Dawning Nebulae: A PetaFLOPS Supercomputer with a Heterogeneous Structure

Article *in* Journal of Computer Science and Technology · May 2011

DOI: 10.1007/s11390-011-1138-3 · Source: DBLP

7 authors, including:

Jing Xing
Chinese Academy of Sciences
5 PUBLICATIONS **43** CITATIONS
SEE PROFILE

Zhigang Huo
Chinese Academy of Sciences
18 PUBLICATIONS **125** CITATIONS
SEE PROFILE

Jin Xiong
Chinese Academy of Sciences
39 PUBLICATIONS **271** CITATIONS
SEE PROFILE

Bo Li
Chinese Academy of Sciences
23 PUBLICATIONS **43** CITATIONS
SEE PROFILE

Sun NH, Xing J, Huo ZG *et al.* Dawning Nebulae: A petaFLOPS supercomputer with a heterogeneous structure. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 26(3): 352–362 May 2011. DOI 10.1007/s11390-011-1138-3

Dawning Nebulae: A PetaFLOPS Supercomputer with a Heterogeneous Structure

Ning-Hui Sun[1] (孙凝辉), *Member*, *CCF*, *IEEE*, Jing Xing[2,3] (邢　晶)
Zhi-Gang Huo[2] (霍志刚), *Member*, *CCF*, *ACM*, Guang-Ming Tan[1] (谭光明), *Member*, *CCF*, *ACM*
Jin Xiong[2] (熊　劲), *Member*, *ACM*, *IEEE*, Bo Li[2,3] (李　波), and Can Ma[2,3] (马　灿)

[1] *Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences, Beijing 100190, China*

[2] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*

[3] *Graduate University of Chinese Academy of Sciences, Beijing 100049, China*

E-mail: {snh, xingjing, zghuo, tgm, xj, leo, macan}@ncic.ac.cn

Received January 31, 2011; revised March 9, 2011.

Abstract Dawning Nebulae is a heterogeneous system composed of 9280 multi-core x86 CPUs and 4640 NVIDIA Fermi GPUs. With a Linpack performance of 1.271 petaFLOPS, it was ranked the second in the TOP500 List released in June 2010. In this paper, key issues in the system design of Dawning Nebulae are introduced. System tuning methodologies aiming at petaFLOPS Linpack result are presented, including algorithmic optimization and communication improvement. The design of its file I/O subsystem, including HVFS and the underlying DCFS3, is also described. Performance evaluations show that the Linpack efficiency of each node reaches 69.89%, and 1024-node aggregate read and write bandwidths exceed 100 GB/s and 70 GB/s respectively. The success of Dawning Nebulae has demonstrated the viability of CPU/GPU heterogeneous structure for future designs of supercomputers.

Keywords supercomputer, heterogeneous systems, performance evaluation

1 Overview of Dawning Nebulae

Dawning Nebulae supercomputer is built by Institute of Computing Technology of Chinese Academy of Sciences, in collaboration with Dawning Information Industry Corp. It will be deployed at the National Supercomputing Center in Shenzhen, Guangdong Province, to power the CNGrid Southern-China backbone node. It is the first supercomputer in China, and the third one in the world, which broke the one petaFLOPS Linpack performance barrier. In June 2010, Dawning Nebulae was ranked the second in the 35th TOP500 List.

To achieve petascale computing, we have addressed several critical design challenges including energy efficiency, packaging, programming model and I/O performance. Among them, energy efficiency was the most pressing one to deal with. The conventional approach in using general-purpose processors as main computing engine would break the power budget of 3 MW. The constraints of system peak performance and power consumption motivated us to choose a heterogenous architecture.

Originating from consumer graphics market, Graphics Processing Units (GPUs) have increased in performance and programmability in the last decade. One of the main barriers to its acceptance in high performance computing is the lack of memory that supports error correcting codes (ECC). The NVIDIA Fermi architecture is the first GPU with ECC capability, which makes it suitable for error-sensitive scientific computing. Compute Unified Device Architecture (CUDA)[1] is NVIDIA's C-like language for general-purpose computing on GPUs (GPGPU) and is currently the most popular environment for GPGPU.

The rest of this paper is organized as follows. Section 2 presents an overview and highlights of Dawning Nebulae supercomputer. Section 3 introduces the software stack in Dawning Nebulae. Section 4 gives an in-depth introduction to performance tuning towards

Regular Paper

This work is supported by the National Hi-Tech Research and Development 863 Program of China under Grant No. 2009AA01A129, the National Natural Science Foundation of China under Grant Nos. 60633040, 60803030, 61033009 the National Basic Research 973 Program of China under Grant No. 2011CB302500, the National Natural Science Foundation for Distinguished Young Scholars of China under Grant No. 60925009, and the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant No. 60921002.

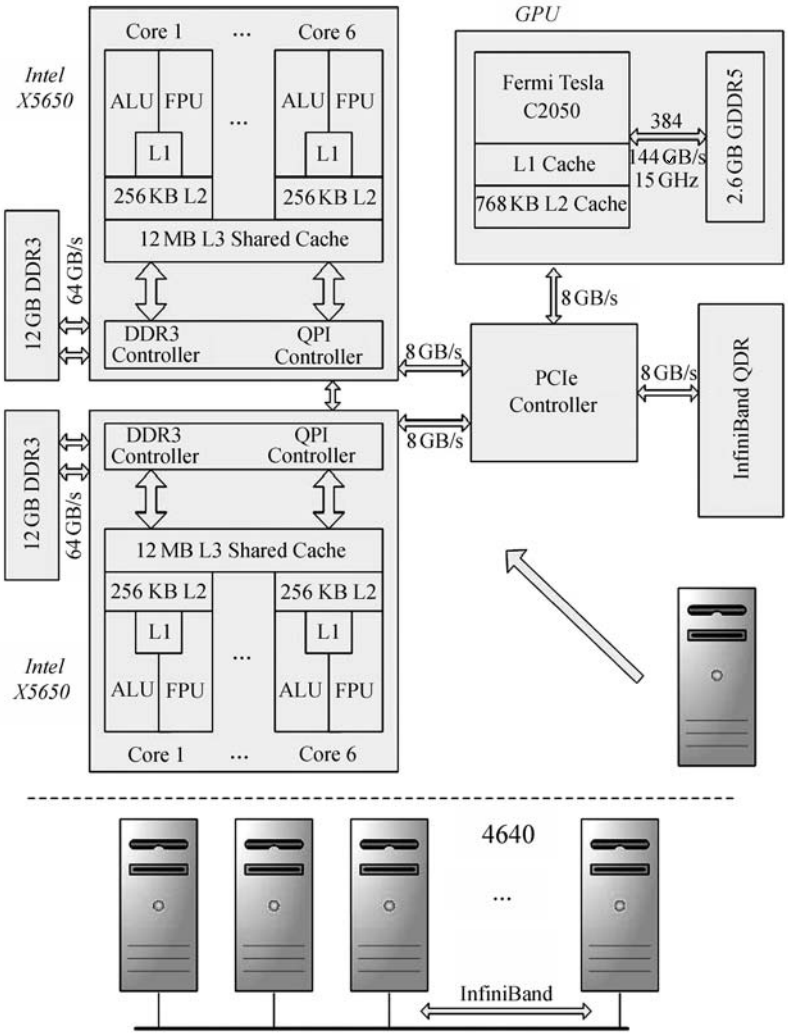


Fig.1. Dawning Nebulae system.

one petaFLOPS in Linpack test. Section 5 provides the design of Dawning Nebulae's file system, HVFS. Finally, Section 6 concludes this paper.

2 Highlights in the Design of the Dawning Nebulae

The architecture of Dawning Nebulae is shown in Fig.1. It is a heterogeneous system, consisting of both x86 multicore general-purpose processors and GPGPU accelerators. The main features of the Dawning Nebulae system is highlighted as follows.

2.1 System Highlights

• The theoretical peak performance of Dawning Nebulae is nearly 3 petaFLOPS. A sustained performance of 1.271 petaFLOPS was achieved in Linpack benchmark experiments. An efficiency of 42.6% was achieved.

• 55 680 cores in 9280 Intel 6-core Xeon processors working at 2.66 GHz are used, together with 4640 NVIDIA Tesla C2050 GPUs, one per CPU blade. There are 4640 two-way blades in 464 chassis housed in 116 cabinets in 7 rows. The system has a main memory capacity of 111.36 TB. Each rack delivers 25.7 teraFLOPS computing power. The high computing density per rack poses considerable challenges on rack cooling.

• All computing nodes are interconnected through a three-layer InfiniBand 4× QDR network. Each link's bi-directional bandwidth is up to 80 Gbps.

• Due to the extra heat generated by the GPUs, the requirement of thermal cooling capability of each rack may vary in a large range, from 25 kW to 35 kW. The system uses a combination of water cooling and intra-rack air cooling.

• Power consumption of the whole system is 2.58 MW. The power efficiency of 0.492 GFLOPS/Watt makes Dawning Nebulae the 14th in the Green500 List in November 2010.

2.2 System Packaging

Dawning Nebulae is built with 464 Dawning TC3600 blade systems (see Fig.2), which was announced at the Intel Developer Forum in September, 2009. It is the first blade system conforming to both the Server System Infrastructure (SSI) standard and pertinent standards from HPC Standardization Committee of China (HPCSC). The rear view of TC3600 in Fig.2 shows its expandability in multiple functional modules, including PCI Express expansions, InfiniBand switch, Ethernet switch, system management module, redundant fans and power supplies. Ten GPUs can be plugged into the PCI express expansion slots in the middle. Each fully-configured and loaded TC3600 blade system consumes about 6 kW of electricity. With 464 chassis, the total power consumption of Dawning Nebulae is less than 3000 kW.

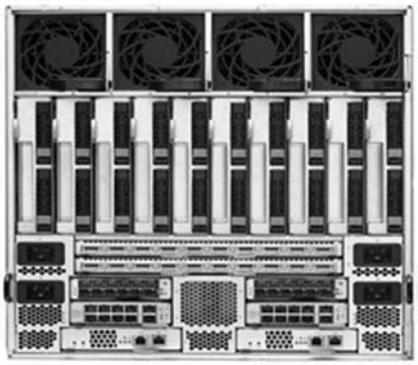


Fig.2. TC3600 blade chassis (rear view).

2.3 Interconnects

Dawning Nebulae is interconnected by a three-layered InfiniBand 4× QDR network. On the top layer of this network, there are five Mellanox 648-port InfiniBand switches. Each leaf switch is an expansion module in a TC3600 blade system. There are 18 outbound ports in a leaf switch, but only 5 of them are used for up links. The InfiniBand mezzanine HCA (Host Channel Adaptor) module is shown at the top left corner of Fig.3. The point-to-point latency at MPI level is 1.5 μs and the point-to-point bandwidth is 3.2 GB/s. It also supports GPU-Direct for communication optimization.

Fig.3. TC3600 blade.

3 Software Stack

The system software of Dawning Nebulae can be divided into three layers, namely the node operating system, the cluster management system and parallel computing middleware.

The node operating system is SUSE Linux Enterprise Server, which was selected in line with the expertise of Dawning engineers and configured to fit a supercomputing environment. The kernel was mildly but deliberately patched to take full advantage of the latest hardware features, such as TurboBoost and RAS features in the Westmere-EP processor.

The cluster management system is Dawning CloudView, which features graphical resources management and system health monitoring. The Dawning PowerConf power management system can help to reduce power consumption of the whole system by 15∼20%.

Parallel computing middleware plays a pivotal role in Nebulae, because the degree of parallelism can reach ten million in the whole system. A three-level hybrid programming model, combining MPI, OpenMP and CUDA, is used to exploit all computing power and parallelism in this hybrid system. Besides, an adaptive decomposition is used to balance the workload between CPUs and GPUs. OpenMPI is used for inter-node communication and OpenMP is utilized for intra-node communication between CPUs and GPUs. To achieve an extremely high floating-point efficiency in real applications, we adopt several optimization techniques to make full use of the architectural features of multi-core CPU and GPU. For example, how to increase active threads with limited shared resources (registers, shared memory); how to exploit the data reuse in the shared memory space. Through these optimizations, the Linpack efficiency of each node reaches 69.89%.

4 Performance Tuning and Optimization to Achieve PetaFLOPS Computing

In this section we describe the methodologies of performance optimization to achieve petascale computing on a Nebulae-like hybrid parallel architecture. From a system point of view, performance optimization should take full advantage of each component in the whole

system. Thus we consider Dawning Nebulae as a heterogeneous system with multiple levels of parallelism for performance optimization. The first level is of parallelism between CPUs and GPUs. Both CPUs and GPUs act as computing devices with different capabilities. The second level of parallelism is exposed by massive on-chip multi-core in both CPUs and GPUs.

In the following context, we take Linpack as an example to show how the multi-level parallelism is exploited to achieve petaFLOPS computing on Nebulae system. The Linpack benchmark is an implementation of the LU decomposition to solve a dense $N \times N$ system of linear equations: $Ax = b$. The solution is obtained by Gaussian elimination with partial pivoting, resulting in a floating point workload of $2/3N^3 + 2N^2$. Details of High Performance Linpack (known as HPL) implementation are referred to [2]. The right-looking variant of the LU factorization is show in Fig.4. After the panels are factorized, the trailing matrices (the yellow area in Fig.4) are updated. The updates perform matrix-matrix multiplications (DGEMM) which occupy most of the Linpack runtime. The bigger the problem size N is, the more time is spent in this routine, so it is critical to optimize DGEMM for achieving a high Linpack score.

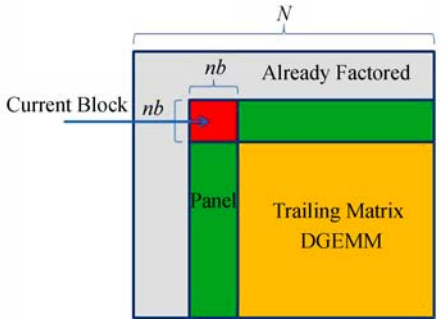


Fig.4. LU factorization.

4.1 Inter-Chip Parallelism

Since CPUs in Nebulae system provide 600 TFLOPS computing capability and DGEMM on the CPU approaches the peak performance, an efficient Linpack implementation should exploit the capability of both CPUs and GPUs in such a heterogeneous system. A feasible way is to partition workloads of matrix-matrix multiplications between CPUs and GPUs. Let us denote the input matrices as $\boldsymbol{A}(M, K)$, $\boldsymbol{B}(K, N)$ and $\boldsymbol{C}(M, N)$, a DGEMM call will compute $\boldsymbol{C} = \boldsymbol{AB} + \boldsymbol{C}$. Basically, the original matrices $\boldsymbol{C}$ and $\boldsymbol{A}$ are viewed as the union of two sub-matrices $\boldsymbol{C} = \boldsymbol{C}_1 \cup \boldsymbol{C}_2$ and $\boldsymbol{A} = \boldsymbol{A}_1 \cup \boldsymbol{A}_2$ with $\boldsymbol{M} = \boldsymbol{M}_1 + \boldsymbol{M}_2$ (see Fig.5). The DGEMM operation $\boldsymbol{C} = \boldsymbol{AB} + \boldsymbol{C}$ is expressed as $\boldsymbol{C} = (\boldsymbol{A}_1\boldsymbol{B} + \boldsymbol{A}_2\boldsymbol{B}) + (\boldsymbol{C}_1 + \boldsymbol{C}_2)$. Fatica[3] proposed a static algorithm to partition the workload of DGEMM. Let us denote G_{GPU} and G_{CPU} to be the workload of DGEMM on GPU and CPU, respectively. The optimal split fraction is $split = G_{\mathrm{GPU}}/(G_{\mathrm{GPU}} + G_{\mathrm{CPU}})$. However, one problem is that the sizes of matrices change when the factorization moves to the right. We observe that there are two types of matrices during trailing matrices update. One type is characterized by a small matrix $\boldsymbol{B}$ with $K = N$, the other has two big matrices $\boldsymbol{A}, \boldsymbol{B}$, where $K \neq N$. A heuristic algorithm is used to update the optimal factor at the end of each DGEMM. The algorithm proceeds as shown in Fig.6.

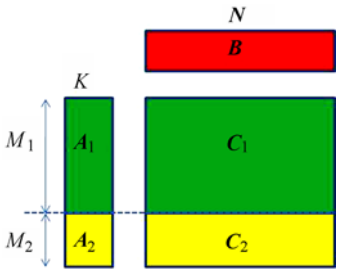


Fig.5. DGEMM is partitioned between GPU (green part) and CPU (yellow part).

```
split[1 ... 128] = guess an initial value;
if (n == k)
    last = split[k/64]
else
    last = split[0]
for each DGEMM call do {
    gpu_flops = G_GPU / GPU_TIME;
    cpu_flops = G_CPU / CPU_TIME;
    opt = gpu_flops/(gpu_flops + cpu_flops)
    if (n == k) {
        split[k/64] = opt;
    } else {
        if (abs(opt − last) > 0.01)
            split[0] = opt;
        else
            split[0] = (opt + last)/2;
    }
}
```

Fig.6. Adaptive load balancing algorithm.

Note that the adaptive load balancing algorithm measures the execution time. The time on GPU includes the extra overhead for transferring data between GPU and CPU (more discussion in Subsection 4.3).

4.2 Intra-Chip Parallelism

For multi-core CPUs we adopt the vendor's optimized math library to utilize multi-core on-chip parallelism since the library has already achieved more

than 90% efficiency of theoretical peak performance. With a new many-core architecture, CUDA GPUs still have room for performance improvements. The challenge of approaching peak performance on a CUDA GPU is how to maximize the number of concurrent threads under the constraint of shared resources (register files, shared memory and off-chip memory). We leverage our previous work on latency tolerant model[4] to address this problem. For completeness, we review the latency tolerant model, which is highlighted as follows.

• *Two-Level Memory Model.* The shared memory space is partitioned into low latency in-core memory (ICM) and high latency out-of-core memory (OCM).

• *Stronger Execution Model.* A computation processing unit (PE) is active only if its required data are in ICM; any data will be copied into ICM only when they are needed by a computation PE.

• *Fine-Grain Parallelism.* Accesses to ICM and OCM proceed in parallel.

Under the latency tolerant model, the execution of a program is a streaming pipeline of LOADLT-EXECLT-STORELT[4].

The design philosophy of CUDA architecture is to accelerate application performance through massive multi-threading. More threads can keep more CUDA cores busy and be scheduled to hide overheads of long latency operations. CUDA cores are organized into multiple groups, each of which is called streaming multiprocessors (SM). Accordingly, all threads are organized into multiple thread blocks, each of which is scheduled to one SM.

With respect to the latency model, shared memory and global memory are mapped to ICM and OCM, respectively. Initially, we assume that three input matrices locate in OCM in a row-wise order. Matrix $\boldsymbol{C}$ is partitioned into blocks of size $NB \times NB$, NB means the blocking factor to partition and distribute the matrix, each of which is mapped to one thread block. Assume that matrix block $\boldsymbol{C}(i,j)$ is mapped to thread block $T(i,j)$. The calculation of $T(i,j)$ needs both stripped $\boldsymbol{A}(i,k)$ and $\boldsymbol{B}(k,j)$, where $k = 1, 2, \ldots, N$. According to the principle of the latency tolerant model, all compute threads only access ICM. Therefore, the stripped matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are further divided into smaller blocks along k dimension, which are loaded into ICM through multiple pipelining stages. According to the third rule of the latency tolerant model, we allocate double buffers in shared memory to implement parallelism between ICM and OCM. The latency tolerant model based algorithm is described in Fig.7.

Each thread executes the same program instructions with different matrix elements as described in Fig.7. Logically the latency tolerant model partitions threads into groups of percolation threads (who perform load/store operations)[4] and compute threads. On CUDA architecture, the thread partition relies on the zero-overhead thread switch. When one warp of threads performs *load*, other warp of threads performs *mult* benefit from the use of double-buffering in shared memory.

In order to further hide the latency of shared memory access and reduce the number of used registers, each thread computes a 4×4 sub-block and performs interleave memory load with double floating-point operations (see Fig.7). The constraints to the number of active threads include register file, shared memory and maximum parallelism on each SM. For the example of NVIDIA Fermi architecture, there are 16 SMs, each of which shares 32 768 registers, 48 KB shared memory (configurable) and supports the maximum number of 8 concurrent thread blocks. In our final program we use 60 registers for each thread and 32 KB shared memory for each thread block. Therefore there are two concurrent thread blocks of 256 threads running on one block.

```
tag = 0;
//represents double-buffering in shared memory
load A(i,0), B(0,j) to buffer [tag];
//KK is the number of blocks
  for kk = 1,...,KK do {
    load A(i,kk), B(kk,j) to buffer[tag];
    //register reuse for saving load
    load a[0],a[1],b[0],b[1] to registers;
    mult(a[0],a[1],b[0],b[1]);
    load b[2],b[3] to registers;
    mult(a[0],a[1],b[2],b[3]);
    load a[2],a[3] to registers;
    mult(a[2],a[3],b[0],b[1]);
    mult(a[2],a[3],b[2],b[3]);
}
```

Fig.7. Latency tolerant algorithm for matrix multiplication.

4.3 Communication Optimization

Linpack with GPU acceleration displays communication patterns quite different from those observed in conventional Linpack tests. One of the most significant differences is that the NB of Linpack with GPU acceleration should be adjusted to a larger value to gain optimized performance. Single node comparison tests show that, 232 is the best choice of NB under CPU-only conditions, while 768 is the best one after a GPU is added. To understand how NB affects communication patterns, we profiled the MPI call times and packet sizes in the Linpack test. For NB equal to 232, all packets are less than 128 MB, while packets between 128 MB

and 512 MB are quite frequent when *NB* equals 768.

There are interactions between multiple GPUs. Prior to the introduction of GPU-Direct technology, any communication between GPUs had to involve the host CPU and required buffer copies. Each GPU-to-GPU communication had to follow the steps as shown in Fig.8(a).

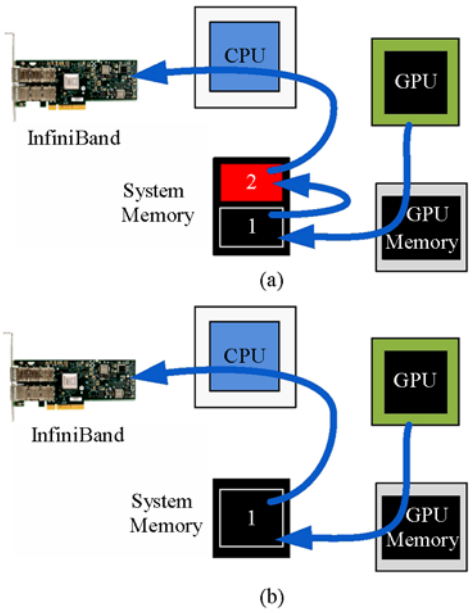


Fig.8. GPU-to-GPU communication without/with GPU-Direct. (a) Without GPU-Direct. (b) With GPU-Direct.

When using GPU acceleration, the application calls *cudaMallocHost* first to allocate a host memory buffer for the GPU. This buffer is used for communication between GPU and CPU. The GPU driver pins the memory buffer down to make sure that the GPU has full control of the allocated memory. In the traditional mode, the memory pinned down by a GPU could not be shared with, or pinned down again by other devices, such as InfiniBand HCA (Host Channel Adapter). This limitation results in that InfiniBand HCA could not use RDMA technology in GPU dedicated host memory, since itself also has to pin down the memory before an RDMA operation can be initiated.

In the GPU-Direct implementation, the GPU driver would add a flag named *VM_DRIVER_PAGES* to the Virtual Memory Area (VMA) structure when *cudaMallocHost* is called. Meanwhile, a *notifier_list* is also added to the VMA structure. When another I/O device needs to share the GPU dedicated host memory, its device driver would add a callback function to the *notifier_list*, which will be called when the memory is being freed by the GPU to avoid collisions between devices sharing the memory. When the InfiniBand HCA driver registers memory, it would first test if the flag *VM_DRIVER_PAGES* in the corresponding VMA structure is set. If so, InfiniBand HCA driver would add a dedicated callback function to the VMA. When VMA is to be freed, the callback function would be called to deregister the memory. In such a way, the Fermi driver could share the GPU dedicated host memory with the InfiniBand driver, thus InfiniBand could initiate RDMA operations for the GPU pinned memory. Fig.8(b) shows the GPU-to-GPU communication mode with GPU-Direct support.

4.4 Linpack Performance on Nebulae

We evaluate the optimization strategies to Linpack benchmark on Nebulae system. First, the improvement to both Linpack and matrix multiplication kernel (DGEMM) on single node is shown. Second, we report the performance and scalability of Linpack for scaling the number of computing nodes on Nebulae.

Fig.9 reports performance comparison for DGEMM between our optimized program and CUBLAS3.0 provided by NVIDIA. It shows our optimization strategies improve performance by about 2 times. The optimized DGEMM can achieve 70% efficiency of the peak performance. Figs. 10 and 11 present Linpack performance on

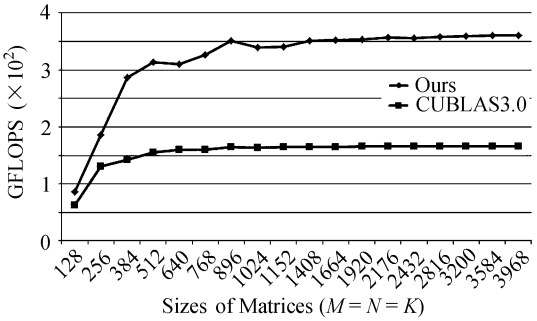


Fig.9. Performance comparison for DGEMM between ours and CUBLAS3.0.

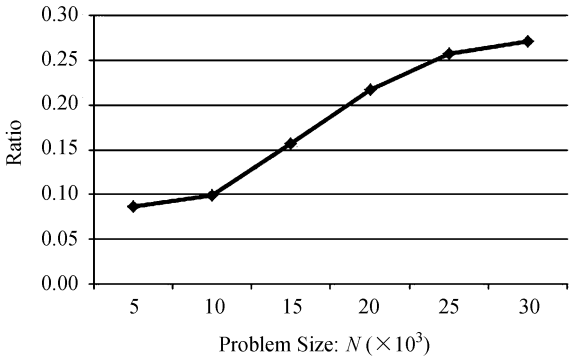


Fig.10. Performance improvement of the adaptive load balancing algorithm.

a single computing node. The adaptive load balancing algorithm for partitioning workload between CPU and GPU improve Linpack performance by more than 20% (see Fig.10). When the problem size is large enough to consume 80% of memory space, Linpack achieves 68.17% efficiency of peak performance. Finally, the scalability of Linpack on Dawning Nebulae is summarized in Fig.12. The first point to break petaFLOPS barrier is 4096 nodes, where Linpack achieves 1.08 petaFLOPS. For the whole system with 4640 nodes, Linpack reaches 1.271 petaFLOPS.

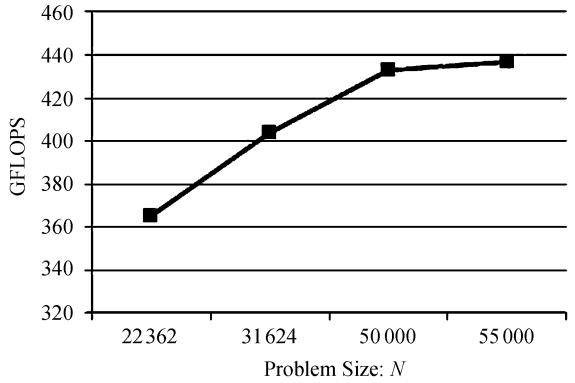


Fig.11. Linpack performance on a single node for the scaling problem sizes.

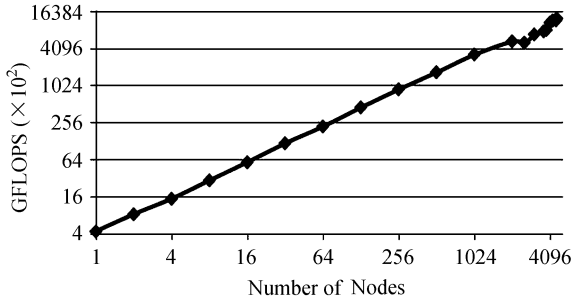


Fig.12. Linpack scores on Dawning Nebulae.

5 File I/O Subsystem

5.1 Design Goals of HVFS

The file I/O subsystem becomes more and more challenging for petascale supercomputers each of which consists of tens of thousands of cores and tens of peta-bytes of storage. The Nebulae supercomputer is designed for many applications with very different file access requirements. For example, some high-end scientific applications may issue many concurrent file accesses to many large files and thus require high aggregate I/O bandwidth, while enterprise applications may just require a low-cost and easy-to-use platform to share the data within a company. Considering the diversity of I/O requirements from different types of applications on Nebulae, we designed and implemented a hyper virtual file system, called HVFS. HVFS is a virtual file system built over multiple physical file systems of different types and integrates them seamlessly into a very large shared file system with a single name space and transparent data placement. HVFS can meet the diversity of I/O requirements by integrating different low-level physical file systems according to the requirements on I/O performance, functionalities, volume, cost, and power. For example, high-performance cluster file systems, such as DCFS3, PanFS[5] and HDFS[6], can be used for the high aggregate I/O bandwidth requirement; parallel file systems, such as GPFS[7], Lustre[8] and PVFS[9], for high parallel I/O performance requirement; shared SAN file systems, such as SANergy[10] and StorNext[11], for data sharing requirement; and high-volume archive file systems, such as DataDomain[12], for archiving of large volumes of data.

The design goals of HVFS are included as follows:

• It should provide a single and global name space to the end users without even knowing the low-level file systems. This goal is out of the consideration of easy-to-use for users and compatible with users' past experience with global file systems on supercomputers.

• It should provide high aggregate I/O bandwidth, especially 200 GB/s for the Nebulae. This requires that the integration layer introduce very little additional overhead for I/O bandwidth and HVFS can achieve almost the I/O bandwidth of low-level physical file system.

• It should place user files on a suitable low-level physical file system in both automatic and manual manner, i.e., without or with user's guidance. The former is for the majority of the users, while the latter is for a few advanced users.

5.2 Key Design Issues

5.2.1 Adaptive and Scalable Distributed Metadata Management

HVFS is a file system that manages tens of petabytes storage and this storage is partitioned among a couple of low-level physical file systems, each of which may be a distributed file system over peta-bytes of storage and hundreds of millions of files. All the files of the low-level file system are also managed by HVFS. Therefore, the number of files in HVFS can be very large, and HVFS aims at efficiently managing billions to trillions of files. Moreover, HVFS also aims at efficiently managing big directories, each of which may contain billions of files. To meet this end, a distributed metadata management should be exploited to make full use of processing capacity, memory caching capacity

and disk I/O capacity of a number of metadata servers. Metadata management of existing cluster file systems as well as parallel file systems has severe limitations when managing billions of files[13-14]. First, the metadata organization and location mechanisms generally aim at managing directories with millions of files and expect hundreds of millions of files in total in a file system, and hence they are not scalable and perform very poor for directories with billions of files. Second, the synchronization mechanism of existing cluster file systems greatly restrict the parallelism of metadata modifications, leading to poor file creation rate if a large number of processes create files simultaneously. Third, caching and load balancing are not considered by existing metadata management. Highly concurrent accesses from a large number of processes result in a vast amount of random accesses from the perspective of each metadata server which causes frequent replacement in the metadata cache, and uneven access loads among metadata servers. Both of them greatly reduce metadata performance under highly concurrent access, and slow metadata processing also impairs the aggregate I/O bandwidth.

To address the above issues, and motivated by Giga+[15], we designed an adaptive and scalable metadata management policy[16] for HVFS. First, in HVFS, an adaptive two-level directory partitioning based on extendible hashing[17] is used to manage very large directories. In the first level each directory is divided into multiple partitions to distribute on multiple servers. The first level partitioning controls the distribution of partitions among metadata servers. And in the second level each partition is divided into a certain number of metadata chunks. The second level partitioning controls the size of each partition. So any file can be located within two I/O accesses. This metadata organization is very scalable and can maintain a billion-scale file system efficiently. Second, HVFS uses smaller data structures, such as partition and chunk, as the metadata control unit for metadata modifications. Therefore, update operations such as file creations or deletions in the same directory can be processed concurrently. Third, since different kinds of metadata have different access patterns, HVFS exploits differences in importance of all kinds of metadata to partition the metadata cache into multiple layers with different replacement priorities. In this way, the most frequently accessed metadata, such as the directory information and the partition information which are shared by many directory entries, can be cached in the memory for the longest time. Finally, a dynamic load balance approach is used to rebalance the workload among metadata servers, when new metadata servers are added to the system, or some metadata servers are unavailable.

5.2.2 High Bandwidth Cluster File System DCFS3

DCFS3 is a cluster file system designed for large supercomputers consisting of thousands of nodes and peta-bytes of storage. The file I/O of DCFS3 is optimized to make full use of network transmission and disk I/O capacity of storage servers to achieve high aggregate I/O bandwidth for a large number of concurrent accesses from many concurrent processes. However, all file systems for large-scale supercomputers face a common challenge. Storage server can hardly make good use of its memory caching and disk I/O capacity, because they are designed for local applications running on the server nodes themselves, whereas applications on supercomputers run on compute nodes, instead of on storage server nodes. With the indirection of compute nodes, access patterns of concurrent processes on a large number of compute nodes are mixed as they arrive at a storage server. The file caching scheme, the prefetching scheme and the disk I/O scheduling scheme on the storage servers are optimized for local applications. The mixed accesses from a large number of compute nodes turn to be the accesses from the agent processes on the storage servers, which make the optimizations not work. Some researchers have proposed several solutions for general cases. For example, Zhou[18], Chen[19], Li[20], Jiang[21] and Yadgar[22] proposed cache replacement policies for multi-level buffer caches; Li[23-24], Liang[25], Zhang[26], and Li[27] proposed prefetching schemes for storage servers. In addition, Nisar[28], Chen[29] and Byna[30] proposed caching and prefetching schemes implemented in the MPI-I/O layer for parallel applications.

To address the I/O problems on the storage servers, and motivated by Chen's work[31], we implement a hint-based I/O scheme in DCFS3. DCFS3's client extracts the user process information of each I/O request as hints and sends the hints along with the I/O request to the storage server. Storage servers use the hints and can link I/O requests to their corresponding applications. 1) For prefetching, the storage servers of DCFS3 cache the prefetching state information (no data) of each user process in memory and fill the readahead data structure in the Linux kernel according to the cached prefetching state information before performing read requests. So the readahead scheme in the Linux kernel can work because DCFS3's server implementation makes it feel like that the user processes run locally. 2) For block allocation, the storage servers of DCFS3 also cache the block reservation information of each object in memory and transfer it to Linux kernel's EXT3 module so that

applications that sequentially write large volumes of data to an object can acquire contiguous disk blocks and thus improve I/O performance of these applications. 3) DCFS3 implemented an object-level scheduler above the block-level scheduler on the storage servers. In the object-level scheduler, there is a request queue for each object. Each time the object-level scheduler chooses a request queue and submits a number of requests on this queue to the block-level scheduler. Since the blocks of each object are contiguous on disk in most cases by the second method mentioned above, the object-level scheduler helps multiple requests that access contiguous blocks of the same object to be merged into a single larger request. Without the object-level scheduler, these requests may be interleaved with the requests that access other objects. Therefore, the object-level scheduler can improve I/O performance by issuing larger requests to disk and greatly reduce the overhead of disk head movement.

5.3 Performance of HVFS

We evaluated the performance of HVFS using the *iozone* benchmark[32]. Figs. 13(a) and 13(b) show the aggregate read and write bandwidth of HVFS by using 1024 nodes, with each configured as both a DCFS3 storage server and client. Each storage server uses a 300 GB local disk to store DCFS3 data. The average read and write bandwidths of each disk are about 110 MB/s and 70 MB/s respectively.

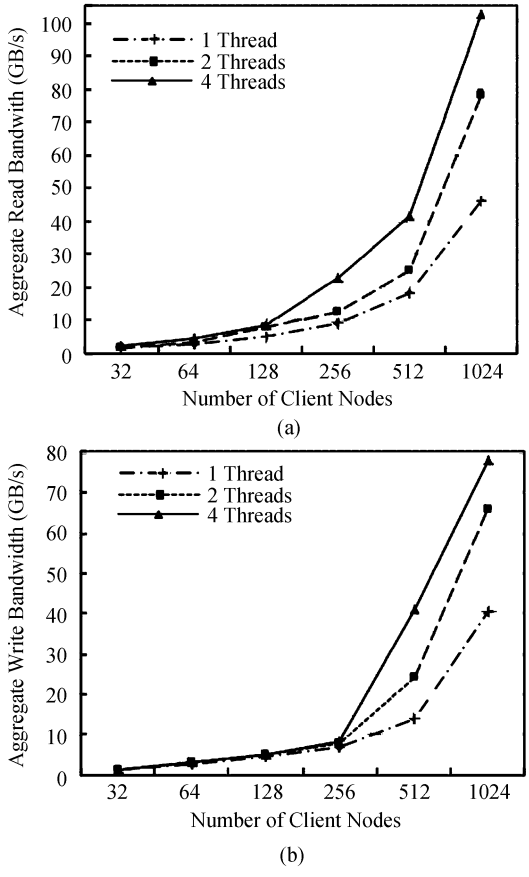


Fig.13. Performance of HVFS. (a) Average read bandwidth. (b) Average write bandwidth.

As shown in Fig.13, the aggregate read and write bandwidth of HVFS increase along with the number of storage servers and the number of I/O threads on each client node. When using 1024 storage servers and 1024 client nodes, with 4 I/O threads on each client node, the aggregate read and write bandwidths exceed 100 GB/s and 70 GB/s respectively.

6 Conclusions

We have addressed various challenges from petascale computing during the development of Nebulae system. It delivers excellent performance with high reliability, under the conditions of limited budget, power constraints, and a tight timeline. The provided computing power and storage capacity have enabled scientists and engineers to solve complex problems in order to advance research in the fields of material, remote sensing, medicine, economic modeling, geoscience and atmospheric science.

The completion of Nebulae infrastructure and its subsequent technology transfers to small scale systems have demonstrated the viability of CPU/GPU structure for the design of supercomputers. As the total cost of ownership of a supercomputer continues to increase and the CPU/GPU hybrid architecture becomes mature, the Dawning Nebulae sets a promising example for the future designs of supercomputers.

References

[1] Compute unified device architecture. http://www.nvidia.com/object/cuda_home_new.html, 2011.

[2] Petitet A, Whaley R C, Dongarra J, Cleary A. HPL — A portable implementation of the high performance Linpack benchmark for distributed memory computers, version 2.0. http://www.netlib.org/benchmark/hpl/, Sept. 2008.

[3] Fatica M. Accelerating Linpack with CUDA on heterogenous clusters. In *Proc. the 2nd Workshop on General Purpose Processing on Graphics Processing Units* (*GPGPU-2*), Washington DC, USA, Mar. 8, 2009, pp.46-51.

[4] Tan G, Sun N, Gao G R. Improving performance of dynamic programming via parallelism and locality on multi-core architectures. *IEEE Transactions on Parallel and Distributed Systems*, 2009, 20(2): 261-274.

[5] Nagle D, Serenyi D, Matthews A. The Panasas ActiveScale storage cluster — Delivering scalable high bandwidth storage. In *Proc. 2004 IEEE/ACM High Performance Computing, Networking and Storage Conference* (*SC 2004*), Pittsburgh, USA, Nov. 6-12, 2004, p.53.

[6] Shvachko K, Huang H, Radia S, Chansler R. The Hadoop distributed file system. In *Proc. the 26th IEEE* (*MSST2010*) *Symposium on Massive Storage Systems and Technologies*

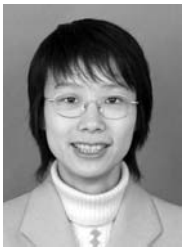
(*Research Track*), Inchine Village, USA, May 3-7, 2010.
[7] Schmuck F, Haskin R. GPFS: A shared-disk file system for large computing clusters. In *Proc. the First USENIX Conference on File and Storage Technologies* (*FAST 2002*), Monterey, USA, Jan. 28-30, 2002, Article No.19.
[8] Braam P J. The Lustre storage architecture. White Paper, Cluster File Systems, Inc., Oct. 2003.
[9] http://www.pvfs.org/, 2011.
[10] IBM Tivoli SANergy administrator's guide, Version 3, Release 2. IBM Corporation, Oct. 2002.
[11] http://www.quantum.com/Products/Software/StorNext/Index.aspx.
[12] http://www.datadomain.com/, 2011.
[13] Ghemawat S, Gobioff H, Leung S T. The Google file system. In *Proc. the 19th ACM Symp. Operating Systems Principles* (*SOSP 2003*), New York, USA, Oct. 19-22, 2003, pp.29-43.
[14] http://ceph.newdream.net/, 2011.
[15] Patil S, Gibson G. GIGA+: Scalable directories for shared file systems. Carnegie Mellon University Parallel Data Lab, Technical Report CMU-PDL-08-110, Oct. 2008.
[16] Xing J, Xiong J, Sun N, Ma J. Adaptive and scalable metadata management to support a trillion files. In *Proc. the SC 2009*, Portland, USA, Nov. 14-20, 2009, Article No. 26.
[17] Fagin R, Nievergelt J, Pippenger N, Strong H R. Extendible hashing — A fast access method for dynamic files. *ACM Trans. Database Systems*, Sept. 1979, 4(3): 315-344.
[18] Zhou Y, Chen Z, Li K. Second-level buffer cache management. *IEEE Transactions on Parallel and Distributed Systems*, Jun. 2004, 15(6): 505-519.
[19] Chen Z, Zhang Y, Zhou Y, Scott H, Schiefer B. Empirical evaluation of multi-level buffer cache collaboration for storage systems. In *Proc. Int. Conf. Measurements and Modeling of Computer Systems* (*SIGMETRICS 2005*), Banff, Canada, Jun. 6-10, 2005, pp.145-156.
[20] Li X, Aboulnaga A, Salem K, Sachedina A, Gao S. Second-tier cache management using write hints. In *Proc. the 4th USENIX Conference on File and Storage Technologies* (*FAST 2005*), San Francisco, USA, Dec. 13-16, 2005, pp.115-127.
[21] Jiang S, Zhang X. ULC: A file block placement and replacement protocol to efficiently exploit hierarchical locality in multi-level buffer caches. In *Proc. the 24th International Conference on Distributed Computing Systems* (*ICDCS 2004*), Tokyo, Japan, Mar. 24-26, 2004, pp.168-177.
[22] Yadgar G, Factor M, Li K, Schuster A. MC2: Multiple clients on a multilevel cache. In *Proc. the 28th International Conference on Distributed Computing Systems* (*ICDCS 2008*), Beijing, China, Jun. 17-20, 2008, pp.722-730.
[23] Li C, Shen K. Managing prefetch memory for data-intensive online servers. In *Proc. the 4th USENIX Conference on File and Storage Technologies* (*FAST 2005*), San Francisco, USA, Dec. 13-16, 2005, pp.253-266.
[24] Li C, Shen K, Papathanasiou A. Competitive prefetching for concurrent sequential I/O. In *Proc. EuroSys 2007 Conference*, Lisbon, Portugal, Mar. 21-23, 2007, pp.189-202.
[25] Liang S, Jiang S, Zhang X. STEP: Sequentiality and thrashing detection based prefetching to improve performance of networked storage servers. In *Proc. the 27th International Conference on Distributed Computing Systems* (*ICDCS 2007*), Toronto, Canada, Jun. 25-29, 2007, Article No. 64.
[26] Zhang Z, Lee K, Ma X, Zhou Y. PFC: Transparent optimization of existing prefetching strategies for multi-level storage systems. In *Proc. the 28th International Conference on Distributed Computing Systems* (*ICDCS 2008*), Beijing, China, Jun. 17-20, 2008, pp.740-751.
[27] Li M, Varki E, Bhatia S, Merchant A. TaP: Table-based prefetching for storage caches. In *Proc. the 6th USENIX Conference on File and Storage Technologies* (*FAST 2008*), San Jose, USA, Feb. 26-29, 2008, Article No. 6.
[28] Nisar, W Liao, A Choudhary. Scaling parallel I/O performance through I/O delegate and caching system. In *Proc. the 2008 International Conference on for High Performance Computing, Networking, Storage and Analysis* (*SC 2008*), Austin, USA, Nov. 15-21, 2008, Article No. 9.
[29] Chen Y, Byna S, Sun X, Thakur R, Gropp W. Hiding I/O latency with pre-execution prefetching for parallel applications. In *Proc. the 2008 International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC 2008*), Austin, USA, Nov. 15-21, 2008, No. 40.
[30] Byna S, Chen Y, Sun X, Thakur R, Gropp W. Parallel I/O prefetching using MPI file caching and I/O signatures. In *Proc. the 2008 International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC 2008*), Austin, USA, Nov. 15-21, 2008, Article No. 44.
[31] Chen H, Xiong J, Sun N. A novel hint-based I/O mechanism for centralized file server of cluster. In *Proc. 2008 IEEE International Conference on Cluster Computing* (*Cluster 2008*), Tsukuba, Japan, Sept. 29-Oct. 1, 2008, pp.194-201.
[32] Norcott W D. Iozone file system benchmark. 2005, http://www.iozone.org/docs/IOzone_msword_98.pdf.

Ning-Hui Sun received his B.S. degree from Peking University in 1989 and M.S. & Ph.D. degrees from the Chinese Academy of Sciences in 1992 and 1999, respectively. He is a professor in the Institute of Computing Technology, Chinese Academy of Sciences. He is the architect and main designer of the Dawning series high performance computers, from Dawning2000 to Dawning Nebulae. His research interests include computer architecture, operating system, and parallel algorithm. He is a member of CCF and IEEE.

Jing Xing received his B.S. degree in computer science and technology from Hunan University of Science and Technology in 2004. He is pursuing his Ph.D. degree of computer architecture at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include cluster file system and distributed system.

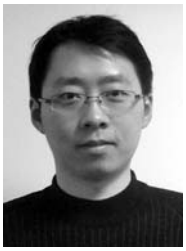
Zhi-Gang Huo received his B.S. degree from Northeastern University in 2000 and his Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences in 2007, both in computer science. He is an associate professor in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include computer architecture, operating system and fault tolerance. He is a member of CCF and ACM.

Guang-Ming Tan received the Ph.D. degree in computer science from the Chinese Academy of Sciences. He is an associate professor in the Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences. From 2006 to 2007, he was a visiting researcher in the Computer Architecture and Parallel Systems Laboratory (CAPSL), University of Delaware. His research interests include parallel algorithm and programming, performance modeling and evaluation, and computer architecture. He has published several papers in important conferences and journals, such as SC, ICS, SPAA, IEEE TPDS. He is a member of CCF and ACM.



Jin Xiong received the B.S. degree from Sichuan University, China, in 1990, the M.S. degree from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) in 1993, and the Ph.D. degree from the Graduate University of Chinese Academy of Sciences in 2006. She is an associate professor at ICT, CAS. Her research interests include cluster file systems, high performance I/O systems and data management. She is a member of IEEE and ACM.

Bo Li is a Ph.D. candidate in Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include high performance communication and networks, and parallel computing middleware.



Can Ma received his B.S. degree from Nankai University in 2006. He is a Ph.D. candidate in the Institute of Computing Technology, Chinese Academy of Science. His research interests include distributed file system and dependable computing.