The Journal of Supercomputing
https://doi.org/10.1007/s11227-021-03829-x

# Design of a simulation model for high performance LINPACK in hybrid CPU-GPU systems

**Yichang Hu**[1] · **Lu Lu**[1]

Accepted: 18 April 2021
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

**Abstract**
High performance LINPACK (HPL) benchmark is used to evaluate the maximum floating-point performance of a computer cluster. Since the performance of the graphics processing unit (GPU) has been improved rapidly, many researchers start to optimize HPL benchmark through GPU to maximize system utilization. Nevertheless, it is difficult to determine the optimal combination of parameters in this process due to the complexity of the input. Therefore, running HPL on a heterogeneous system is time-consuming and is not flexible under different hardware components. So we propose a simulation model of HPL in this paper. The model is no longer limited by hardware components and able to simulate the execute process of HPL across different computing node in heterogeneous GPU-enhanced clusters at any scale. It can also assist researchers in evaluating the floating-point performance quickly and provide a reference for the hardware investment.

**Keywords** High-performance LINPACK · Heterogeneous systems · GPU acceleration · Simulation

## 1 Introduction

In modern scientific research, high performance computing (HPC) has played an important role in various fields, including biology, e-commerce, finance, insurance, and so on. As the third scientific method and the primary productive force, the status of HPC is widely recognized. It has been widely used not only in computing science, but also in petroleum exploration, molecular materials and countless fields.

✉ Lu Lu
lul@scut.edu.cn

Yichang Hu
sxy05100415@126.com

[1] School of Computer Science and Engineering, South China University of Technology, Guanzhou 510000, China

Published online: 03 May 2021



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

LINPACK benchmark is a popular benchmark for assessing the performance of a given computing system. The largest and fastest supercomputers in the world are ranked twice a year on the TOP500 list. Among the benchmarks that are often used to evaluate those supercomputers, the LINPACK (HPL) implementation [7] has emerged as the de-facto standard benchmark, although other benchmarks, such as HPCG and HPGMG, have recently been proposed to become the new standards.

Today, machines with more than 100,000 cores are prevailing, and several machines beyond the 1,000,000 cores are already in production. With core-counts beyond the 100,000, an optimization of HPL parameters (problem size, grid arrangement, granularity, collective operation algorithms, etc.) specifically suited to the network topology and performance is essential. Such optimization can be particularly time-consuming and can hardly be done through simple mathematical performance models. Furthermore, to yield the best benchmark results, runtimes (such as Open Performance prediction of Message Passing Interface [9, 13]) and supporting libraries (such as Basic Linear Algebra Subprograms) need to be fine-tuned and adapted to the underlying platform. Performance results of an already current-generation machine typically play a role in the funding process for future machines. The optimal performance for the deployed machine is hence considered critical for HPC designers and vendors. We can benefit from being able to tune parameters without actually running the benchmark for hours. Therefore, it is necessary to establish a reliable simulation model to simulate the operation in multi-GPUs and CPUs clusters [18] and predict the performance of the HPL by a slight modification of parameters. The model can be considered as a reference to simulate the peak performance in the process of design and implementation of a machine. In this paper, we will explain how to establish the model through the HPL algorithm and estimate the running time and simulate the execution flow. Finally, the prediction will be compared with the actual results of HPL, and the error will be got in order to prove the feasibility of the simulation model.

In the remainder of this paper: Sect. 2 describes some existing related work, and Sect. 3 introduces HPL benchmark briefly. Then, in Sect. 4, we propose our HPL simulation model and the approach used for the model, including the different models of the single node and multiple nodes. Next, the simulation results and the evaluation will be presented in Sect. 5. Finally, we offer the conclusion and discuss the future work in Sect. 6.

## 2 Related works

Performance prediction of message passing interface (MPI) application through simulation has been widely studied over the last decades. Since the HPC systems have been scaled up, it brought challenges to the debugging and tuning of large-scale parallel programs, especially in an exascale supercomputer. Bigsim [29] , proposed by Zheng et al., is used to predict the performance of a computing system that has a large number of processors. Mubarak et al. develop a tool named CODES [22] to enable the exploration of HPC network and storage system design and present a high-fidelity and scalable model. This approach allows researchers to study



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

the behavior of MPI applications directly. The SST Marco [1] developed by Janssen et al. is an instrument to simulate MPI applications through online simulation. Degomme et al. [8] discuss several problems that may be faced in creating computationally efficient simulations and validated the network models used by SMPI, a flexible simulator of application, through a series of experiments. Cornebize et al. [6] put forward a methodology that can capture the complexity of adaptive applications by emulating the MPI code while skipping insignificant parts. Lin et al. [17] propose an emulation system that supports debugging and tuning of large-scale parallel programs by executing parallel programs in the desired scale on a small cluster. The system supports popular GPU-enhanced heterogeneous architecture. With the help of the emulation system, programmer will not execute the program in a specified scale repeatedly and no longer be limited by the inflexibility of the debugging.

Concerning HPL benchmark, a variety of optimizations have been carried out. Chen et al. [4, 25] present an energy-efficient implementation of dense lower-upped (LU) factorization, which is a critical part of LINPACK benchmark, to avoid a large number of data transfers via Peripheral Component Interconnect-Express (PCI-E). Gan et al. [10] propose OAMM (the orchestrating algorithm for matrix multiplication) to improve the efficiency of the heterogeneous system composed of CPU and China accelerator. Besides, virtualization in HPC has been much exploited by researchers, and the overhead in the virtualization layer is the critical bottleneck in the HPC environment [27]. Martin et al. [19] explore the feasibility of this interoperable Rkt container in high performance applications. Mohammadi [21] also demonstrate the viability of the cloud for HPC applications. It can be concluded that more and more environments depend on HPL to evaluate the system performance. However, it is hard to build a hardware platform, especially for an exascale supercomputer [24]. Thus, we analyze the algorithm of HPL, and construct a hardware-unrelated model to simulate the execution process of HPL benchmark.

In recent years, the method of HPL prediction by simulation model has also been explored. Zhang et al. [28] establish the simulation models of HPL based on a large number of natural experiments and find the rule of determining block size NB theoretically, so as to reduce the experimental cost. Cornebize et al. [5] also simulate HPL benchmark on a large-scale server to predict the actual performance on supercomputers because the financial cost of running HPL on supercomputers for several hours is unbearable. Zhao et al. [11] compare the prediction of their model with the results of the Sugon advanced computing system and indicate that several factors such as the ratio of matrix operations to HPL calculation, the utilization of matrix operation library functions and the network transmission can broadly reflect the calculation efficiency of the HPL of the supercomputing system.

Although these studies have been proven to be efficient, the research on the simulation model for HPL is not enough. In this paper, we will introduce a design of simulation model for HPL and establish the simulation model according to different network types and node numbers to predict the results of HPL.



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

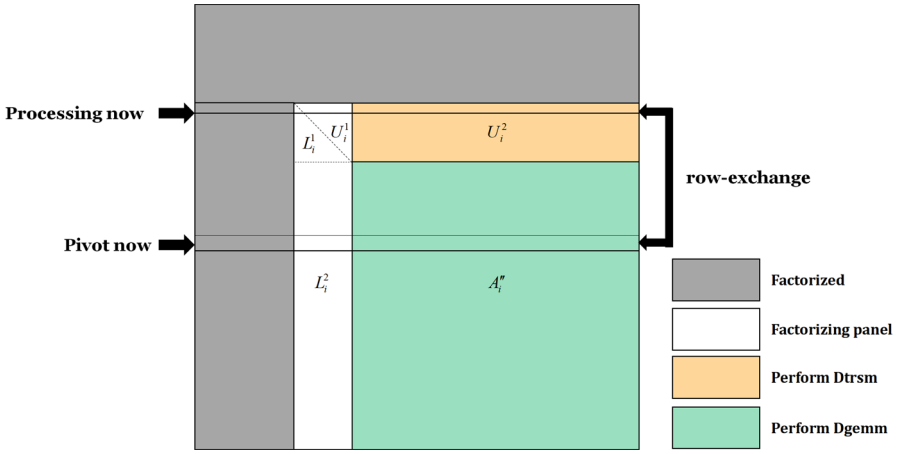



**Fig. 1** Block LU decomposition

## 3 HPL algorithm

HPL is regarded as a standard implementation of LINPACK benchmark, which requires MPI to implement a LU decomposition, i.e., a factorization of a square matrix $A$ as the product of a lower triangular matrix $L$ and an upper triangular matrix $U$ [20].

It randomly generates and solves a system of linear algebraic equations with iteration method of the form:

$$Ax = b;\ A \in \mathbb{R}^{N\times N};\ x, b \in \mathbb{R}^N \tag{1}$$

by calculating LU decomposition iteratively with row partial pivoting. HPL contains four steps generally: Factorization, Broadcast, Swap, and Update. HPL checks the correctness of the factorization by solving a linear system, but only the factorization step is benchmarked. Figure 1 shows the working principle of the factorization. It consists of a series of panel factorizations which are followed by an update of the trailing sub-matrix. From the figure, the current factorizing panel $L_i$ will be decomposed into three submatrices $L_i^1$, $U_i^1$, and $L_i^2$ in each iteration and, the block-cyclic data distribution of $A$ is employed in HPL.

The main parameters of HPL are listed as follow:

- $N$ is the order of the square matrix $A$.
- $NB$ is the size of a block. The coefficient matrix is divided into $NB \times NB$ blocks.
- $P$ and $Q$ define the number of process rows and the number of process columns, respectively. All the blocks are distributed onto a two-dimensional $P \times Q$ grid.
- *REACT* determines the panel factorization algorithm. HPL provides three kinds of algorithms (left-looking, right-looking and Crout).
- *SWAP* is the algorithm for swapping while pivoting.



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

– *BCAST* is used for the step of broadcast. After the step of factorization and swap, HPL will broadcast the panel of columns to the other process columns.
– *DEPTH* controls the iteration number of the outer loop which can overlap with each other.

As the decomposition proceeds, the augmented matrix is decomposed into the product of the lower triangular matrix $L$ and the upper triangular matrix $U$. Therefore, it is easy to solve $x$ by the equation $Ux = y$. In each iteration, only the step of factorization and update involve double-precision floating-point operations. And these two parts are the main parts of solving linear equations, which take up more than 90% of the running time. The time complexity is calculated as:

$$T(N) \approx \frac{\frac{2}{3}N^3 + 2N^2}{P \cdot W \cdot w} + \theta((P+Q) \cdot N^2) \tag{2}$$

where $w$ is the flop rate of a single node and the second term corresponds to the communication overhead, which is affected by the network capacity and the parameters.

## 4 Design of the simulation model

In this section, we propose the process of the simulation, including the different models of single node and multiple nodes. In addition to time-saving, the researchers can detect the bottleneck of the heterogeneous system by altering the hardware environment and comparing the predictions [23], because HPL reflects the peak performance of the HPC system and the bottleneck can not be observed easily. Without the simulation model, it is likely that the researchers need to reinstall the driver or even the entire operating system before running the HPL benchmark, which is extremely time-consuming. Beyond that, some hardware, such as PCI-E bus, CPU slot or DRAM slot, cannot be changed once the motherboard is manufactured. Therefore, if the bottleneck lies in the unchangeable hardware, different types of motherboards are required. With the assistance of the simulation models, the issues discussed above no longer exist.

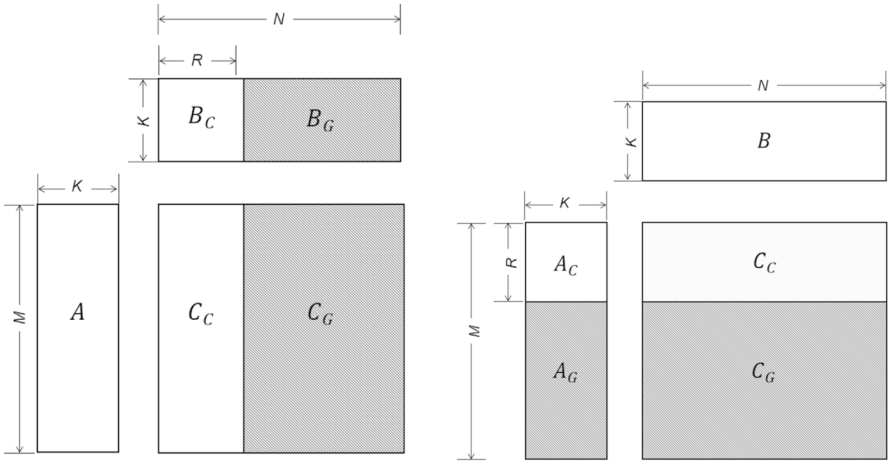
### 4.1 HPL accelerated by GPU

The optimized version of HPL is mainly completed by CPU and GPU to accelerate. Taking matrix multiplication $C = A \times B$ as an example, the matrix division is shown in Fig. 2, which is divided into two cases: 2(a) (vertical-cut $B$) and 2(b) (cross-cut $A$). In both cases, $A$, $B$, $C$ are $M \times K$, $K \times N$, and $M \times N$ dense matrices.

In HPL, there are two different situations of the $M$, $N$ and $K$. One case is that $K$ is small, and $M$ and $N$ are almost the same size. The system will achieve better performance when $B$ is cut vertically. The other case is that $K$ is equal to $N$ and they are small, but compared with $K$, $M$ has a larger scale. In this case, crosscutting $A$ will be a better solution. Since the large gap [26] between the computing power of the CPU and GPU, the proportion of computation that the CPU takes up



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com




**(a)** vertical-cut $B$ **(b)** cross-cut $A$

**Fig. 2** The matrix division

will influence the overall performance. $R$ is the ratio of the transverse or vertical tangent of the matrix and the workload ratio of CPU and GPU can be modified by adjusting the scaling factor. However, due to the data communication overhead between the CPU and GPU, it is necessary to balance CPU and GPU workload. In the best case, the run time on the CPU contains the transmission time of the data from the CPU to the GPU, the computation time of the GPU and the transmission time of the data from the GPU back to the CPU.

In order to make the scaling factor R for the best effect, it is essential to get the working time of CPU and GPU as close as possible. Ideally, when

$$R = \frac{\text{GFLOPS}_{\text{GPU}}}{\text{GFLOPS}_{\text{GPU}} + \text{GFLOPS}_{\text{CPU}}} \tag{3}$$

where $\text{GFLOPS}_{\text{CPU}}$ and $\text{GFLOPS}_{\text{GPU}}$ are the theoretical speed of CPU and GPU in floating operation, the workload balancing of the system can be optimal. When solving $Ax = b$, the time is mainly spent on the general matrix-matrix multiply (DGEMM) and triangular solve matrix (DTRSM), which can be executed on CPU or GPU selectively. Therefore, R can be calculated according to the theoretical speed of CPU and GPU in floating operation, and then the workload of DEGMM and DTRSM will be allocated to GPU and CPU. Afterward, the HPC system can reach the highest theoretical peak.

Assuming that the ratio of the operation speed between CPU and GPU is $\frac{1}{8}$, $\frac{8}{9}$ of the workload should be assigned to GPU and the rest to CPU. However, considering the overhead of data transmission between CPU and GPU, the transmission time is set as $t_\tau$ and the total workload is set as W. We can get the following formula:



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

$$\mathrm{GFLOPS}_{\mathrm{CPU}}(t_{\mathrm{CPU}} + t_{\tau}) + \mathrm{GFLOPS}_{\mathrm{GPU}} t_{\mathrm{GPU}} = W \tag{4}$$

Then,

$$R = \frac{W - \mathrm{GFLOPS}_{\mathrm{CPU}} \cdot t_{\tau}}{\mathrm{GFLOPS}_{\mathrm{CPU}} + \mathrm{GFLOPS}_{\mathrm{GPU}}} \cdot \mathrm{GFLOPS}_{\mathrm{GPU}} \tag{5}$$

The total amount of data transmitted is (calculated by double-precision floating-point) as follows:

$$D = \sum_{k=1}^{\left\lfloor \frac{N}{NB} \right\rfloor} 64 \cdot [(N - k \cdot \mathrm{NB}) \cdot NB \cdot (1 + R) + (N - k \cdot NB)^2 \cdot R] \tag{6}$$

If the bandwidth of PCI-E is $B_{\mathrm{PCI}} - E$, then

$$t_{\tau} = \frac{D}{B_{\mathrm{PCI}-E}} \tag{7}$$

and the scaling factor R is as follows:

$$R = \frac{W - \mathrm{GFLOPS}_{\mathrm{CPU}} \cdot \dfrac{D}{B_{\mathrm{PCI}-E}}}{1 + \dfrac{\mathrm{GFLOPS}_{\mathrm{GPU}}}{\mathrm{GFLOPS}_{\mathrm{CPU}}}} \Bigg/ W \tag{8}$$

So the GPU workload is $R \cdot W$, and the CPU workload is $(1 - R) \cdot W$.

### 4.2 HPL single node simulation model design

Despite the complexity of the algorithm, the most time-consuming operations are GEMM, TRSM, and data transmissions between CPUs (and GPUs if the system has), which account for about 95% of the total computation time. GEMM and TRSM operations consist of multiple floating-point adding and multiplying, and the number of adding and multiplying operations is fixed once the problem size is given.

For each adding or multiplying operation, the run time is fixed simultaneously and can be calculated if the computing power of the CPUs (and GPUs) is available. Similarly, the time of all data transmissions between CPUs (and GPUs) can be calculated if the bandwidth of the data transmission bus is given.

Thus, the total time it takes for a HPC system to run HPL benchmark can be approximate to the total time of GEMM and TRSM operations along with the data transmission time. So the floating-point power of the system can be measured approximately. We design an algorithm that can simulate the process of GEMM, TRSM and data transmission. The main steps of the single node HPL simulation model are shown in Fig. 3.

The following is a rough simulation of HPL in a single node:



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

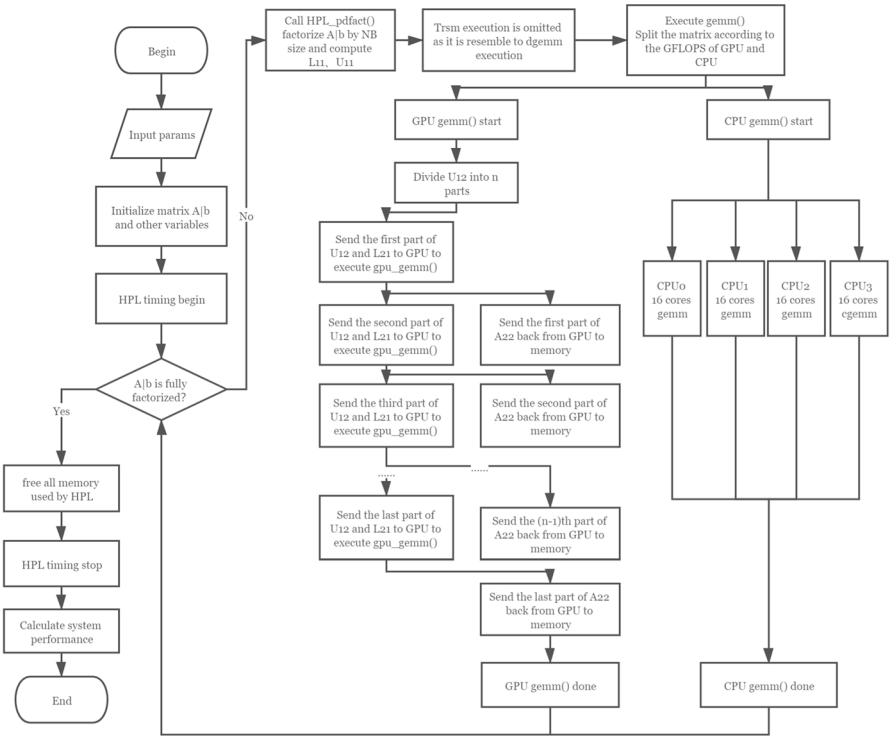



**Fig. 3** Flowchart of the single node model

1. Read parameters from input.
2. Initialize *A*|*b* according to the parameters.
3. HPL timing begins.
4. Factorize *A*|*b* by NB size [2].
   ——loop i:

   (1) HPL_pdpanel_pdfact()
   time to factorize *A*|*b*, this part is neglect since it involves no double-precision floating-point operations and do not take much time.
   (2) HPL_pdupdate()
   compute $A_{12}$, $A_{21}$, $A_{22}$ and it is consist of the following two parts: HPL_dtrsm() & HPL_dgemm()
   Since the parallel computation of HPL_dtrsm() resembles HPL_dgemm(), the part of HPL_dtrsm() is omitted.
   (3) Split the matrices that HPL_dgemm() needs into two parts according to CPU FLOPS and GPU FLOPS (the theoretical performance).
   (4) The part of $U_{12}$ used by GPU is split into $n$ parts.
   (5) Send $L_{21}$ and the first part of $U_{12}$ to GPU to perform gemm()
   how to perform parallel compute depends on BLAS



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

(6) Send $L_{21}$ and the second part of $U_{12}$ to GPU to perform gemm() meanwhile send the first part of $A_{22}$ to memory.
(7) ...
(8) Send $L_{21}$ and the last part of $U_{12}$ to GPU to perform gemm() meanwhile send the $(n-1)$th part of
(9) Send the last part of $A_{22}$ to memory.
GPU gemm() done.
Since the factorization size for each loop is the same, the $A_{22}$ matrix computed at each loop is of the same size.
And generally, $A_{22}$ matrix is larger than $L_{21}$ and $U_{12}$, so the time to transfer $A_{22}$ is longer than $L_{21}$ and $U_{12}$, so the paralleled time for each gpu_gemm() loop is:
Time to transfer $A_{22} + n$ times of gpu_gemm() time $t_{gpu\_gemm\_partial_i}$
That is $t_{gpu\_gemm_i} = t_{A_{22_i}} + n \cdot t_{gpu\_gemm\_partial_i}$

5. Factorization of $A|b$ is finished. The most time-consuming part of HPL is done, so the rest of the time is neglected.
6. Timing ends, the total of HPL time can be calculated as follows
$t_{HPL} \approx \sum_i \max(t_{gpu\_trsm_i}, t_{cpu\_trsm_i}) + \max(t_{gpu\_gemm_i}, t_{gpu\_gemm_i})$
7. Calculate the system performance according to the total time.

### 4.3 HPL multiple nodes simulation model design

The multiple nodes [3] simulation model follows the single-node simulation model in the time computation like GEMM and TRSM, but the communication overhead has to be re-considered because of the communication across multiple nodes. Currently, the model simulates the process of HPL performed on multiple nodes, and the cluster does not have a leading node and all compute nodes are treated equally. The model can simulate two kinds of internet connections: Ethernet and InfiniBand (IB) [12]. The model also involves five types of network topology: Star, Bus, Mesh, Ring, and Tree topology.

The main difference from the single node simulation model is the cross-node communication overhead. In single node simulation model, the communication is performed within the computation node, mainly by using the shared memory. However, in the case of multiple nodes, the communication is performed across the entire cluster via network devices. Thus, the bandwidth and latency are vital to the efficiency of the whole cluster.

In our model, we choose TCP/IP as the network protocol to calculate the overhead of Ethernet. In TCP/IP, the physical layer requires 8 bytes for each MAC data frame. For each MAC data frame, it uses 6 bytes for destination MAC address, 6 bytes for source MAC address, 2 bytes for frame type and 4 bytes for Frame Check Sequence. So each data frame in the MAC layer takes extra 18 bytes overhead. Moreover, the Ethernet requires a minimum gap of 12 bytes between two data frames. In the Transmission Control Layer, each TCP segment requires 20 bytes for the segment header. Thus, each data frame can hold maximum data



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

of [MTU − (8 + 18 + 12 + 20)] bytes, where MTU is the Maximum Transmission Unit of Ethernet. In most cases, MTU is 1500 (bytes).

Moreover, the Ethernet contains slot time before sending a data frame. So the network overhead of Ethernet can be formulated by:

$$\text{Ethernet overhead} = \text{Frames} \cdot t_{\text{slot}} + [8 \cdot \text{Frames} \cdot (8 + 18 + 12 + 20) + D] \cdot \frac{1}{B} \tag{9}$$

$$\text{Frames} = \left\lceil \frac{D}{\text{MTU} - (8 + 18 + 12 + 20)} \right\rceil \tag{10}$$

$$t_{\text{slot}} = \frac{512}{B} \tag{11}$$

where $B$ denotes the bandwidth of Ethernet, $D$ denotes the data size that needed to be transferred, $t_{slot}$ denotes the Ethernet slot time, and *Frames* denotes the total frames that needed to be transferred. The IB provides high bandwidth and low latency internet connection for supercomputers. Unlike Ethernet, the network protocol is handled by the IB adapters so IB can provide higher bandwidth and lower latency. In our model, we calculate IB overhead by the following equation:

$$\text{IB overhead} = \frac{D}{B \cdot \eta} \tag{12}$$

$$\eta = \frac{\text{Data Rate}}{\text{Signaling Rate}} \tag{13}$$

$$B = \text{Data Rate} \cdot \text{Virtual Links} \tag{14}$$

where $\eta$ denotes the efficiency of IB. The SDR, DDR, and QDR versions use 8/10 encoding, so effective throughput for these is lowered to 80%. From FDR version, IB uses 64/66 encoding, allowing a higher effective throughput, and the signaling rate ratio can be 96.97%.

Despite the network media, the topology of the network may also affect the network overhead. In our model, we consider five types of topology. We compute a topology factor for each topology, and the total network overhead is $t \times \text{factor}_{\text{topology}}$, where $t$ is the Ethernet or IB overhead. The topology factor for each network topology is defined as follows:

- *Star*: In this type of network topology, all nodes are connected to the same network switch. Because the data frames need to be retransmitted by the switch, each data frame will be transmitted twice in a star network, so the topology factor for a star network is 2.
- *Bus*: In this type of network topology, all nodes are connected using the same network bus. If all nodes want to transfer data, they have to do it one after



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

another because the bus can only be obtained by one node at a time. So the topology factor for a bus network is equal to the number of nodes.
- *Mesh*: In this type of network topology, all nodes are fully connected. So each data frame will be sent only once, and the topology factor for a mesh network is 1.
- *Ring*: A ring network is a network topology where each node connects to two other nodes. So the maximum communication time it takes depends on the two nodes that have the farthest distance. Data frames transmitted between the two nodes will be transmitted $\left\lfloor \frac{\text{nodes}}{2} \right\rfloor$ times. So the topology factor is $\left\lfloor \frac{\text{nodes}}{2} \right\rfloor$.
- *Tree*: Since there are many ways to form a tree topology, so in our model, we only take a complete binary tree as the tree topology. In a complete binary tree, the farthest distance between two nodes is $2 \cdot h$, where $h$ denotes the height of the tree. And $h = \lfloor \log_2 \text{nodes} \rfloor$. So the topology factor is $h = 2 \cdot \lfloor \log_2 \text{nodes} \rfloor$.

The cross-node communication data size in each factorization step is $NB \cdot (N - i \cdot NB)$ doubles. Because each node owns $\frac{1}{\text{nodes}}$ of the whole matrix, so each node only needs to broadcast $\frac{NB \cdot (N - i \cdot NB)}{\text{nodes}}$ doubles at each factorization step.

If the cluster uses Ethernet, the whole network overhead is:

$$\left\lceil \frac{8 \cdot \frac{NB \cdot (N - i \cdot NB)}{\text{nodes}}}{\text{MTU} - 58} \right\rceil \cdot \frac{512}{B} + \left\lceil 8 \cdot \left\lceil \frac{8 \cdot \frac{NB \cdot (N - i \cdot NB)}{\text{nodes}}}{\text{MTU} - 58} \right\rceil \cdot 58 + \frac{NB \cdot (N - i \cdot NB)}{\text{nodes}} \right\rceil \cdot \frac{1}{B} \tag{15}$$

If the cluster uses IB, the whole network overhead is:

$$\frac{8 \cdot \frac{NB \cdot (NB \cdot (N - i \cdot NB))}{\text{nodes}}}{\text{Data Rate} \cdot \text{Virtual Links}} \cdot \frac{\text{Signaling Rate}}{\text{Data Rate}} \tag{16}$$

Figure 4 is the flowchart of the multiple nodes HPL simulation model. We can find that it is almost the same as that of the single node simulation model, only the communication overhead is added.

## 5 Evaluation

To evaluate and verify the reliability of the HPL simulation model, we conduct a series of experiments in this section to compare the actual performance of HPL on a heterogeneous system that is equipped with multi-cores and multi-GPUs. We published the simulation model on our website[1] and the model takes some hardware parameters as input and outputs the system performance and efficiency.

[1] Single-node HPL simulation model: http://www.i-test.com.cn/hpc/singleNode.html.
Multi-node HPL simulation model: http://www.i-test.com.cn/hpc/multiNode.html.



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

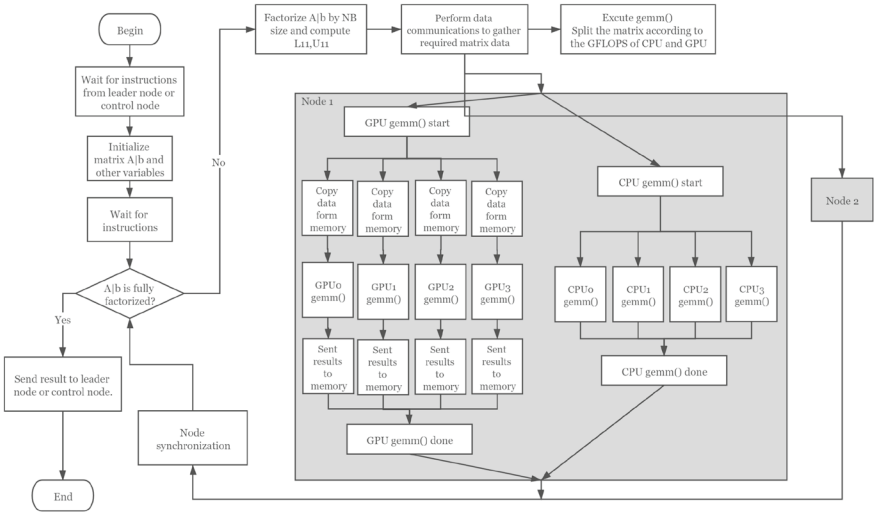



**Fig. 4** Flowchart of the multiple nodes model

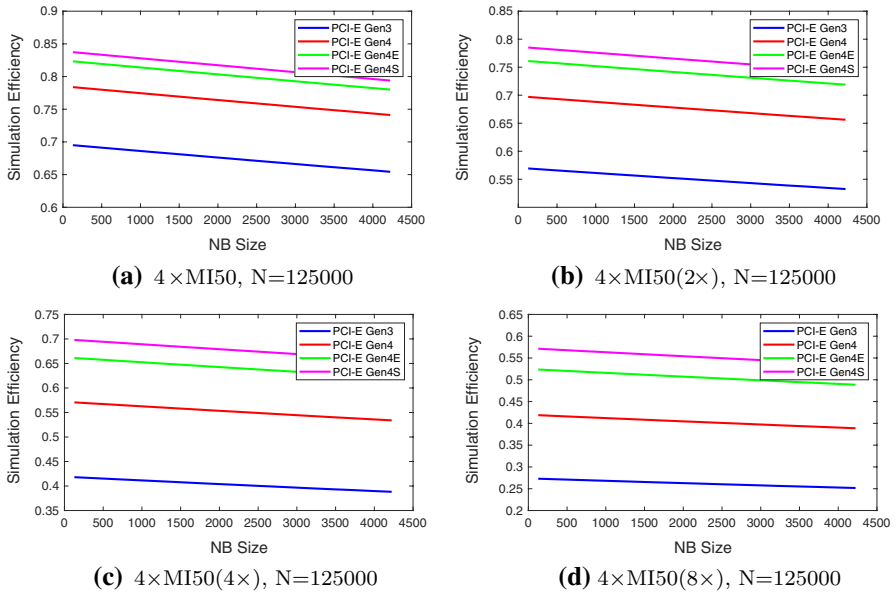
### 5.1 Case study for MI50 simulations

In this section, we carry out the HPL simulation according to the performance of the AMD Radeon Instinct MI50 GPU, so as to analyze the results of the model that we have established. The problem size of HPL (i.e., the size of $N$) is set to 125000. If the value of $N$ is increased, it will run out of the memory capacity of heterogeneous systems that we mention in Sect. 5.2. But if the N decreases, the capacity of GPU may not be fully explored. On the contrary, the performance of the system for DGEMM will be better and closer to the theoretical peak if the N increases. Therefore, within the allowable range of system memory capacity, we would better set $N$'s value to the maximum. Figure 5 shows the results of HPL single node simulation model at different PCI-E speeds. Figure 5a is the simulation result of four MI50 GPUs at different PCI-E speeds, while the 5b, c and d are twice, four times and eight times the performance of MI50. For instance, 4 × MI50 (2x) means four GPUs in the system and the theoretical performance of each GPU is twice the performance of MI50.

Assuming that the efficiency of Blas library and the parallel efficiency of GPU is 90%, the double-precision floating-point rate of MI50 is about 6600 Gflops. The transmission speed of PCI-E is as follows: Gen3 speed is 32 GB/s; Gen4 speed is 64 GB/s; Gen4E speed is 102.4 GB/s; Gen4S speed is 128 GB/s. In Fig. 5, the higher the PCI-E transmission rate is, the closer the actual performance of the system will be to the theoretical peak performance. The results also indicate a dropped efficiency



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com




**(a)** 4×MI50, N=125000 **(b)** 4×MI50(2×), N=125000

**(c)** 4×MI50(4×), N=125000 **(d)** 4×MI50(8×), N=125000

**Fig. 5** Simulation under different PCI-E speed

when *NB* grows larger. As a matter of fact, for the previous version of HPL, the matrices are stored in the host memory. The update of the matrix is sent and done inside the device memory after the CPU factorizes it. When blocking size (*NB*) is larger, the part of matrix needed to be updated becomes smaller, resulting in higher efficiency. However, the matrices are stored in device memory in the optimized version [14] of HPL. As it turned out, the update step is completed in device memory, while the factorization is sent to host memory and done by CPU. A larger block size will lead to more CPU workload, and the GPU cannot but wait while the CPU is doing factorization. The CPU has become the bottleneck of computing power under the circumstances. Therefore, the efficiency drops when the blocking size grows. Besides, the experiment demonstrates that the higher the theoretical performance of the system is, the lower the simulation result is when the PCI-E speed is the same. The reason is that the PCI-E has become the bottleneck of the system, and the system can not fully exploit the performance.

Figure 6 shows the PCI-E speed required to maintain 75% of the theoretical system performance for different *NB* sizes. N is set to 125000, the efficiency of GPU parallel and Blas library are both set to 90%. The PCI-E speed needs to be added with the increase in the *NB* size. This is also because the large block size will bring about high workloads.

To seek out how the network bandwidth of a cluster influences the system performance, we studied the case of the MI50 series using our multiple nodes simulation model. In our study, the cluster consists of 2 nodes and each node has 4 GPUs. The cluster uses a star topology InfiniBand network. We assumed that CPU parallel efficiency, GPU parallel efficiency, BLAS efficiency and network



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

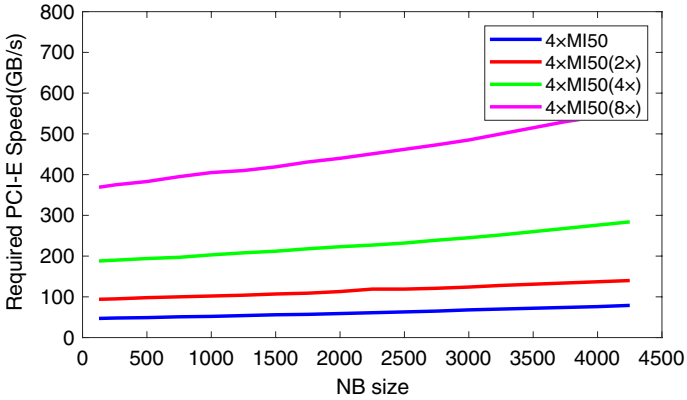
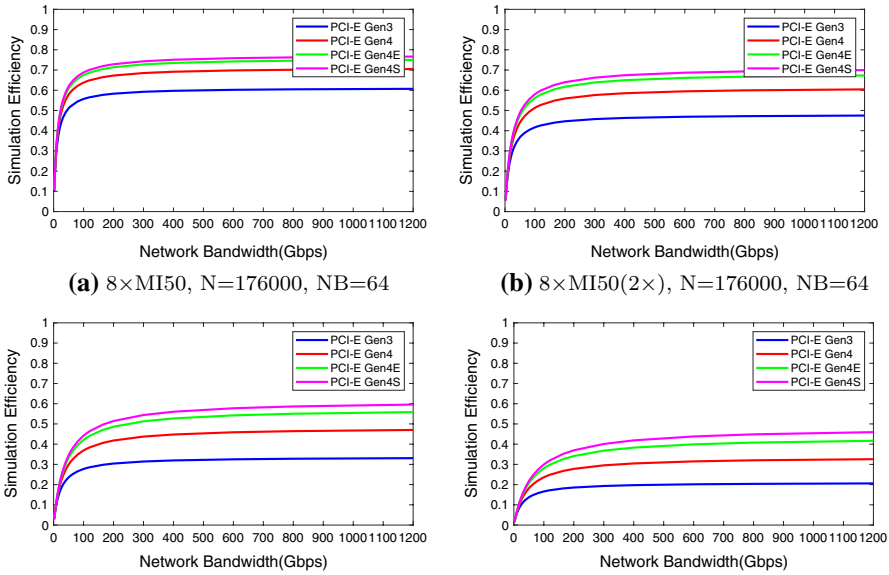



**Fig. 6** Simulation of required PCI-E speed to maintain 75% system efficiency




**(a)** 8×MI50, N=176000, NB=64 **(b)** 8×MI50(2×), N=176000, NB=64

**(c)** 8×MI50(4×), N=176000, NB=64 **(d)** 8×MI50(8×), N=176000, NB=64

**Fig. 7** Simulation the effect of network bandwidth

efficiency of the system can reach 80%, 80%, 85%, and 90%, respectively. We simulate several types of InfiniBand, so the network bandwidth will vary from 1Gbps to 1000Gbps. And the cluster does not have a leading node and all compute nodes are treated equally. The MI50 is used as a baseline GPU, then we study GPUs with twice, four times, and eight times performance of MI50. We include different PCI-E generations in our study to find out how the PCI-E bandwidth affects the efficiency.



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

**Table 1** The configuration of the heterogeneous system

| Platform | |
|---|---|
| CPU | Dual 16-core Intel Xeon Gold 6142 CPUs @2.60 GHz |
| Host memory | 256GB |
| GPU | Quad AMD Vega 20 GPUs, 32GB RAM |
| PCI-E | PCI-E Gen3 x16 |
| OS | Ubuntu 16.04 |
| ROCm | ROCm 3.0.6 |
| MPI | OpenMPI 4.0.1 |
| BLAS | OpenBLAS 0.3.9 |

**Table 2** Comparison of actual performance with estimate one

| Numbers of nodes | MI50 per node | N | NB | Est T. | Real per. (GFLOPS) | Est per. (GFLOPS) | Diff. |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 88000 | 256 | 55.21 | 8238 | 8656 | 5.07% |
| 1 | 2 | 88000 | 384 | 52.65 | 8232 | 8635 | 4.90% |
| 1 | 4 | 127000 | 256 | 85.42 | 15314 | 15945 | 4.12% |
| 1 | 4 | 127000 | 384 | 85.65 | 15230 | 15916 | 4.50% |
| 2 | 2 | 125000 | 128 | 56.10 | 14190 | 14659 | 3.31% |
| 2 | 2 | 125000 | 256 | 89.03 | 14079 | 14633 | 3.93% |
| 2 | 4 | 176000 | 64 | 139.26 | 24950 | 26099 | 4.61% |
| 2 | 4 | 176000 | 128 | 139.35 | 24832 | 26082 | 5.10% |

Figure 7 shows that the network bandwidth across different nodes does affect the system notably. Especially when the network bandwidth is less than 100 Gbps, the system performance will be badly restricted. This prove that the high-speed network are essential to the multiple nodes cluster. We also notice that higher PCI-E bandwidth can lead to higher performance, but the performance does not increase as the PCI-E bandwidth exceeds Gen4E (102.4 GB/s).

### 5.2 Model validation

In this part, we measure the performance of our HPL implementation on a heterogeneous system which equipped multi-cores and multi-GPUs to evaluate the credibility of the model. The specific configuration of a single node is listed in Table 1 [15]. In the case of multiple nodes, each node is equipped with the same software environment in Table 1. The cluster consists of 2 nodes, and all compute nodes are treated equally. The network we used in the multiple nodes is a star network based on 100Gbps InifiBand (4 Links EDR).

The results and comparison of the simulation model are shown in Table 2. We call the estimated time of HPL as Est T, the real performance as Real per, the



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

estimate performance as Est per and the difference as Diff. In order to measure the double-precision floating-point rate of the overall system for more accuracy, we lock the GPU core clock at 1301MHz and the memory clock at 801MHz. The estimated performance of the cluster is quite close to the real value, and the difference is about 5%. The above verification further ensures the reliability of the simulation model under normal circumstances. Benefit from the model, we can obtain the performance reference value of large-scale parallel system efficiently, conveniently, and accurately.

## 6 Summary

Large-scale studies on HPC applications are very time-consuming and resource-consuming. In this paper, we design an HPL simulation model since the mean tends to be an effective method for researchers to study the HPC system. According to the two different situations of the single node and multiple nodes, the results of HPL benchmark can be predicted on various high-performance heterogeneous systems accurately. We establish the simulation model using the parameters of the hardware, the network configuration and the HPL problem size. Then, we analyze the simulation results and verify the predictions through a series of experiments. The experiment shows that the block size *NB* will produce an effect on the performance of the heterogeneous system. For the optimized version of HPL, the increase in *NB* will reduce the performance of the system. On the other hand, PCI-E speed and the communication between multiple nodes are also an important contributor to the performance. With the improvement of PCI-E speed and the data rate of the communication, the impact on the performance will gradually reduce.

Although the simulation model has an enormous advantage, there are several limitations. Since it is the simulation of the HPL, the results do not represent the actual performance of an HPC system and are only for reference. Some small factors that may affect the system performance are not considered in the simulation model and it is hard to find out the bottleneck if it lies on these factors. Ultimately, the model only involves the general way of data transmission between CPUs and GPUs, other ways like NVLink are not considered. Because a reasonable utilization of simulation models will be of great help to the researchers to go deeply into the HPC system, we intend to conduct similar studies with other HPC benchmarks (e.g., HPCG or HPGMG [16]) as a future work.

**Acknowledgment** This work was supported by National Natural Science Foundation of China (CN) and Guangzhou Produce & Research Fund under grand no. 201902020004.

## References

1. Adalsteinsson H, Cranford S, Evensky DA, Kenny JP, Mayo J, Pinar A, Janssen CL (2010) A simulator for large-scale parallel computer architectures. Int J Distrib Syst Technol 1(2):57–73. https://doi.org/10.4018/jdst.2010040104



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

2. AMD (2017) Hpl-rocm. https://github.com/rocmarchive/HPL-ROCm
3. Ben-Nun T, Sutton M, Pai S, Pingali K (2017) Groute: An Asynchronous Multi-GPU Programming Model for Irregular Computations. In: Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Association for Computing Machinery, New York, NY, USA, PPoPP '17, pp 235–248, https://doi.org/10.1145/3018743.3018756,
4. Chen C, Fang J, Tang T, Yang C (2017) LU factorization on heterogeneous systems: an energy-efficient approach towards high performance. Computing 99(8):791–811. https://doi.org/10.1007/s00607-016-0537-2
5. Cornebize T, Heinrich FC, Legrand A, Vienne J (2017) Emulating High Performance Linpack on a Commodity Server at the Scale of a Supercomputer, https://hal.inria.fr/hal-01654804, working paper or preprint
6. Cornebize T, Legrand A, Heinrich FC (2019) Fast and Faithful Performance Prediction of MPI Applications: the HPL Case Study. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp 1–11, https://doi.org/10.1109/CLUSTER.2019.8891011
7. Davies T, Karlsson C, Liu H, Ding C, Chen Z (2011) High Performance Lipack Benchmark: A Fault Tolerant Implementation Without Checkpointing. In: Proceedings of the International Conference on Supercomputing, Association for Computing Machinery, New York, NY, USA, ICS '11, p 162–171, https://doi.org/10.1145/1995896.1995923
8. Degomme A, Legrand A, Markomanolis GS, Quinson M, Stillwell M, Suter F (2017) Simulating MPI applications: the SMPI approach. IEEE Trans Parall Distrib Syst 28(8):2387–2400. https://doi.org/10.1109/TPDS.2017.2669305
9. Dittmer S, Kluth T, Henriksen MTR, Maass P (2020) Deep image prior for 3d magnetic particle imaging: a quantitative comparison of regularization techniques on open mpi dataset. arXiv:2007.01593
10. Gan X, Hu Y, Liu J, Chi L, Xu H, Gong C, Li S, Yan Y (2018) Customizing the HPL for China accelerator. Sci China Inf Sci 61(4):42102. https://doi.org/10.1007/s11432-017-9221-0
11. Haitao Zhao Leisheng Li, Wenhao Yang, Hui Zhao, Huiyuan Li JS (2020) Research on HPL parallelcComputing model for a class of complex heterogeneous supercomputer system. http://www.jfdc.cnic.cn
12. Hemmatpour M, Montrucchio B, Rebaudengo M (2018) Communicating efficiently on cluster-based remote direct memory access (RDMA) over infiniband protocol. Appl Sci 8(11):2034
13. Hjelm N, Pritchard H, Gutiérrez SK, Holmes DJ, Castain R, Skjellum A (2019) MPI Sessions: Evaluation of an Implementation in Open MPI. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp 1–11, https://doi.org/10.1109/CLUSTER.2019.8891002
14. Huang J, Lu L (2019) Performance Optimization of High-Performance Linpack Based on GPU-Centric Model on Heterogeneous Systems. In: 2019 IEEE International Conference on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom), pp 1371–1377, https://doi.org/10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00197
15. Jo G, Nah J, Lee J, Kim J, Lee J (2015) Accelerating LINPACK with MPI-OpenCL oncClusters of multi-GPU nodes. IEEE Trans Parallel Distrib Syst 26(7):1814–1825
16. Kwack J, Bauer GH (2018) HPCG and HPGMG benchmark tests on multiple program, multiple data (MPMD) mode on Blue Waters–A Cray XE6/XK7 hybrid system. Concurr Comput: Pract Exp 30(1):e4298. https://doi.org/10.1002/cpe.4298
17. Lin F, Liu Y, Guo Y, Qian D (2020) ELS: Emulation system for debugging and tuning large-scale parallel programs on small clusters. J Supercomput. https://doi.org/10.1007/s11227-020-03319-6
18. Liu J, Xue Y, Ren K, Song J, Windmill C, Merritt P (2019) High-performance time-series quantitative retrieval from satellite images on a GPU cluster. IEEE J Sel Topics App Earth Observ Remote Sens 12(8):2810–2821. https://doi.org/10.1109/JSTARS.2019.2920077
19. Martin JP, Kandasamy A, Chandrasekaran K (2018) Exploring the support for high performance applications in the container runtime environment. Human-centric Comput Inf Sci 8(1):1. https://doi.org/10.1186/s13673-017-0124-3
20. McCalpin JD (2018) HPL and DGEMM Performance Variability on the Xeon Platinum 8160 Processor. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp 225–237, https://doi.org/10.1109/SC.2018.00021
21. Mohammadi M, Bazhirov T (2018) Comparative Benchmarking of Cloud Computing Vendors with High Performance Linpack. In: Proceedings of the 2nd International Conference on High




hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com

Performance Compilation, Computing and Communications, Association for Computing Machinery, New York, NY, USA, HP3C, pp 1–5, https://doi.org/10.1145/3195612.3195613
22. Mubarak M, Carothers CD, Ross RB, Carns P (2017) Enabling parallel simulation of large-scale HPC network systems. IEEE Trans Parallel Distrib Syst 28(1):87–100. https://doi.org/10.1109/TPDS.2016.2543725
23. Rohr D, De Cuveland J, Lindenstruth V (2016) A Model for Weak Scaling to Many GPUs at the Basis of the Linpack Benchmark. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp 192–202
24. Végh J (2018) Limitations of performance of exascale applications and supercomputers they are running on. arXiv:1808.05338
25. Yang C, Chen C, Tang T, Chen X, Fang J, Xue J (2016) An Energy-Efficient Implementation of LU Factorization on Heterogeneous Systems. In: 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), pp 971–979, https://doi.org/10.1109/ICPADS.2016.0130
26. Yang W, Li K, Li K (2017) A hybrid computing method of spmv on cpu-gpu heterogeneous computing systems. J Parallel Distrib Comput 104:49–60
27. Yong C, Lee GW, Huh EN (2018) Proposal of container-based HPC structures and performance analysis. J Inf Process Syst 14(6):1398–1404
28. Zhang Wenli and Fan Jianping CM (2004) Emulation and Forecast of HPL Test Performance. http://crad.ict.ac.cn
29. Zheng G, Kakulapati G, Kale LV (2004) BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In: 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings., p 78, https://doi.org/10.1109/IPDPS.2004.1303013

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



hyichang0510@gmail.com - May 6, 2021 - Read articles at www.DeepDyve.com