Front. Comput. Sci. China 2010, 4(4): 445–455
DOI 10.1007/s11704-010-0383-x

RESEARCH ARTICLE

TH-1: China's first petaflop supercomputer

Xuejun YANG (✉), Xiangke LIAO, Weixia XU, Junqiang SONG, Qingfeng HU, Jinshu SU, Liquan XIAO, Kai LU, Qiang DOU, Juping JIANG, Canqun YANG

School of Computer Science, National University of Defense Technology, Changsha 410073, China

Abstract In recent years, heterogeneous systems and cooperative computing have become popular research directions in the field of high performance computing. With fast scaling of the size of high performance computer systems, problems such as power consumption and reliability come to the forefront. The aim of high performance computing has thus shifted from merely seeking peak performance to comprehensively pursuing high efficiency, which takes into consideration many factors including performance, cost, power, reliability and so on. A heterogeneous computing system consisting of general-purpose CPU(s) and special-purpose accelerator(s) features high performance, lower power consumption and low cost, etc. Hence, it has already become the mainstream in the field of high performance computing. However, such systems still face many challenges and problems, for example, programmability and reliability. In this paper, we firstly analyze the main challenges facing heterogeneous computing systems. Then, we introduce the architecture of the first petaflop computing system in China, the Tianhe-1 (TH-1) heterogeneous system, including its hardware/software interface and interconnect network. During development of the TH-1 system, several challenges were encountered; research into the solutions of these challenges is subsequently presented.

Keywords heterogeneous systems, cooperative computing, Tianhe-1(TH-1) system, load balancing, programming models, low power consumption, fault tolerance

1 Introduction

With the development of accelerator technologies, field-programmable gate array (FPGA), application specific integrated circuits (ASIC) and graphic processing units (GPU) find their applications in many fields. People have begun to consider how to apply accelerators to scientific computing, i.e., construction of heterogeneous parallel computing systems, combining general-purpose processors with special-purpose accelerators. In 2005, National University of Defense Technology (NUDT) entered this field and designed the first high performance 64-bit stream processor for scientific applications. The processor was successfully developed and equipped in a high performance computing system in 2007 [1]. This research verified the feasibility of the heterogeneous parallel architectures for high performance computing.

In June 2008, IBM announced a supercomputer, named Roadrunner [2], which achieved a practical performance of over 1 petaflop measured by the Linpack benchmark. The main characteristic of Roadrunner is its heterogeneous parallel architecture. It uses 6480 dual-core AMD Opteron chips as the main processors, and 12960 IBM Cell Enhanced Double-Precision (EDP) chips as the accelerator components (also known as the computing array). Taking advantage of the novel heterogeneous parallel architecture, Roadrunner achieves a performance-per-watt up to 443 Mflops/W. The layout of Roadrunner shows high efficiency in heterogeneous parallel computing.

In contrast, the earlier IBM Blue Gene/L system adopted the low-power embedded chip PowerPC 440. Although the power consumption of each node in Blue Gene/L is low, its single node performance is also low. In order to achieve a higher overall system performance,

Received April 26, 2010; accepted September 28, 2010

E-mail: xjyang@nudt.edu.cn

Blue Gene/L used a vast number of nodes: the number of the cores reached one hundred thousand.

In November 2009, Cray updated its Jaguar system by using six-core AMD processors, and thus promoted Jaguar's Linpack benchmark performance to 1.759 Pflops. Yet, the power consumption of the Cray system is enormous, reaching as high as 1750 W per square foot. Consequently, Cray has to use advanced water-cooling technology to implement system cooling. Table 1 lists the performance-per-watt results of several current leading high performance computers. It can be seen that heterogeneous parallel systems attain a substantial increase in energy efficiency through relatively low integration and low power consumption.

With diverse development of high-efficient computing architectures, combining general-purpose processors and special-purpose accelerating components to construct heterogeneous systems has gained increasing popularity. On the one hand, general-purpose processors, such as Intel Xeon, AMD Opteron, and IBM Power, take the responsibilities of basic computing, system management and control, etc., which allows heterogeneous systems to support software inheritance well. On the other hand, incorporating special-purpose chips, such as Cell, GPUs and ClearSpeed [3], as the accelerators can provide extended computing capacity and acceleration of specific applications, and thus improve the system's efficiency.

Currently, GPUs, as commercially available graphics accelerator cards, feature high peak double precision performance and have a great market share. For example, the theoretical peak performance of AMD HD4870 reaches 240 Gflops, and a measured Linpack performance-per-watt of above 1 Gflops/W. Hence, the performance of GPUs has greatly exceeded that of general-purpose processors and specific accelerator ASICs such as Cell and ClearSpeed. Building high-performance parallel computer systems using GPUs as accelerators can achieve the goals of low cost and high energy efficiency. Moreover, the software developing environment for GPU continues to evolve, which weighs down other special-purpose accelerators.

In 2009, NUDT developed Tianhe-1(TH-1), the world's first heterogeneous high-performance computer system built with general-purpose processors and GPUs. TH-1 ranked 5th on the TOP500 list and the 8th on the Green500 list in November 2009. The success of TH-1 verifies the advantage of the heterogeneous parallel architecture combining general-purpose processors and special-purpose accelerators.

Heterogeneous parallel architectures show an unprecedented high efficiency and thus open a new era for high performance computing.

Section 2 analyzes the current challenges facing heterogeneous computers. Section 3 describes the hardware and software architecture of TH-1. Section 4 presents our research on the TH-1 heterogeneous system. Conclusions are given in the final section.

2 Challenges for the development of heterogeneous parallel computers

The heterogeneous parallel architecture composed of general-purpose processors and special accelerators is illustrated in Fig. 1. Each computing node contains one or more general-purpose CPUs and several accelerators, and can communicate with other nodes through a high speed network.

Because the structure and performance of general-purpose CPU is quite different from that of the special accelerators, the method chosen for collaboration determines the efficiency of the heterogeneous architecture. In our perspective, there are five major challenges to the technology: programming model, task distribution, memory wall, power consumption and reliability.

2.1 Heterogeneous parallel programming model

It is important to supply a programming model which can enable effective cooperation between the CPU and accelerators for the heterogeneous parallel system illustrated in Fig. 1. Several current parallel programming models are aimed at heterogeneous accelerators, such as

Table 1 Performance-per-watt results of three leading supercomputer systems

	IBM Blue Gene/L	Cray Jaguar	IBM Roadrunner
Architecture	homogeneous (embedded processor)	homogeneous (general processor)	heterogeneous (general processor and special accelerator)
Performance/Pflops	0.478	1.759	1.042
Power/MW	2.33	6.95	2.35
(Mflops/W)	205	253	443

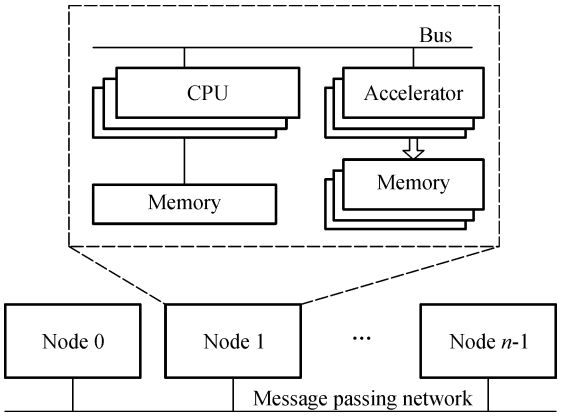

Fig. 1 Heterogeneous parallel architecture

CUDA [4], Brook+ [5], and OpenCL [6]. However, the programming capability is limited and far from satisfactory for large scale parallel programming. So far, MPI is used by most of the parallel systems, and OpenMP is typically used to improve the communication performance in an SMP node. Providing support for accelerators efficiently in the MPI-OpenMP programming environment is a big challenge for system designers.

2.2 Task distribution

In homogeneous parallel systems, due to the same performance of each node and each processor, a simple equal division can meet load balance constraints. But in a heterogeneous system constructed by general purpose processors and accelerators, there is a significant performance difference between them. Moreover, after introducing the accelerators, the workload between different CPUs or different cores in one CPU is also different. Thus, determining how to distribute the workload between CPU and accelerator or among the cores in one CPU is critical to achieve high throughput of heterogeneous system.

2.3 Memory wall

In traditional large scale computer systems, memory read/write speed within a single node is far below CPU processing speed. For instance, one clock cycle of typical memory read/write is 90 ns, but a processor clock cycle takes only takes 0.3 ns. The huge gap between clock speeds greatly degrades the system performance, and hence becomes a bottleneck. Worse still, this performance gap will widen according the Moore's Law. The communication speed between system nodes is also far below the local memory access speed (the ratio is about 2000 ns/ 90 ns). As a result, the efficiency of remote data memory access is rather low, and the performance of parallel computing is greatly reduced. Currently, the efficiency of parallelism for large scale parallel computers in real world application is approximately 5%.

In heterogeneous systems, the floating point computing performance of accelerators is better than general purpose CPUs so if the CPU communicates with the accelerator frequently, the cooperative computing application performance of CPU and accelerators would be seriously reduced by the overhead of memory access and communication. How to solve this memory wall problem in heterogeneous systems to improve the system usability is a great technical challenge.

2.4 Power consumption

High power consumption influences a large-scale parallel system in several aspects, such as system stability, availability and reliability. The cost of system cooling and running will also increase. Hence, a new list, Green500 [7], has been put forward, in which high performance computers are ranked by their power consumption. It gives another view for examining the objectives a high performance computer should pursue. When a heterogeneous system, consisting of general purpose CPUs and accelerators, achieves an extremely high peak performance, it also brings new challenges to system power management. In homogeneous systems, the computing capability and power efficiency of all processors are the same, so it is easier to analyze the time gap and reduce the power consumption with DVFS technology [8]. However, in heterogeneous systems, processors have different computing capabilities and power efficiencies, so we need to provide overall consideration to the heterogeneous processors to make judicious power optimization [9].

Although generally the accelerators have high efficiency, their performance depends on many factors, such as the computing density, the problem size, the memory access pattern and so on. One breakthrough point in system power optimization is to dynamically select execution units for a particular application, so that we can achieve the greatest efficiency of the whole system. On the other hand, with the development of manufacturing technologies, the ratio of static power consumption in the total power consumption of the chip increases steadily.

For example, the static power consumption of modern GPUs is around 40 W. Finding an optimization to reduce static power consumption is also very important to the whole system.

2.5 Reliability

With increasing system integration, system performance is improved; however, system reliability problems become more and more severe. American Los Alamos National Lab (LANL) monitored 22 high performance computers from 1996 to 2005, and statistics showed that the failure rate is approximately once in 512 hours and with average system down time of approximately 2 hours. Through analysis, they found that the storage overhead for repairing the failure is 0.2 hours. If the system is constructed by thousands of nodes, to ensure we obtain the correct result from the system, the cost of fault tolerance will be huge.

Besides the same problem, namely the huge fault tolerance cost, with the homogeneous systems, heterogeneous systems meet new problems because of the introduction of new accelerators. For example, a GPU contains many more integrated computing units than a traditional CPU. An extremely high temperature may be reached during execution which will definitely degrade the reliability of the heterogeneous systems. Whether the traditional fault tolerance methods can be used directly in the heterogeneous system needs to be studied further.

3 Introduction to the TH-1 heterogeneous system

In November 2009, School of Computer Science, NUDT, announced the first petaflop computing system in China, TH-1. The peak performance of TH-1 reaches 1.206 Pflops. The peak performance of TH-1 system involved in the LINPACK test, including the computing array and accelerator array, is 1.157 Pflops, and the Linpack performance is 563.1 Tflops. The computation efficiency reaches 48.67%, and performance-per-watt is 379.24 Mflops/W. In the TOP500 list published in November 2009, TH-1 ranked No.5 (No.1 in Asia), and in the Green500 list published in November 2009, TH-1 ranked No.8.

TH-1 is a multi-array, configurable, cooperative parallel system, composed of high performance general-purpose microprocessors, GPUs, and a high-speed Infiniband network. The TH-1 is an important development we have made in heterogeneous system and cooperative computing.

3.1 Hardware system

The hardware system of the TH-1 includes multiple arrays. Each array has different computation resources, and can provide high performance computing services through flexible configuration. The system framework is shown in Fig. 2.

The system is made up of the computation array, the

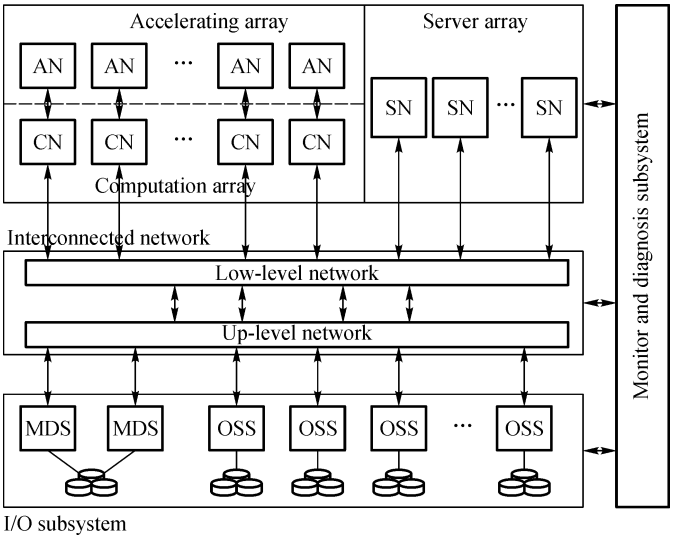

Fig. 2 Hardware system

accelerator array, the server array, the I/O storage system, and the monitoring and diagnosis system. The subsystems are connected by an interconnect network. The detailed specification is listed as follows.

1) The computation array includes 2560 nodes, each of which has two Intel Xeon CPUs and 32 GB memory.

2) The accelerator array includes 2560 GPUs (ATI Radeon HD4870x2), which are used to accelerate scientific computation applications.

3) The server array consists of 512 nodes, each including two Intel CPUs. The server array provides a variety of user services such as login, programming and compilation, resource management, and task distribution.

4) The interconnect network subsystem is a two-level Infiniband QDR network. The bandwidth and latency of a single communication link are 40 Gbps and 1.2 μs respectively.

5) The I/O storage system uses global distributed shared I/O architecture with a total capacity of 1PB.

6) The monitoring and diagnosis system adopts centralized management architecture, and provides the security monitoring, system control, and hardware diagnosis services.

3.2 Interconnect network

The interconnect network uses an Infiniband QDR network. The link bandwidth is 40 Gbps with a speed of 10 Gbps per line. The network latency is 1.2 μs. The Infiniband network supports the communication of computation tasks, I/O access, system management, and has the characteristics of high bandwidth, low latency, and high scalability. The network supports collective communication operations.

The interconnect network of TH-1 is shown in Fig. 3. The network uses a photo-electric hybrid multi-level switch system, which is made up of 64 switching modules (embedded in cabins), 10 first-level switches, and 4 second-level switches. The computing nodes and I/O storage system are connected to the first-level switches. The sever nodes are connected by the switching modules embedded in cabins. All first-level switches and switching modules are connected to the second-level switches by Infiniband QDR uplinks.

The interconnect network of TH-1 forms a logical Infiniband subnet. The subnet uses a centralized route management mechanism, in which one single Subnet Manager (SM) takes charge of subnet configuration, subnet activation, and fault tolerance. The process of the subnet management is divided into four stages: topology discovery, path computation, path dispatch, and route reconfiguration. The Subnet Manager is the kernel of the interconnected network and plays a critical role in improving network performance and reliability.

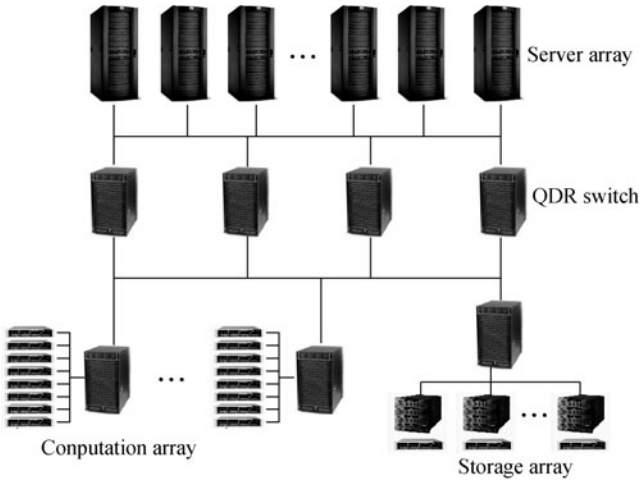

Fig. 3 Interconnect network

3.3 Software system

The software infrastructure of TH-1 includes the operating system, compilation system, resource management system, and parallel programming environment.

1) The operating system of TH-1 is a 64-bit Linux, supporting high-performance parallel computing, energy management, virtualization, and security insulation.

2) The compilation system is made up of multiple programming languages, including heterogeneous programming languages for GPUs. The system can obtain high computation efficiency through several optimizations, such as dynamic task partitioning, streaming load/store, software pipelining, and affinity scheduling.

3) The resource management system provides a uniform global view of the entire system to users, and realizes several strategies of resource assignment and job scheduling, which can efficiently improve the resource utilization and system throughput.

4) The parallel programming environment provides an integrated graphical user interface, facilitating debugging and performance analysis.

4 Key technologies of high-efficiency heterogeneous systems

The TH-1 heterogeneous system supports high efficient

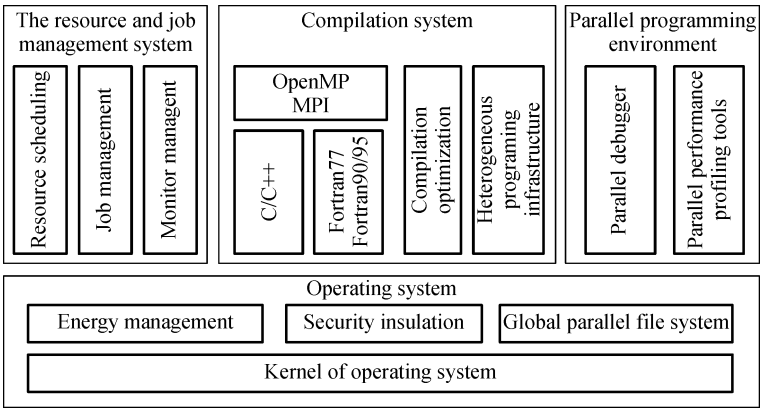

Fig. 4 Software system

cooperating computing. During the prophase research and development of the TH-1 system, we achieved a series of technical breakthroughs in heterogeneous programming models, task scheduling, low-power optimization and fault-tolerance computation.

4.1 Heterogeneous programming model

Hybrid programming models including MPI and OpenMP are widely adopted on large-scale parallel computing systems. These systems are generally made up of several homogeneous nodes connected by the message passing network. MPI is used to exploit coarse-grained parallelism across multiple nodes and OpenMP is used to exploit fine-grained parallelism within a single node. The MPI-OpenMP hybrid programming model has become the industry standard and a huge number of applications have been developed with this model.

GPUs introduce a new problem to programming within a node. It is a big challenge to port traditional OpenMP programs onto CPU-GPU heterogeneous nodes quickly and efficiently. In order to inherit from existing OpenMP applications, we extend OpenMP with a group of GPU-oriented compiler directives, so as to explicitly map the most time-consuming computing fragments of the programs onto the GPU. The OpenMP directives are mainly used to denote those parallelizable loops; which means the OpenMP programs can be naturally translated to GPU programs. We call the extended OpenMP OpenStream. Programming with OpenStream requires only a small modification to original OpenMP programs, and thus significantly reduces the cost of program porting. The extended directives and clauses are listed in Table 2.

An OpenStream compiler for AMD's high performance GPUs is implemented based on GCC 4.2, which already supports OpenMP. The compiler performs a source-to-source translation of the GPU-mapped program fragments into Brook+ programs. The framework of the compiler is shown in Fig. 5.

Table 2 Extended directives and clauses in OpenStream

Name	Syntax	Clauses
SB	!$omp stream begin	streamout, if
SE	!$omp stream end	
KB	!$omp kernel begin	depth, reduction gather, scatter, if
KE	!$omp kernel end	

We extend two kinds of node in GCC's abstract syntax tree system to present the stream region and kernel region qualified by the directives. Using these clauses, the compiler performs stream transformation on the array references, scalar references and loop variables respectively, then generates the kernel function.

Besides the basic translation, the compiler also performs several optimizations, including stream and kernel optimizations as well as the optimizations of data communication between the CPU and the GPU.

The optimizations of stream and kernel include: long stream tiling based on the loop tiling technique, which resolves the long stream problem and hides the transfer latency; kernel fusion, based on the loop fusion technique, is used to increase the computing density of the kernel and transform memory level locality into register level locality; kernel splitting, based on the loop distribution technique, splits a large kernel into several smaller ones to reduce the number of general purpose registers each kernel needs, thus improving the parallelism under limited resources.

Optimizations of off-chip data communication are used to eliminate unnecessary data transfer between the CPU

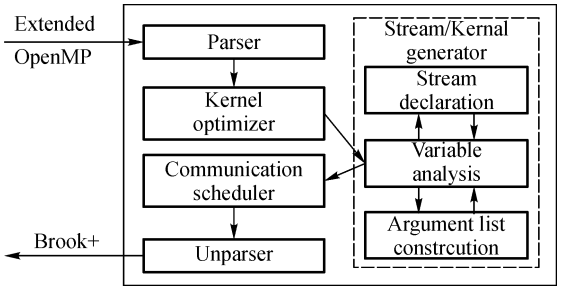

Fig. 5 OpenStream compiler framework

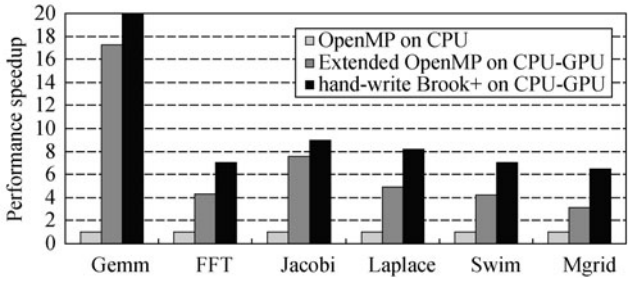

Fig. 6 Performance speedups

and the GPU; otherwise this is prone to become an execution bottleneck since the chips are connected by a peripheral bus. In the basic transformation step, the compiler inserts data transfer statements before and after each kernel invocation. For optimization, the compiler performs data flow analysis on a special control flow graph which treats each kernel as a basic block. The definition information of each basic block is transferred along the flow graph. Once the definition information on the incoming edge of one node conflicts with the operation in the node, a proper data transfer statement will be inserted according to the definition information. In this manner, the data transfer inside a loop will be scheduled outside the loop if possible.

So far, our compiler has succeeded in handling a series of applications, including two standard benchmarks (Swim and Mgrid from SPEC OMP2001) and several typical application kernels (FFT, Gemm, Jacobi and Laplace) in the field of scientific computing on a heterogeneous system incorporating an Intel Xeon E5405 CPU and an AMD FireStream 9250 GPU. The experimental results show that programming with the extended directives deviates from programming with OpenMP by less than 11% modification, and achieves significant speedup ranging from 3.1 to 17.3 on the heterogeneous system over the Xeon CPU alone. Fig. 6 gives the performance speedup results.

The results in Fig. 6 show that the compiler can efficiently accelerate the program execution on CPU-GPU heterogeneous platforms. For some computationally intensive applications such as *gemm* and *jacobi*, the performance of the compiler-generated version is even close to that of the hand-written version.

4.2 Task partitioning and dynamic load-balancing

In order to increase the parallel efficiency of CPU-GPU heterogeneous systems, we need to partition the tasks between the CPU and GPU as well as onto different cores of the multi-core CPU. Task partition is very important to the efficiency of the heterogeneous system. Since the computing ability varies greatly between CPU and GPU, the tasks should be partitioned dynamically according to the computing ability and varying dynamic load.

To avoid load imbalance between CPU and GPU as well as different cores of CPU, we propose a dynamic, two-level task partitioning method, including CPU-GPU partitioning and core-core partitioning, so as to improve the overall performance of the TH-1 system.

The key problem of CPU-GPU task partitioning is that each task block size is different (possibly by a large factor), which means that the partitioning strategy of one task block cannot be reused for the next. Assume *Gsplit* is the proportion of GPU workload to total workload. We assign *Gsplit* with different values according to different data sizes and keep updating the value based on the measured performance of CPU and GPU, so as to reach the load-balance point. We first initialize *Gsplit* under each data size with an experiential value according to parameters including data size, CPU peak performance, and GPU peak performance. Then we update the *Gsplit* according to the actual performance of CPU and GPU during execution, which will be used to guide the partitioning strategy for the next task block. As for the task partition across CPU cores, a static method is not efficient. Some cores tend to exchange data with GPUs which weakens their computing ability. The dynamic task partitioning method for CPU cores is similar to that for CPU-GPU except that the computing ability of a CPU core is not as sensitive to data size; so we do not need to maintain a respective proportional value for each data size. Assume $Csplit_i$ is the proportion of the ith CPU core workload on the CPU workload. Initially, we distribute the CPU workload across all cores equally. Then we update $Csplit_i$ according to the actual performance of each core

during execution, which will be used to guide the partitioning strategy for the next task block.

Formally, let S be the computing workload, N be the number of CPU cores, *Gsplit* be the proportion of the GPU workload to the total workload, $Csplit_i$ be the proportion of the ith CPU core workload on the CPU workload, S_{cpu} and S_{gpu} be the workload of the CPU and GPU respectively, and $S_{cpu[i]}$ be the workload for the ith CPU core. Initially, we have

$$S_{gpu} = S \times Gsplit,$$
$$S_{cpu} = S - S \times Gsplit,$$

and

$$S_{cpu[j]} = (S - S \times Gsplit) \times Csplit_j.$$

After the tasks are finished on the heterogeneous system, we collect the execution time

$$(T_{gpu}, T_{c[0]}, T_{c[1]}, ..., T_{c[N-1]}),$$

where T_{gpu} denotes the GPU execution time and $T_{c[j]}$ denotes the jth CPU core execution time. We denote

$$T_{cpu} = \max(T_{c[0]}, T_{c[1]}, ..., T_{c[N-1]}),$$

to be the CPU execution time.

We can calculate the GPU performance factor,

$$P^{\Delta}_{gpu} = S_{gpu}/T_{gpu} = S \times Gsplit_i/T_{gpu},$$

and the CPU performance factor,

$$P^{\Delta}_{cpu} = S_{cpu}/T_{cpu}$$
$$= (S - S \times Gsplit_i)/\max(T_{c[0]}, T_{c[1]}, ..., T_{c[N-1]}).$$

Then we update the *Gsplit* value with the formula

$$Gsplit^{\Delta}_i = \frac{P^{\Delta}_{gpu}}{P^{\Delta}_{cpu} + P^{\Delta}_{gpu}},$$

and the $Csplit_j$ value with the formula

$$Csplit^{\Delta}_j = P^{\Delta}_{cpu[j]} / \sum_{k=0}^{N-1} P^{\Delta}_{cpu[k]}.$$

The updated value is used to guide the task partitioning strategy for the next task block.

Compared with the existing task partitioning method, our method has the following advantages:

1) During the execution of large-scale scientific applications, it chooses an appropriate partition strategy according to data size and provides a good load-balance between CPU and GPU.

2) In view of the different computing capabilities of CPU cores, the dynamic partitioning method can sufficiently exploit the computing ability of all cores and provide good load-balancing.

In order to verify the effect of our dynamic task partitioning method, we perform several experiments on a CPU-GPU heterogeneous platform, which includes 2 Intel Xeon 5540 CPUs and 2 ATI Radeon HD4870 GPUs. The CPU works at 2.53 GHz and the GPU works at 625 MHz. We implement the Linpack benchmark using the AMD CAL programming model. The peak performance and measured performance under different platform configurations are shown in Fig. 7. With the dynamic task partitioning method, the computing efficiency reaches 70.27% under 1CPU-1GPU, 70.46% under 2CPU-1GPU, and 66.5% under 2CPU-2GPU configuration.

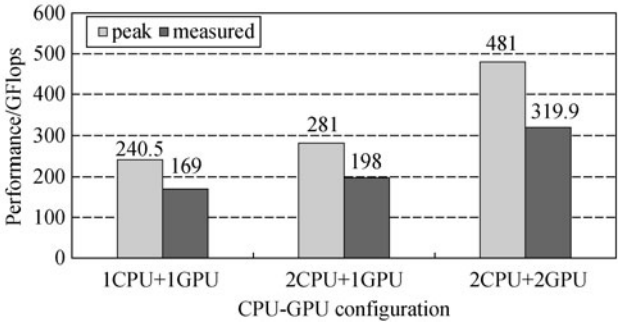

Fig. 7 Peak performance and Linpack performance under different configurations

4.3 Integrated power management framework

Fig. 8 shows the integrated power management framework for TH-1 system. The integrated framework mainly consists of a power consumption sampling module, a power utilization status detecting module, a multi-level low-power optimization module and a power control module. According to statistics on resource utilization provided by the processor sampling unit or QoS provided by the operating system, and the system events provided by external sensors, the power consumption detection module calculates the system power consumption, based on a pre-defined relationship between events and power information. After that, the module sends the CPU utilization data to the power utilization status detection module. Based on this information, the power utilization status detection module determines the appropriate power optimization strategy, and notifies the required multi-level power optimization modules. The power optimization

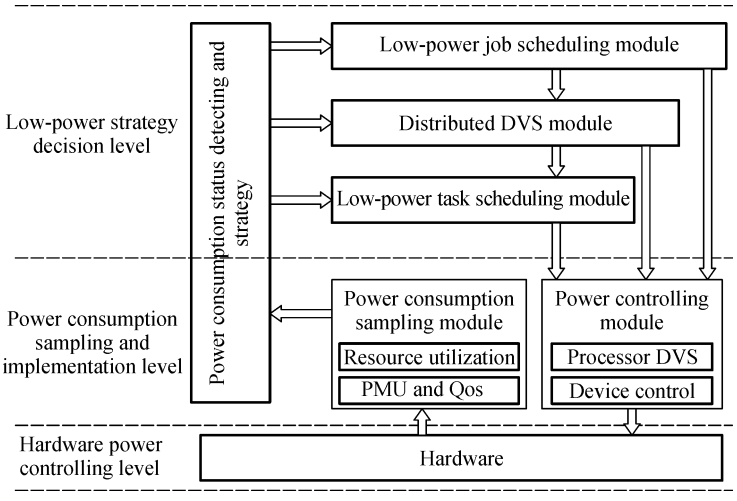

Fig. 8 Integrated power management framework

modules include a low power job scheduling module, a distributed DVS module in kernel and a task scheduling module. The low power optimization module carries special optimization method according to the low power strategy. Finally, the power control module implements processor DVS control and device control through ACPI.

The integrated power management framework provides an application-transparent power management strategy. Under this framework, the compiler provides specific power optimizations for an application. The compiler can analyze the program behavior, and tune the processor's power utilization through DVFS operation in a finer granularity manner. In the CPU-GPU heterogeneous system, the GPU deals with computing-intensive tasks, while the CPU commonly controls the GPU and finishes residual computation. Owing to this difference, there commonly exists an unbalanced task distribution. So we can utilize the slack time to lower the frequency of the lightly loaded processor.

Fig. 9 gives a typical case of CPU-GPU cooperative computation. Before calling the kernel on GPU, the CPU has to load the input data to the GPU space; when the GPU finishes the kernel execution, the CPU will save back the output data.

The total execution time is

$$T = \max\left(T_C, T_R + \sum_{i=1}^{N} T_{Gi} + T_W\right),$$

where T_C denotes the CPU execution time, which is a function of CPU's frequency, denoted as $T_C = g_C(f_C)$.

T_{Gi} is the execution time of the ith kernel on GPU, and

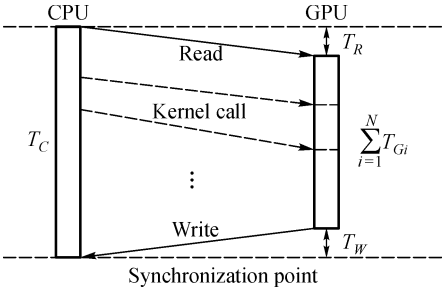

Fig. 9 CPU-GPU cooperative computation

also a function of the GPU's frequency, denoted as

$$T_G = g_G(f_C).$$

The symbol T_R is the data transfer time from CPU to GPU, while T_W represents the inverse data transfer time. According to the difference in workloads for the CPU and GPU, we can divide the problem into two cases:

If $T_C < T_R + \sum_{i=1}^{N} T_{Gi} + T_W$, it means that the CPU has a relatively light workload, then we should scale down the CPU frequency to T_C/T of the original one.

Otherwise, if $T_C > T_R + \sum_{i=1}^{N} T_{Gi} + T_W$, we should scale down the GPU's frequency. That is scaling down the execution frequency for the kernel subset N' to minimize the following expression

$$\sum_{i \in N'} E_{Gi} + 2|N'|E_{tran},$$

which satisfies the following scheduling length constraint

$$\sum_{i=1}^{N} T_{Gi} + 2|N'|T_{tran} \leqslant T_C - T_R - T_W,$$

where E_{tran} and T_{tran} represent energy and time overheads during the running level transition, and E_{Gi} denotes the energy consumption of the ith kernel execution.

In order to evaluate the effect of the proposed power optimization method, we extract a parallel computation section from a real job. In this program, the CPU and GPU respectively deal with matrix reduction and matrix multiplication operations. There exists a synchronization statement after their individual computation. In different processing stages, the problem size for each processor is variable. When the ratio of the problem size is 2048∶512, the CPU has the lighter workload. When the ratio is 3072∶512, the GPU has the lighter workload. For each problem size, we tune the frequency for the lighter workload processor according to the proposed method, and reduce the system power without sacrificing the performance, as listed in Table 3.

Table 3 Power optimization result

Prob. size (CPU:GPU)	Orig/watt	Opt/watt
2048 : 512	308	288
3072 : 512	308	279

4.4 Fault tolerance

Compared to homogeneous systems the reliability of heterogeneous architecture is also different. As an accelerator, a GPU is designed for high performance, but the working temperature of a GPU is very high because of its high density of functional units and execution of computing-intensive programs, so its reliability is poorer than that of traditional CPUs. In traditional fields of graphics processing, due to the particularity of applications, defective GPUs are acceptable. In this case, the designer does not need to consider any fault-tolerance technique for GPU. But in the field of general-purpose computing, especially scientific computing, the reliability of the applications must be guaranteed [10].

When a GPU becomes faulty, it does not return any information to the CPU, so it is difficult to detect the fault and perform fault-tolerance. To improve the reliability of the TH-1 system, we propose a fault tolerance mechanism of dual modular detection and checkpoint-recovery.

There is no ECC in the memory system of a GPU, so when the TH-1 system is running, some transient faults may cause silent error data, which is a non-fail-stop transient fault. We introduce a new fault tolerance mechanism to address it. Using this mechanism, we use time dual modular redundancy to detect fault and checkpoint-recovery to recover from faults for each kernel.

A non-fail-stop transient fault does not influence the state of program execution and is usually detected by redundancy [11]. Redundancy methods can be categorized into time redundancy and space redundancy. In the TH-1 system, we design a time redundancy fault detection mechanism for the kernels. First, the memory system of GPU is not reliable, so all GPU data is duplicated. There are two copies of each data of GPU: one is called the original version and the other the redundancy version. Second, we invoke each kernel twice continuously, and all GPU data of the second kernel calls are the redundancy version. Finally, when the execution of the two kernels is finished, both results are written back to a CPU, and compared. If the two copies of results are not consistent, a fault recovery process is performed.

In the model of a transient fault, a simple and efficient fault recovery method is checkpoint-recovery [12]. Upon detecting a fault, the checkpoint-recovery method rolls back the program to the previous checkpoint, recovers the context, and recomputes the operation. When using the above fault detection mechanism for kernels, the results of each kernel are written back to the CPU, so there are GPU data on CPU. We call the GPU data on the CPU the shadow data. To perform checkpointing efficiently, instead of deleting the shadow data immediately after data comparison, we save the shadow data until the corresponding GPU data are deleted. But when a kernel uses GPU data as simultaneous input and output, we still require a duplicate copy prior to kernel execution so as to exactly recover the GPU context. We call this copy of data the backup data. The flow chart of one kernel execution is illustrated in Fig. 10.

5 Conclusions

Heterogeneous parallel systems integrating general purpose CPUs and special purpose accelerators have several advantages such as: high performance, low power, and low cost; which brings them into the mainstream in the high performance computing field. In this paper, we discuss the technical challenges that the heterogeneous

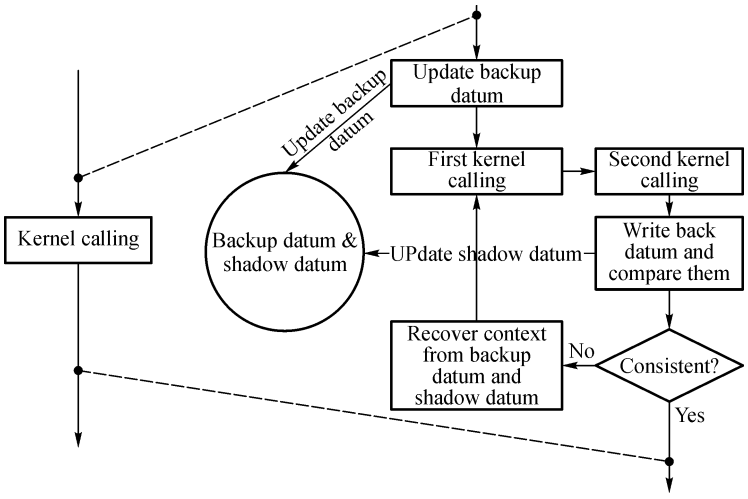

Fig. 10 Flow chart of dual-modular detecting & checkpoint-recovery for simple kernel

parallel systems face, including: programming model, task distribution, system memory wall, and power optimization. We also provide an introduction to the hardware and software architecture of the TH-1 system, and present several pivotal technical breakthroughs during the research and development of the system.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No.60921062).

References

1. Yang X, Yan X, Xing Z, et al. A 64-bit stream processor architecture for scientific applications. In: Proceedings of 34th Annual International Symposium on Computer Architecture. 2007, 210–219
2. Barker K J, Davis K, Hoisie A, et al. Entering the petaflop era: the architecture and performance of Roadrunner. In: Proceedings of 2008 ACM/IEEE Conference on Supercomputing. 2008, 1–11
3. ClearSpeed Technology plc. ClearSpeed whitepaper: CSX processor architecture. 2007, http://www.clearspeed.com/docs/resources/
4. Kirk D. Nvidia cuda software and gpu parallel computing architecture. In: Proceedings of 6th International Symposium on Memory Management. 2007, 103–104
5. Advanced Micro Devices, Inc. AMD Brook+. http://ati.amd.com/technology/streamcomputing/AMDBrook- plus.pdf
6. Munshi A. OpenCL: parallel computing on the GPU and CPU. Presentation at SIGGRAPH, 2008, http://s08.idav.ucdavis.edu/munshi-opencl.pdf
7. http://www.green500.org
8. Semeraro G, Magklis G, Balasubramonian R, et al. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In: Proceedings of 8th International Symposium on High-Performance Computer Architecture. 2002, 29–42
9. Luk C, Hong S, Kim H. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In: Proceedings of 42nd Annual IEEE/ACM International Symposium on Microarchitecture. 2009, 45–55
10. Dimitrov M, Mantor M, Zhou H. Understanding software approaches for GPGPU reliability. In: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units. 2009, 94–104
11. Oh N, Shirvani P, McCluskey E J. Error detection by duplicated instruction in super-scalar processors. IEEE Transactions on Reliability, 2002, 51(1): 63–75
12. Norman A N, Choi S, Lin C. Compiler-generated staggered checkpointing. In: Proceedings of 7th ACM Workshop on Languages, Compilers, and Runtime Support for Scalable Systems. 2004, 1–8