcb} = t_{cmp} \left(\frac{l_m}{c_m} \right) / \left(\frac{\max L_n}{c} \right) + t_{oth}, \quad L_n = \left\{ \sum\nolimits_{p \in P_n} l_p : n \in N_a \right\} \qquad (2)$$

Here, t_{cmp} denotes the time of computations that are parallelized with OpenMP in the applications, and t_{oth} denotes the time of the other part of the application such as the communication and sequential computational part. m denotes the ID of the process that has the largest load in the node n which has the largest sum of load. In (2), t_{dcb} is calculated as the ratio of the maximum load allocated to the core with and without DCB on the node which has the largest sum of the loads. (2) also shows that the performance improvement of the DCB is capped by the load imbalance among the node. If the loads among the nodes are equalized, $maxL_n$ is minimized, and the effectiveness of DCB is maximized.

The second policy of the DCB library focuses on reducing energy consumption. In the PA policy, the DCB library reduces the number of cores bound to a process based on the largest load of the process. Fig 1(c) illustrates the result of applying the PA policy to the state Fig. 1(a). In this example, two cores are not mapped to any process, and their power status is changed to "deep sleep". The energy consumption of the deep sleeping cores is drastically reduced relative to cores in the running state.The number of bound cores with the PA policy is determined as follows:

$$c_p = \left\lceil \frac{l_p}{\max L_a} c \right\rceil, \quad L_a = \{ l_p : p \in P_a \} \qquad (3)$$

The bound cores c_p by PA policy are simply decided as the ratio of l_p and maximum load in all processes. The expected reduction ratio ρ of the energy consumption is calculated as follows:

$$\rho = (J_g - J_{idle}) \frac{\sum_{p \in P_a} c_p}{c \times |N_a|} + J_{idle} \qquad (4)$$

Here, J_g and J_{idle} denote the energy consumption of the application without DCB and the idle CPU state, respectively. The ratio ρ is expressed as the ratio of actually bound cores to all cores that construct the target system. In contrast to the RC policy, the effectiveness of DCB with the PA policy is not capped by the load imbalance among nodes.

2.3 Load balancing among nodes

DCB only supports load balancing within a node. Load balancing among nodes must be maintained by another approach. Load balancing among nodes can be considered a massive COP; this COP is similar to a job scheduling problem and is categorized as NP-complete. Solving the COP with the existing libraries for classical computers takes a long time and is impractical. Then we consider solving the COP by using SA, which can be easy to solve. SA has a lot of experience in solving complex COPs, and various companies provide services that enable the use of SA. In order to solve the COP using SA, it is necessary to convert the COP into a quadratic unconstrained binary optimization (QUBO) and submit it to SA.

Then, a requirement and constraints of COPs for load balancing among the nodes are as follows:

Item.1 The loads among nodes are equalized
Item.2 Every process is mapped to one of the node
Item.3 Every node has more than one process

These requirements are expressed as the following Hamiltonian of the QUBO model:

$$\begin{aligned} H = \underbrace{\sum_{n=1}^{|N_a|}\left(\frac{\sum_{p=1}^{|P_a|} l_p}{|N_a|} - \sum_{p=1}^{|P_a|} l_p x_{p,n}\right)^2}_{Item.1} &+ \underbrace{\lambda \left\{ \sum_{p=1}^{|P_a|}\left(1-\sum_{n=1}^{|N_a|} x_{p,n}\right)^2 \right.}_{Item.2} \\ &+ \underbrace{\left. \sum_{n=1}^{|N_a|}\left(\sum_{s=0}^{\lceil \log_2 m\rceil} 2^s y_s - \sum_{p=1}^{|P_a|} x_{p,n}\right)^2 \right\}}_{Item.3} \end{aligned} \tag{5}$$

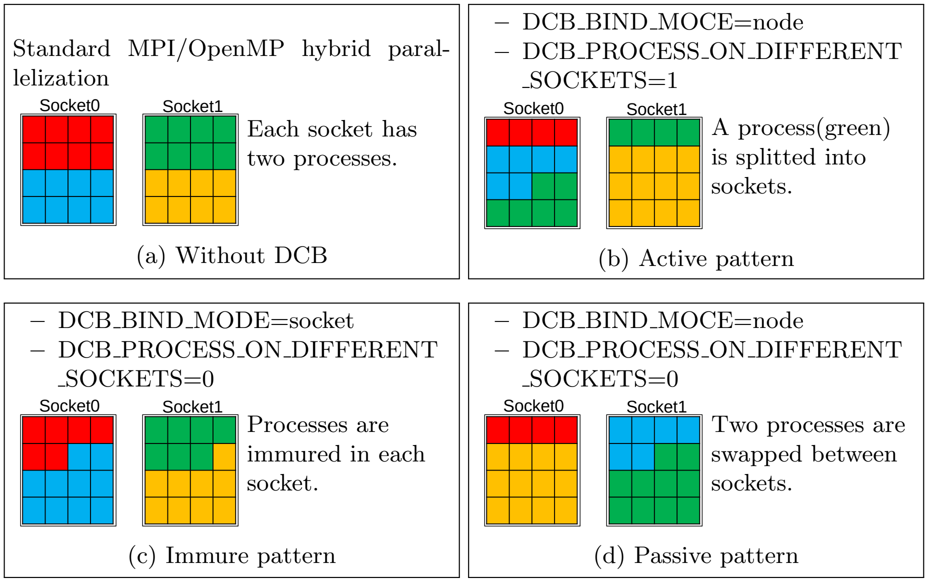
In this Hamiltonian, the first, second, and third terms express the Item.1, Item.2, and Item.3 of the requirements, respectively. Here, $x_{p,n}$ denotes the binary values appearing in the mapping. If $x_{p,n} = 1$, process p is mapped to node n. Also, binary values y_s denote sticky bits for expressing inequality of Item.3. λ denotes the penalty term for expressing the constraints. The objective of SA is looking for a combination of $x_{p,n}$ when Hamiltonian H becomes zero. (5) is constructed such that H is zero if all conditions are satisfied. We use pyQUBO [16], a Python module, for handling, and Fujitsu Digital Annealer (FDA) [8] to solve the QUBO model.

3 DCB library

In this section, we introduce the implementation of and how to use the DCB library.

3.1 Interface

Important functions of the DCB libraries are two, ”*DCB_init*” and ”*DCB_balance*. Users must call the ”*DCB_init*” function before calling ”*DCB_balance*”. ”*DCB_*


Standard MPI/OpenMP hybrid parallelization
Socket0
Socket1
Each socket has two processes.
– DCB_BIND_MOCE=node
– DCB_PROCESS_ON_DIFFERENT _SOCKETS=1
Socket0
Socket1
A process(green) is splitted into sockets.
– DCB_BIND_MODE=socket
– DCB_PROCESS_ON_DIFFERENT _SOCKETS=0
Socket0
Socket1
Processes are immured in each socket.
– DCB_BIND_MOCE=node
– DCB_PROCESS_ON_DIFFERENT _SOCKETS=0
Socket0
Socket1
Two processes are swapped between sockets.

(a) Without DCB

(b) Active pattern

(c) Immure pattern

(d) Passive pattern

Fig. 2: Mapping pattern of cores on node controlled by numerical environment. Two sockets are on the node and four processes are mapped to the node. The loads l_p (passed parameters from the user) of each process are as follows : Process0:20, Process1:30, Process2:50, Process3:60

balance" is the main function, and the number of cores bound to each process is changed by referencing l_p that is passed as an argument of the function. After calling the "*DCB_balance*" function, users can use the DCB environment inside "*#pragma omp parallel*" regions. If the load of each process is expected to vary significantly from one OMP parallel region to another, calling "*DCB_balance*" can maintain a uniform amount of computation for each process in a node.

- DCB_MODE_POLICY = ("compute" or "power"): change the binding policy of the DCB
- DCB_DISABLE = (0 or 1): disable the DCB
- DCB_PROCESS_ON_DIFFERENT_SOCKETS = (0 or 1): map a process on a different socket
- DCB_BIND_MODE = ("node" or "socket"): Bounding processes to the whole node or a socket domain

DCB_MODE_POLICY is used to change the policies of the DCB library. If users set DCB_MODE_POLICY=compute or DCB_MODE_POLICY=power, the DCB library uses the RC or PA policy, respectively. DCB_DISABLE=1 disables the DCB library. DCB_PROCESS_ON_DIFFERENT_SOCKETS and DCB_BIND_MODE are used to control the patterns of the core bounds on systems with the NUMA domain. Fig. 2 presents the relationship between the numerical environments and mapping patterns on the NUMA domain. In this figure, there are two sockets and four processes mapped to a node. The four processes have 20, 30, 50, and 60 loads, respectively. Fig. 2(b) presents the mapping pattern with the

DOI: 10.1007/978-3-031-36024-4_30

numerical environments DCB_BIND_MODE=node and DCB_PROCESS_ON_DIFFERENT_SOCKET=1. This is the most active pattern for obtaining balanced loads among the cores. One process is mapped to different sockets depending on the process loads, as illustrated in Fig. 2(b). If users set BIND MODE=socket and DCB_PROCESS_ON_DIFFERENT_SOCKET=0, the process is mapped as displayed in Fig. 2(c). With this pattern, the core bound to the process is immured to the inside of each socket. Fig. 2(d) presents the mapping pattern with the numerical environments DCB_BIND_MODE=node and DCB_PROCESS_ON_DIFFERENT_SOCKET=0. Although this pattern is similar to the most immure pattern (Fig. 2(c)), it permits moving the process among sockets.

3.2 Creation of load-balanced communicator

In this section, we describe applying the approximate solution obtained by SA to the execution of the application. A common approach for changing the mapping of processes to nodes is to spawn processes and move them by internode communication. This approach requires many application-side implementation tasks and a large overhead. Here, we focus on applications where the operations allocated to each process are determined by the rank ID. In these applications, we achieve load balancing among nodes by creating a new MPI communicator that has a new rank ID derived from the solution of the COP. Hereafter, this communicator is referred to as a load-balanced communicator. In the DCB library, load balancing among nodes is achieved by process spawning using the usual MPI functions and the result of the COP. Based on the result of the COP, "*MPI_Comm_spawn_multiple*" is called inside the DCB library to spawn a different number of additional processes on each node. There are unused processes created when launching the program; thus, these processes wait at a barrier synchronization until the end of the program. In addition, one core is allocated as if it were a process not used for the operation, and all processes are allocated to that core. The core bound to unused processes is not mapped to the used processes for computation. Within the DCB library, a new communicator containing all the processes used for computation is generated and returned to the user.

4 Numerical evaluations

In this section, we demonstrate the effectiveness of DCB using numerical evaluations.

4.1 Target application

In this section, we introduce the target application and its load imbalance.

We consider a lattice $\mathcal{H}$-matrix-vector multiplication (LHMVM) using the MPI/OpenMP hybrid programming model for distributed memory systems. Its

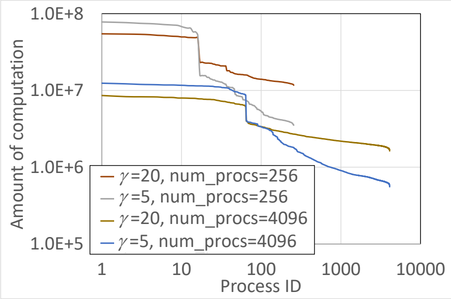


Fig. 3: load imbalance among processes (problem = 1188kp25)

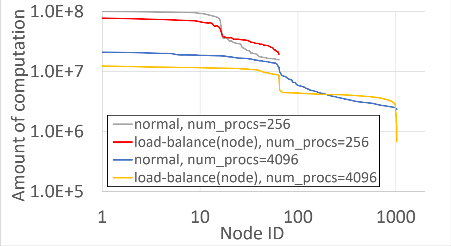


Fig. 4: Result of applying obtained result of simulated annealing(SA) (problem = 1188kp25, parameter $\gamma = 5$)

efficient communication patterns ensure that communication costs remain constant even as the number of MPI processes increases. The parallel scalability of the LHMVM should be significantly improved if the DCB resolves the load imbalance among MPI processes.

The load imbalance of the lattice $\mathcal{H}$-matrix depends on the number of lattice blocks into which the target dense matrix is divided. Basically, the large number of lattice blocks leads the better load balancing. However, the reduction of memory usage and amount of computation due to approximation becomes smaller. Fig.3 illustrates the amount of computation bound to each process in 256 and 4,096 processes parallelization with a particular model(1188kp25), which is approximated by the lattice $\mathcal{H}$-matrix. A parameter γ in fig.3 is used to control the number of lattice blocks. The amount of computation with $\gamma = 5$ is smaller than $\gamma = 20$, but the load imbalance of the $\gamma = 5$ among the node is larger than $\gamma = 20$. This trend is observed regardless of the number of processes. Therefore, the advantage of the lattice $\mathcal{H}$-matrix can be maximized if the load imbalance is absorbed by approximating with smaller γ.

The evaluation was performed by 50 LHMVMs. To use the DCB library in LHMVM, we use the amount of computation executed by each process as the argument l_p. Target problems utilizing the lattice $\mathcal{H}$-matrix are "1188kp25" and "human-1x100"(models for the surface charge method of an electromagnetic simulation)

4.2 Environments and conditions

For the numerical evaluations, we used the Oakbridge-CX (OBCX) and Wisteria /BDEC-01 Odyssey (WO) systems at the Information Technology Center at the University of Tokyo. Tables 1 and 2 present the system specifications and compiler information, respectively. On the OBCX, the turbo boost technology is enabled.

In the numerical evaluations, we set four processes per node to bring out the effectiveness of DCB. This condition is the same in the evaluation without the DCB library. In advance evaluations of the original LHMVM implementation, we have confirmed that the performance of four processes per node is better

Table 1: System specifications

Specifications		Oakbridge-CX	Wisteria/BDEC-01 Odyssey
CPU	Model	Xeon Platinum 8280 (Cascade Lake)	A64FX
	Number of cores	56 (2 Sockets)	48
	Clock	2.7GHz	2.2GHz
	L2-cache	1kB/core	8MB/CMG
Memory	Technology	DDR4	HBM2
	Size	192GB	32GB
	Bandwidth	281.6GB/sec	1,024B/sec
Network	Interconnect	Omni-Path	Tofu Interconnect D
	Topology	Full-bisection Fat Tree	6D mesh / Torus
	Bandwidth	100Gbps	56Gbps

Table 2: Compiler and options on each system

Oakbridge-CX	mpiifort	2021.5.0	-xHost -O3 -ip -qopt-zmm-usage=high
Wisteria/BDEC-01 Odyssey	mpifrtpx	4.8.0 tcsds-1.2.35	-O3 -Kfast,lto,openmp,zfill,A64FX -KARMV8_A,ocl,noalias=s

than that of one process per node. We have confirmed that the performance of FlatMPI is slightly better than that of four processes per node, but we do not use flatMPI. This is because LHMVM has a sequential execution part and it requires a large amount of memory when handling large problems.

To measure the energy consumption of the application, we used the ”powerstat” command on OBCX and the PowerAPI [7] library on WO. The result of energy consumption shows a sum of CPUs and memory in the nodes in OBCX, and the sum of all modules in the nodes in WO. In the WO system, before measuring the energy consumption and computation time, we enabled the retention feature of every core. The retention feature provides a type of deep sleep in an A64FX CPU.

4.3 Effectiveness of the load-balancing among nodes using SA

We evaluated the effect of load balancing among nodes by applying the approximate solution of SA. Fig. 4 illustrates the effectiveness of load balancing from the approximate solution in the 1188kp25 problem with 256 and 4,096 processes and parameter $\gamma = 5$. By applying the approximate solution of SA, the maximum load was reduced for each condition. We observed similar effectiveness for the other conditions and problems.

When applying the approximate solution of SA, the method of generating the load-balanced communicators described in Section 3.2 was used. Then, we executed LHMVM with one process per node initially, and spawned the required processes when creating the load-balanced communicator. In the WO system, the Fujitsu MPI does not support a non-uniform number of processes spawning. Thus, we spawned the same number of processes on all nodes corresponding to the node requiring the most number of processes.

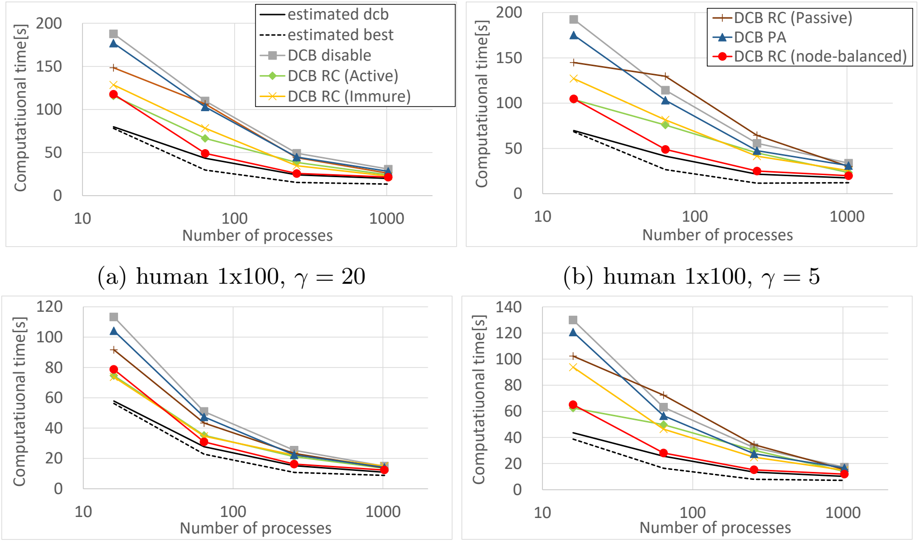
4.4 Performance improvement

Figs. 5 and 6 display the computation time of 50 LHMVMs with and without the DCB library, including the results using the load-balanced communicator (denoted as "RC (node-balanced)" in the figures). The core-binding pattern used for execution with the load-balanced communicator was the active pattern (Fig. 2(b)). The figures also display the estimated computation time with the DCB library (2) and the best cases (denotes no load imbalances among nodes). As the computation node of OBCX is a two-socket NUMA domain, we evaluated all mapping patterns of the cores on the node to the processes in Fig. 2. For WO, we only evaluated the active patterns. For all problems and parameters on all systems, the DCB library improved the computation performance. The performance improvement of DCB tended to be larger at smaller-parallelism and smaller at larger-parallelism. This is because the performance in the high-parallel condition was degraded by the load imbalance among nodes. When using the load-balanced communicator, the further performance improvement was obtained on OBCX. The effectiveness of the load-balanced communicator was maximum at 64 processes, and the computation time with the load-balanced communicator was close to the estimated time on OBCX. By using the DCB library without the load-balanced communicator, we achieved a performance improvement of 31%–52% for 16 processes and 9%–31% for 1,024 processes. With the load-balanced communicator, we achieved a performance improvement of 39%–58% compared with the result without DCB. Compared with using DCB (active), we achieved a 33%–50% performance improvement for 256 processes. However, on WO, we could not achieve performance improvement using the load-balanced communicator with high-parallel conditions. This was due to the increase in communication overhead, as illustrated in Fig. 7. The communication pattern among the processes became complex when applying the load-balanced communicator. The communication pattern among processes was not considered when we constructed the COP of load balancing among nodes. In particular, the network topology of WO is a six-dimensional torus, and the communication overhead is large in complex communication patterns.

We also examined the effect of the DCB library on the parameter γ of the lattice $\mathcal{H}$-matrix. Without the DCB library, the computation time with $\gamma = 5$ was longer than that with $\gamma = 20$ in spite of lower memory usage and amount of computation. This was due to load imbalance, as displayed in Fig. 3. In contrast, using the DCB library, the computation time with $\gamma = 5$ was shorter than or similar to that with $\gamma = 20$. This indicates that the use of the DCB library improved the benefits of the lattice $\mathcal{H}$-matrix.

The computation time of the PA policy was expected to be similar to that in conditions without the DCB library. In OBCX, the computation time of the PA policy was slightly lower than that without DCB. This was because the number of cores used for each node was reduced. This created a margin for memory access, and resources for accessing memory by processes that had a large load were obtained. In WO, the computation time of the PA policy was similar to that without DCB, as we expected.

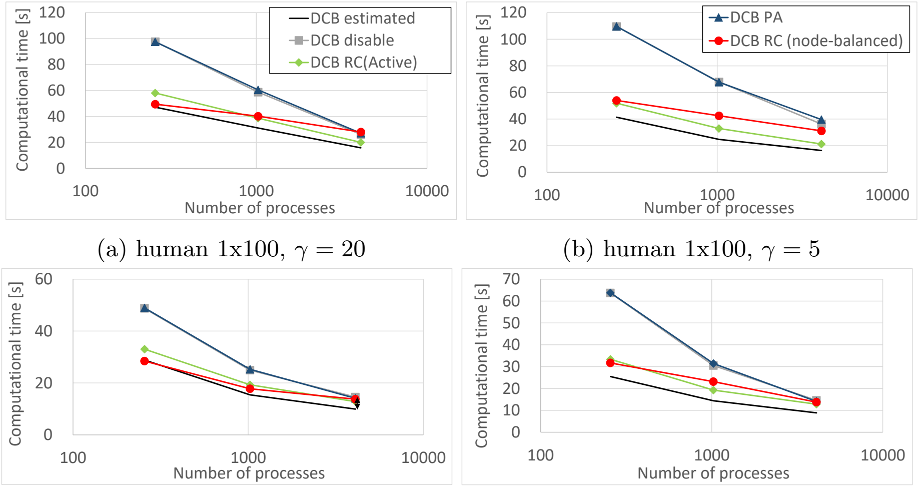
DOI: 10.1007/978-3-031-36024-4_30



(a) human 1x100, $\gamma = 20$ (b) human 1x100, $\gamma = 5$

(c) 1188kp25, $\gamma = 20$ (d) 1188kp25, $\gamma = 5$

Fig. 5: Computatinal time of LHMVM with DCB on Oakbridge-CX



(a) human 1x100, $\gamma = 20$ (b) human 1x100, $\gamma = 5$

(c) 1188kp25, $\gamma = 20$ (d) 1188kp25, $\gamma = 5$

Fig. 6: Computation time of LHMVM with DCB on Wisteria/BDEC-01 Odyssey

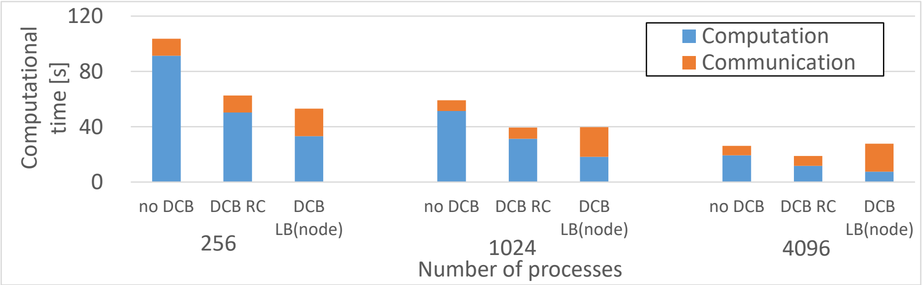
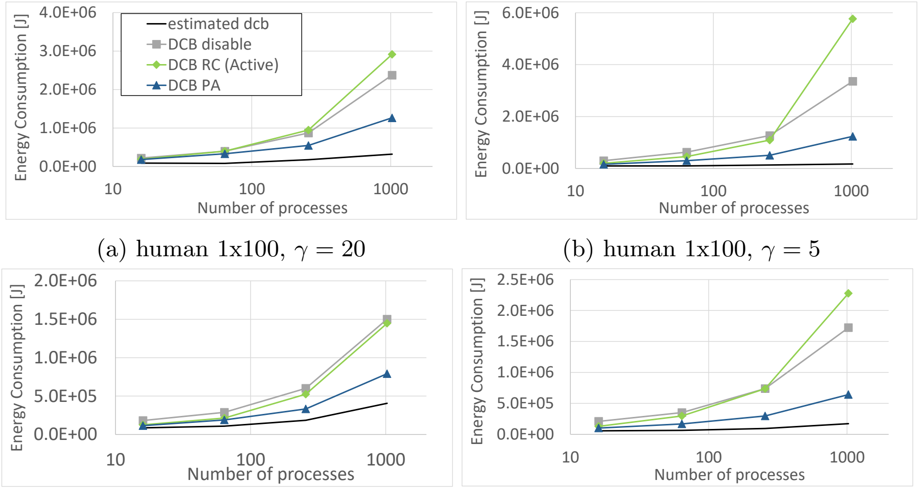


Fig. 7: Breakdown of execution time in WO (1188kp25, $\gamma = 20$)

DOI: 10.1007/978-3-031-36024-4_30



(a) human 1x100, $\gamma = 20$

(b) human 1x100, $\gamma = 5$

(c) 1188kp25, $\gamma = 20$

(d) 1188kp25, $\gamma = 5$

Fig. 8: Energy consumption of LHMVM with DCB on Oakbridge-CX

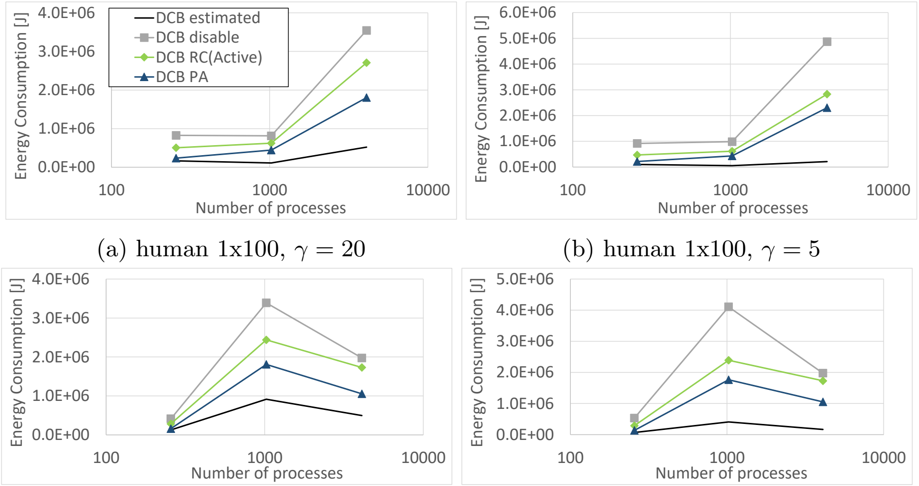
4.5 Reducing energy consumption

This section evaluates the effectiveness of the PA policy in reducing energy consumption. Figs. 8 and 9 display the energy consumption for the number of processes in each system. A reduction in energy consumption using the PA policy of the DCB library was observed in all conditions. In OBCX, the energy consumption tended to decrease with high parallelism. This was because the load imbalance among processes increased with high parallelism, and many cores were not used for computation due to the DCB library. In OBCX, we achieved a 47%–63% reduction in energy consumption with 1,024 processes. In WO, we achieved a 61%–77% reduction in energy consumption with 256 processes and a 46%–56% reduction in energy consumption with 4,096 processes.

In OBCX, the energy consumption of the RC policy of the DCB library was higher than that without the DCB, even though the computation time was shorter. We are assumed to be due to the turbo boost. That is also why the energy consumption with the PA policy was higher than estimated. In WO, the energy consumption was higher than estimated because the energy consumption of the network modules and assistant cores was not considered.

5 Conclusion

In this paper, we propose the DCB approach to reduce load imbalance among processes. By this approach, we can expect to reduce the computation time and/or energy consumption for applications for which load balancing is difficult. In the DCB environment, the load imbalance among processes is rectified by changing the number of cores bound to each process to equalize the loads of the cores. We also use SA to consider load balancing among nodes, which cannot



(a) human 1x100, $\gamma = 20$ (b) human 1x100, $\gamma = 5$

(c) 1188kp25, $\gamma = 20$ (d) 1188kp25, $\gamma = 5$

Fig. 9: Energy consumption of LHMVM with DCB on Wisteria/BDEC-01 Odyssey

be achieved using DCB. The results of applying DCB to the lattice $\mathcal{H}$-matrix demonstrate a reduction in computation time of more than 50% and a reduction in energy consumption of more than 70% for OBCX and WO.

In this study, core binding to processes was determined based on the parameters received from the user in the DCB library. In addition, the number of using cores is based on the ideal condition that the bottleneck is only load balancing. In practice, it is necessary to consider the computer architecture and NUMA. However, it is difficult to create a realistic model that takes into account the complexity of the computer architecture and the characteristics of the applications. Therefore, in the future, we will study an algorithm for automatically determining core binding based on a CPU performance counter and other information. The load of each process with the lattice $\mathcal{H}$-matrix was determined when the matrix was generated. There are many applications in which the process loads are changed dynamically in runtime, and we will consider applying DCB to these applications. Then, the small overhead of changing the core binding is important to minimize; therefore, we will attempt to reduce this overhead by examining system calls and other aspects.

For load balancing among nodes, we did not consider the communication patterns among processes. Therefore, we could not achieve performance improvement using the load-balanced communicator. In future work, we will consider the communication patterns among processes and the network topology of systems to improve the effectiveness of DCB. We will also investigate an approach for automatically changing the process mapping to nodes to support dynamic load balancing. For this investigation, we will refer to research on fault tolerance.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 18K18059, 21H03447, and 19H05662. This work is also supported by ”Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures (JHPCN)” in Japan (Project ID: jh230058).

References

1. Attiya, I., et al.: Job scheduling in cloud computing using a modified harris hawks optimization and simulated annealing algorithm. Computational intelligence and neuroscience **2020** (2020)
2. Corbalan, J., et al.: Dynamic load balancing of mpi+openmp applications. In: International Conference on Parallel Processing, 2004. ICPP 2004. pp. 195–202 vol.1 (2004)
3. Curtis-Maury, M., et al.: Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In: Proceedings of the 20th annual international conference on Supercomputing. pp. 157–166 (2006)
4. Curtis-Maury, M., et al.: Prediction-based power-performance adaptation of multithreaded scientific codes. IEEE Transactions on Parallel and Distributed Systems **19**(10), 1396–1410 (2008)
5. Garcia, M., et al.: A dynamic load balancing approach with smpsuperscalar and mpi. In: Facing the Multicore-Challenge II, pp. 10–23. Springer (2012)
6. Garza-Santisteban, F., et al.: A simulated annealing hyper-heuristic for job shop scheduling problems. In: 2019 IEEE Congress on Evolutionary Computation (CEC). pp. 57–64 (2019)
7. Grant, R.E., et al.: Standardizing power monitoring and control at exascale. Computer **49**(10), 38–46 (Oct 2016)
8. Hiroshi, N., et al.: Third generation digital annealer technology (2021). URL https://www.fujitsu.com/jp/documents/digitalannealer/researcharticles/DA_WP_EN_20210922. pdf
9. Ida, A.: Lattice $\mathcal{H}$-matrices on distributed-memory systems. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 389–398 (2018)
10. Iwashita, T., et al.: Software framework for parallel bem analyses with h-matrices using mpi and openmp. Procedia Computer Science **108**, 2200–2209 (2017)
11. Klinkenberg, J., et al.: Chameleon: reactive load balancing for hybrid mpi+ openmp task-parallel applications. Journal of Parallel and Distributed Computing **138**, 55–64 (2020)
12. Korte, B.H., et al.: Combinatorial optimization, vol. 1. Springer (2011)
13. Li, D., et al.: Strategies for energy-efficient resource management of hybrid programming models. IEEE Transactions on parallel and distributed Systems **24**(1), 144–157 (2012)
14. Nakajima, K., et al.: Communication-computation overlapping with dynamic loop scheduling for preconditioned parallel iterative solvers on multicore and manycore clusters. In: 2017 46th International Conference on Parallel Processing Workshops (ICPPW). pp. 210–219 (2017)
15. Suleman, M.A., et al.: Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on cmps. ACM Sigplan Notices **43**(3), 277–286 (2008)
16. Zaman, M., et al.: Pyqubo: Python library for mapping combinatorial optimization problems to qubo form. IEEE Transactions on Computers **71**(4), 838–850 (2022)