Accelerating Spectral Calculation through Hybrid GPU-based Computing

Jian Xiao*, Xingyu Xu*, Ce Yu*, Jiawan Zhang*, Shuinai Zhang†, Li Ji†, Jizhou Sun*
*School of Computer Science and Technology
Tianjin University, Tianjin, China 300072
Email: {xiaojian, xingyuxu, yuce, jwzhang, jzsun}@tju.edu.cn
†Purple Mountain Observatory
Chinese Academy of Sciences, Nanjing, China 210008
Email: {snzhang, ji}@pmo.ac.cn

***Abstract*—Spectral calculation and analysis have very important practical applications in astrophysics. The main portion of spectral calculation is to solve a large number of one-dimensional numerical integrations at each point of a large three-dimensional parameter space. However, existing widely used solutions still remain in process-level parallelism, which is not competent to tackle numerous compute-intensive small integral tasks. This paper presented a GPU-optimized approach to accelerate the numerical integration in massive spectral calculation. We also proposed a load balance strategy on hybrid multiple CPUs and GPUs architecture via share memory to maximize performance. The approach was prototyped and tested on the Astrophysical Plasma Emission Code (APEC), a commonly used spectral toolset. Comparing with the original serial version and the 24 CPU cores (2.5GHz) parallel version, our implementation on 3 Tesla C2075 GPUs achieves a speed-up of up to 300 and 22 respectively.**

***Keywords*-numerical integration; load balancing; GPU; hybrid architecture; spectral calculation;**

I. Introduction

Essentially all information about astronomical objects outside the solar system comes through the study of electromagnetic radiation (light) as it reaches us. The observed spectrum contains a multitude of important information about star temperature, age, metal abundance and stellar composition etc [1]. So it is a common task for modern astronomers to fit the observed spectrum with the spectrum calculated from theoretical models in order to verify their researches.

One of the most important spectra is the Radiative Recombination Continuum (RRC) [2] from a hot plasma, where an electron collides and recombines with an ion, emitting a photon in the process. The calculation of RRC is based on atom models and radiation mechanisms. Equation (1) describes the spectrum of the RRC:

$$\frac{dP}{dE} = n_e n_{Z,j+1} 4(\frac{E_\gamma - I_{Z,j,n}}{kT})\sqrt{\frac{1}{2\pi m_e kT}} * A$$
$$A = \sigma_n^{rec}(E_\gamma - I_{Z,j,n}) exp(-\frac{E_\gamma - I_{Z,j,n}}{kT})E_\gamma \tag{1}$$

where P is the emitted power, n_e is the electron density, $n_{Z,j+1}$ is the density of the ion $(Z, j+1)$ (here Z is the atomic number of the ion, and $j+1$ is the ionization state), E_γ is the energy of the emitted photon, $I_{Z,j,n}$ is the binding energy for an electron in level n of the ion (Z, j), m_e is the mass of an electron, $\sigma_n^{rec}(E_\gamma - I_{Z,j,n})$ is the recombination cross section to level n at the electron energy $E_\gamma - I_{Z,j,n}$. In order to obtain a high-resolution spectrum in a broad range of wavelengths, the practical method would be to integrate the emissions over a sufficient number of the energy bins for all levels of all ions:

$$\Lambda_{RRC}(E_{bin}) = \int_{E_0}^{E_1} \frac{dP}{dE}(E)dE \tag{2}$$

where E_0, E_1 are the minimum and maximum energies for the bin respectively.

For the one-dimensional definite integral like (2), many classical numerical integration algorithms can solve it efficiently. However, for a real-world spectrum calculation even with a moderate parameter space, these simple tasks will become overwhelming.

Fig.1 presents a typical structure of spectrum calculation. There is a three-dimensional parameter space: temperature, density and time. The parameter space is often given by a result of astrophysical simulation or a configuration file. For each grid point in the parameter space, the RRC integrations are required to perform in three nested loops. Considering the most abundant elements in the universe which totally contain 496 ions, and thousands different energy levels in each ion (theoretically there are an infinite number principal energy levels and sub-levels, in practice, some methods of cutting off the level calculation is necessary), and 10^5 energy bins in each energy level (the count of energy bins is application specific, and 10^5 is of moderate size), the total amount of RRC integrations in each grid point is up to 10^8 order of magnitude.

According to the experiments performed on a single processor of an Intel® Xeon® E5-2640 (2.5GHz), the average time of one grid point is nearly 800s, and the profiling result based on GNU gprof shows that the integral operations account for more than 90% of the total time, consequently for a parameter space of a moderate real-world astrophysical simulation containing 128^3 sampled points, it

0190-3918/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPP.2015.13

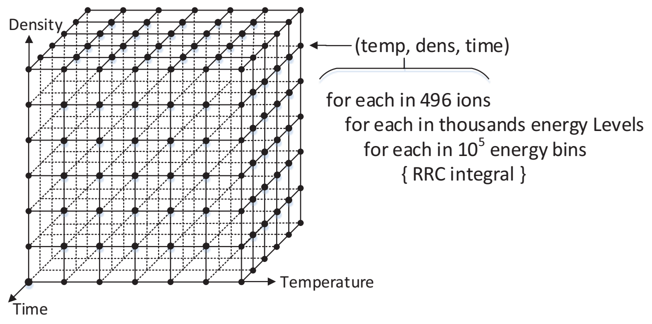


Figure 1: The parameter space and main program structure of the spectrum calculation. Each solid black dot represents one group of determinate input parameters (temperature, density, time), and for each point, the number of RRC integrations within three nested loops is up to $2.0 * 10^8$

will take approximately 0.5 millions CPU hours. Thus many trivial tasks make a huge workload.

Currently, quite a few tools had been developed to model, calculate and analysis the electromagnetic spectrum in astrophysics, and the widely used ones include the Interactive Spectral Interpretation System (ISIS) [3], the XSPEC [4][5], the XSTAR [6], and the Astrophysical Plasma Emission Code (APEC) [7]. With the development of modern telescopes and HPC systems, the observed data explosively increase and high-resolution astrophysical simulations also become possible, consequently the spectral calculation proportionally increase. However, all of these tools come from legacy systems based on traditional CPU-only architecture, which could not efficiently tackle so many computing-intensive tasks of spectral calculation, and many new functionalities have been integrated into them over the past decades, except for well adaptation to the modern heterogeneous HPC architecture.

Meanwhile GPU-based high performance computers have gained popularity in scientific computing as a low cost and powerful parallel architecture in the last decades, and the use of GPUs has proven to significantly increase the performance in numerous applications, including solving large differential equations [8][9] and high-dimensional numerical integrations [10][11]. However, the spectral calculation has two distinct characteristics that common GPU-based numerical integration schemes [10][12] seldom address:

1) Each single one-dimensional integral computing is very small and fast, but there are huge amounts of small integrations.
2) Classical load balancing approaches for CPU-GPU hybrid architecture may be not efficient to schedule so many small tasks due to the extra overhead proportional to the frequency of scheduling.

The existing GPU-based numerical integration schemes are efficient to solve large high-dimensional integration, and the main reason is that the compute-intensive GPU portion accounts for the majority of the total running time while the communication between device and host only plays a minor role. On the contrary, the calculation of one-dimensional RCC integral is trivial to modern GPUs, but the overhead of launching GPU kernels frequently introduced by the large number of small integrations is non-trivial. Our experiments verified that the GPU-based integration algorithm has not much advantage in performance if task scheduling unit is single RRC integral because of the excessive memory-copy overhead between GPU and CPU. So the expected acceleration may not be achieved just by transplanting the numerical algorithm from CPU to GPU directly if lacking consideration in the above characteristics of the problem.

This paper contributes in the following three aspects:

1) Proposed a GPU-CPU hybrid parallel framework to accelerate the spectral calculation and moreover the framework is adaptable to many other compute-intensive applications consisting of many small tasks.
2) Developed a dynamic load balancing scheme between CPUs and GPUs via share memory for a large number of similar and small tasks.
3) Evaluated the efficiency and the accuracy of the proposed approach based on the widely used spectral calculation package–APEC.

The outline of the paper is organized as follows. The related work is presented in Section II. The detail description of the proposed method is provided in Section III. The experiment and performance evaluations are discussed in Section IV. Finally, the conclusion is given in Section V.

II. Related Work

Previous studies in three topics that are most related to our research are reviewed. Firstly, GPU-accelerated numerical integration algorithms are reviewed. Secondly, related load balancing strategies between CPU and GPUs are revisited. Lastly, existing tools to solve spectral calculation are surveyed.

A. Numerical Integration on GPU

Numerical integration constitutes a broad family of algorithms for calculating the numerical value of a definite integral. Many algorithms have been developed and presented in the standard numerical libraries such as QUADPACK [13], MKL [14], CUBA [15] and GSL [16]. In the last several years, as GPUs gained increasingly general-purpose capabilities and steady performance growth, many classical algorithms already had their GPU-accelerated version. Kamesh et al. [10][11] reused the CHURE algorithm, and proposed a deterministic and memory efficient parallel algorithm on GPU to solve the adaptive multi-dimensional numerical integration. Evren et al. [17] presented an efficient implementation of Arakawa's formula using vectorized Streaming SIMD Extension and Advanced

Vector Extension instructions, and achieved nearly two-fold performance improvement. Daniel's [12] group developed a novel heuristic adaptive quadrature that is better suited for accelerating massively-parallel calculation on GPUs. All of these works had a dramatic performance in large multidimensional integrations, but they are not originally designed for intensive small tasks, and for such a task, the communication overhead between host and device may be higher than the computation itself. So it is very important to minimize the frequency of data exchange between CPU and GPU as well as maximize the computation utility of GPU in the GPU-optimized solution.

B. Load Balancing between CPU and GPU

The heterogeneous multi-core CPU and multi-GPU systems increase the difficulties of automatic task scheduling between CPUs and GPUs. Shuo et al. [18] implemented dynamic load balancing for Fast Fourier Transform computation by splitting the total execution into several primitive sub-steps in either GPU or CPU, according to their performance model and heterogeneous execution flow. Giuliano et al. [19] proposed a parallel adaptive algorithm for the computation of multi-dimensional integral on heterogeneous GPU and multi-core based systems but only provided load balancing among threads on multi-core CPU. Yu's method [9] dynamically assigned the integrations to either the CPU-based implicit solver or GPU-based explicit solver based on a real-time estimated stiffness ranking of the equations to be solved. The most advantage of the method is that each task will be dispatched to the solver more suitable and efficient for it.

Wang proposed co-scheduling with asymptotic profiling (CAP) strategy [20] for data-parallelism, which adopts the profiling technique to predict performance and partitions the workload according to the performance. Alejandro et al. [21] developed a dynamic load balancing library that allows parallel code to be adapted to a wide variety of heterogeneous multi-GPU systems. Computation tasks can be distributed to multiple GPUs by message passing or share memory, and this method is quite efficient when all tasks have the same size. Bo et.al. [22] presented an effective task distribution model for embarrassingly parallel problem on heterogeneous CPU/GPU clusters. The main idea of the task distribution model is simple: do not idle the microprocessors with higher computing capability unless the idleness cannot be avoided. A load balancing factor based on the fluctuation of CPU working time and experimental data made their model more effective. The load balancing scheme adopted in our paper is familiar with Bo's model.

It is worth pointing out that the Multi-Process Service (MPS) [23] offered by NVIDIA also can be used to balance workloads between CPU and GPU. The MPS is a binary-compatible client-server runtime implementation of the CUDA API, and the goal of it is to take advantage of the inter-MPI rank parallelism, and increase the overall GPU utilization. Though the MPS can support multi-GPUs, the client-server architecture will introduce much extra overhead if each task is fast and scheduling is quite frequent like in the spectral calculation.

C. Parallelism in Spectral Tools

There are several classical spectral packages in astrophysics. Arnaud et al. developed the XSPEC and continuously enhanced its functions [4][5]. It is a tool most widely used in X-ray astronomy for spectral fitting, with a legacy spanning more than two decades and hundreds of citations. The superset of the XSPEC is the ISIS [3], which was designed for analysis of high-resolution Chandra X-ray gratings spectra. Both ISIS and XSPEC can obtain process-level parallelism via the ISIS's plug-in PModel, which allows arbitrary components of a broad range of astrophysical models to be distributed across processors. XSTAR [6] is a program for calculating the physical conditions and emission spectra of photoionized gases, and a Parallel Virtual Machine (PVM) wrapper can be used to foster concurrent execution of the XSTAR command line application on independent sets of parameters [24]. By plugging the XSTAR into the PModel, the ISIS can also invoke multiple XSTAR instances simultaneously. APEC is another powerful and widely used tool, which calculates both line and continuum emissivity for a hot, optically-thin plasma in collisional ionization equilibrium [7], but it does not provide parallelism natively.

Up until now, to the best of our knowledge, there is very little research on GPU-accelerated spectral calculation in a heterogeneous multi-CPU and multi-GPU environment.

III. Method

The proposed approach consists of four main parts: a parallel wrapper based on MPI, the CPU-based integration component, the GPU-accelerated integration component, and the task scheduler for dynamically balancing the work loads between multiple CPUs and GPUs. Fig.2 shows the architecture and main workflow of the hybrid parallel approach. The main program is responsible for reading the input parameters, invoke all MPI processes, and assign sub parameter spaces to them. MPI processes will prepare tasks, and dispatch each task to either the CPU-based calculator within its context or a shared GPU calculator through the task scheduler, and finally aggregate result of each tasks.

A. Dynamic Load Balancing

As the core component of our approach, the load balancing strategy is based on a simple but effective idea: try best to make GPU busy. The program is built upon MPI, but for simplicity and stability there is no central load balance server in the parallel program, instead each physical node is equipped with a local task scheduler. The main program is responsible for load balance among the different physical

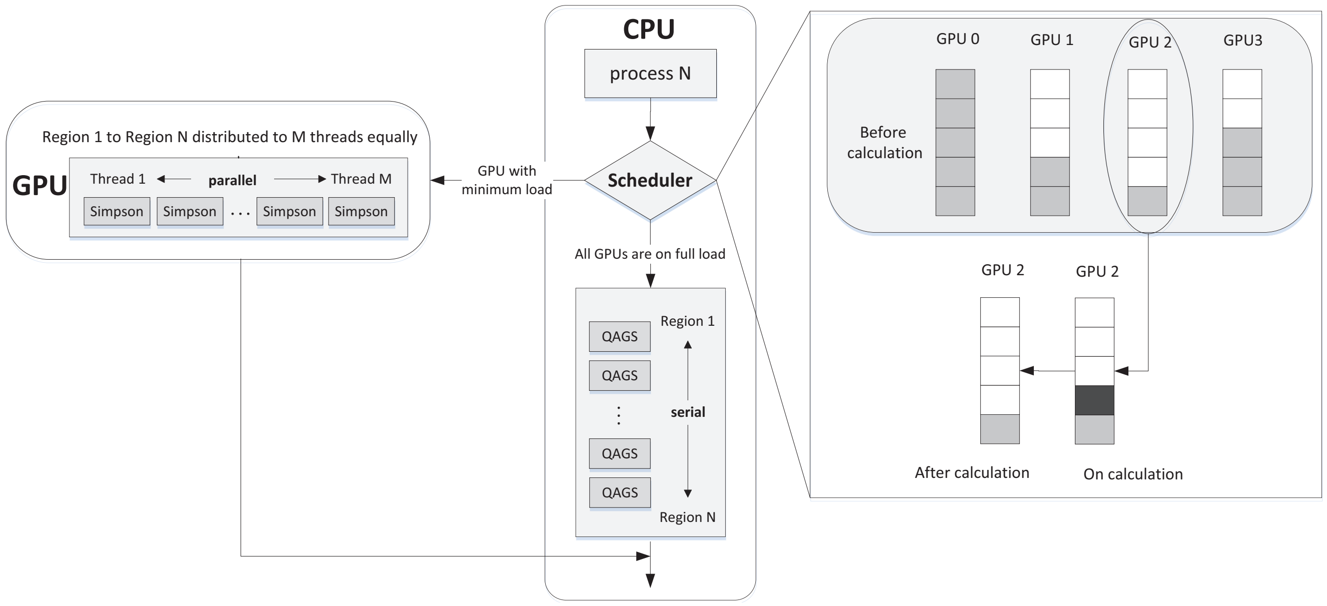


Figure 2: The flow-chart of the hybrid parallel framework for spectral calculation. Each CPU process will try to require a device from the scheduler when it receives a task, and then the scheduler will check the task queue of each GPU, and allocate the task to the GPU with the minimum workload, but if all GPUs are running on full load, the task will be performed at the CPU processor.

machines by dividing the whole parameter space into several equal subspaces, and the task scheduler is only in charge of these tasks within its local node. This strategy is simple but very efficient when the size of all tasks is approximately equivalent such as in spectral calculation.

The basic data structure for load balancing is the task queue, and each GPU device has one private queue. The main terminologies of the task queue are as follows:

- *Active task*: The task that is running on GPU.
- *Waiting task*: The task that will be run on GPU later.
- *Load*: The sum of the current active tasks and waiting tasks.
- *Maximum queue length*: The upper bound of the load, GPUs with full load will not receive tasks.
- *History task count*: The accumulative total count of tasks that a task queue has received so far.

As Fig.2 shows, the scheduler is responsible for maintaining all the task queues and dispatching tasks to the proper GPU or CPU. First, the MPI process will request the scheduler to add its current task into the queue of some proper GPU. Then the scheduler will select a GPU that has the minimum work *load* currently. If there are two or above GPUs with the same *load*, the GPU with the minimum *history task count* will be chosen. If there is a candidate GPU, the task will be performed on it, and the corresponding position of the task queue will keep on being occupied until the calculation finishes. Otherwise, if all the GPUs are busy, that is the *loads* of all GPUs have already reached the *maximum queue length*, the original CPU process will continue to achieve the task by calling traditional QAGS [13] routine serially.

It is worth mentioning that the *maximum queue length* depends on both the computing capability of the device and the application itself. The relation between performance and maximum queue length is detailedly explored in the following experiment section (Fig.4), and in practice, the scheduler chooses the maximum queue length through an automatic test. At the beginning the scheduler will try to find the most proper *maximum queue length* by increasing the value of it gradually until the performance inflexion occurs. And then the *maximum queue length* will be fixed at the value leading to the inflexion point. In addition, task scheduling and concurrency mode inside GPU will be different on different GPU architectures. For example, application-level context switching is necessary on Fermi, that is the queued tasks are performed serially in their submission orders. Meanwhile, the Hyper-Q technique can allow for up to 32 simultaneous connections from multiple MPI processes on some Kepler GPUs, and this feature can get higher effective GPU utilization. So for some Kepler GPUs, the count of *active task* may be more than one.

The detail of the scheduling algorithm is described in Algorithm 1. L is the lower limit of integral, U is the upper limit of integral, N represents the number of integral regions, f_{rrc} is the integrand and $device$ is the GPU chosen to perform integration task.

Algorithm 1 Scheduling

```
1: for all $process_i$ do
2:     $device$ = SCHE-ALLOC();
3:     if $device \geq 0$ then
4:         GPU-Integr($L, U, N, f_{rrc}, device$);
5:         SCHE-FREE($device$);
6:     else
7:         CPU-Integr($L, U, N, f_{rrc}, err_{abs}, err_{rel}$);
8:     end if
9: end for
```

Algorithm Subroutines in Scheduling

```
1: $l_i$ is the load of device $i$;
2: $h_i$ is the history tasks count of device $i$;
3: Both $l_i$ and $h_i$ are global variable;

4: function SCHE-ALLOC()
5:     $l_i, h_i \leftarrow$ shmat();
6:     $l_{min} \leftarrow l_0$;
7:     $h_{min} \leftarrow h_0$;
8:     for all $i < device\_num$ do
9:         $l_{min} \leftarrow min(l_i, l_{min})$;
10:        if $l_i == l_{min}$ then
11:            $device \leftarrow min(h_i, h_{min})$;
12:        end if
13:    end for
14:    if $l_{min} < l_{MAX}$ then
15:        atomic operation {
16:            $l_{device} + +$;
17:            $h_{deivce} + +$;
18:        }
19:        return $device$;
20:    else
21:        return $-1$;
22:    end if
23: end function

24: function SCHE-FREE($device$)
25:     $l_{device} \leftarrow$ shmat();
26:     atomic operation {
27:         $l_{device} - -$;
28:     }
29: end function
```

B. The GPU-accelerated Numerical Integration Algorithm

In order to tackle these large number of small RRC integrations and reduce both data transfer volume and frequency between host and device, we defined a coarse-grained task, and such a task contains tens of thousands RRC integrals. As illustrated in Fig.1, every grid point contains 496 ions, and each ion has different number of energy levels, so it is natural that both the energy level and the ion(one ion can include numerous energy levels) can be used to define the task scope.

If the task granularity is energy level, which usually contains 50k energy bins (integrals) but it still is a relatively fine-grained parallelism, and compared with the data IO between CPUs and GPUs, the kernel computation on GPUs still occupies a small portion of running time. Otherwise, if the task scheduling unit is ion, there are 496 tasks at one parameter grid point. Experiments show that such a granularity of the task can lead to better overall performance for the hybrid parallel approach. It is important to note that the granularity of task is also application specific, for spectral calculation, the optimum granularity is ion, because if element is used (one element includes several ions), the logic of the kernel will become more complex so that it is not suitable to run on GPU.

The pseudo-code of RRC integration is showed in Algorithm 2. Each thread in GPU is responsible for several small integral regions. In each integral region, the classical Simpson method [25] is employed to perform the integration. For each ion, the result of emissivity of each energy level in each energy bin will be accumulated on GPUs until the task is completed, and then transfer the result from GPU to CPU.

Algorithm 2 GPU-Integr ($L, U, N, f_{rrc}, device$)

```
1: $bin\_num \leftarrow N/thread\_num$;
2: $bin\_size \leftarrow (L - U)/N$;
3: $idx \leftarrow threadIdx.x + blockIdx.x * blockDim.x$;
4: while $i < bin\_num + 1$ do
5:     $r_i \leftarrow L + (bin\_num * idx + i) * bin\_size$;
6: end while
7: for all $j < bin\_num$ do
8:     $left \leftarrow r_j$;
9:     $right \leftarrow r_{j+1}$;
10:    $emi_{bin\_num*idx+j} \leftarrow$ Simpson($f_{rrc}, left, right$);
11: end for
   $\triangleright$ $emi$ : a $N$ size array containing emission powers of all energy bins.
```

C. Implementation

This paper's approach is prototyped based on APEC [7], programmed in C and CUDA C. A MPI wrapper is developed for giving the original APEC the capabilities of CPU-based parallel computing and communication with GPU. The CPU integration component based on the classic QAGS routine [13], and a general interface of the GPU-accelerated component is developed, so that different numerical integration algorithms can be connected to the main program on demand. In the current implementation, both the Simpon and the Romberg integration are provided. Due to the pluggable and modular design, the implementation puts a minimal impact on the original APEC code, so only a few code changes are required for users adopting our approach. Furthermore, the program will detect the number of GPU devices automatically, and it can run normally in the runtime environment without GPU device.

In order to avoid the extra overhead in the client-server architecture, the local task scheduler communicates with MPI processes and GPUs via share memory. The shared memory contains two types of arrays, one is the *load* count of task queue on each device, and the other is the *history task count* of each device. When a GPU is selected for a given task, the scheduler will increase the current load value of the GPU by one in an atomic operation. Similarly when the task is over, the load value will be decreased by one and the history task count will also be increased by one in one atomic operation.

IV. Evaluation and Discussion

The experimental environment consists of 2 Intel Xeon CPU E5-2640 (2.5GHz), 12 cores/CPU and 4 NVIDIA Tesla C2075 GPUs. The Tesla C2075 is based on the Fermi architecture with 6GB GDDR5 on-board memory, 448 streaming processor cores (1.15GHz), and delivers up to 515 gigaflops of double-precision peak performance in each GPU. These GPU devices interconnect with the host through PCI Express 2.0.

The test parameter space consists of 24 grid points, and all the tests were executed by 24 MPI processes on one physical node. It is not very meaningful to use much more grid points for testing, the main reason is that the amount of calculation at each point is approximately the same when all of these points locate within a small region, additionally each point has 496 tasks that is enough to verify the effect of load balance.

All the GPU tests used the Simpson integration algorithm except the Table I and the Fig.6. The tests with 1 GPU device are marked with *1 GPU*, in the same way, the tests with 2 GPU devices are marked with *2 GPUs*, and so on. The tests on pure CPU version (serial and MPI only) are not plotted together with these GPU counterparts, because the CPU curves almost overlap with the horizontal axis due to the big speedup (The MPI parallel version with 24 cores can only speed up the computation by a factor of 13.5 relative to the original serial version).

A. *Performance*

Fig.3 plotted the speedup of two task granularities respectively. The coarse-grained task is mapped to an ion of an element, marked as *Ion*. The relatively fine-grained task is mapped to an energy level of an ion, marked as *Level*. The speedup was computed by comparing the total execution time of the hybrid CPU/GPU version against the time used in the original serial APEC. As Fig.3 shows, the speedup of *Level* implementation is good, but the speedup of the *Ion* version gets more considerable increment as the number of GPU devices rises. The frequent memory copy between device and host in the former is the main reason. In the latter, a lot of communication overhead is avoided by moving accumulation operation of all energy levels of one ion into GPU. So for a large number of small tasks, it is important to choose a proper granularity through combining small tasks into a bigger one so as to maximize the utility of GPU.

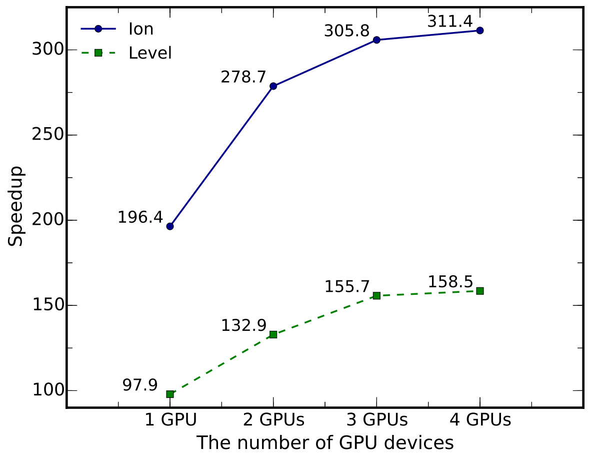


Figure 3: The speedup on different task granularities. The solid line is obtained by mapping an ion to a task (coarse-grained), and the dotted line is obtained by mapping an energy level of an ion to a task (fine-grained).

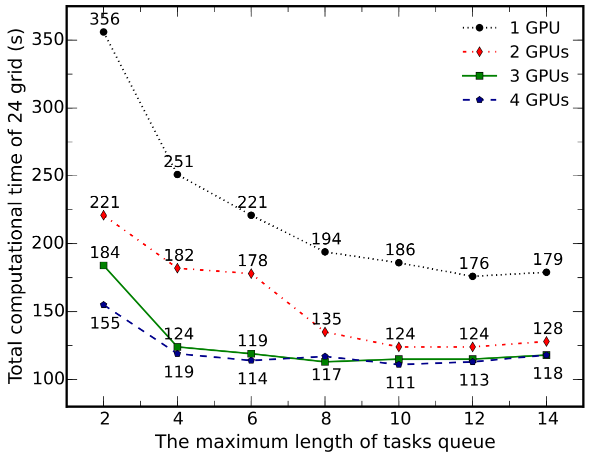


Figure 4: The total computing time with different maximum queue length.

Besides task scheduling granularity, the *maximum queue length* is another important factor that influences the overall performance and the load balancing result. Fig.4 compares the total computing time of 24 grid points with different setting of *maximum queue length* and different number of GPUs. In these experiments, the total number of tasks for scheduling is 24(points)*496(ions), more than ten thousands tasks. As illustrated in Fig.4, the total computing time decreases as the *maximum queue length* increases, and peak performance occurs when the maximum queue length equals 10 or 12 for all testcases, after that the performance begins to drop slightly. This phenomenon can be explained by Fig.5, as the *maximum queue length* rises, more and more tasks are distributed to GPUs, and the computing capability of GPU is far higher than the CPU counterpart, consequently the overall performance becomes better. But as the workload of GPUs continuously increases, task waiting time on GPUs becomes longer and longer while CPUs are idle, so the performance drop occurs. It is worth pointing out that the difference of total computing time between 2 GPUs and 3 GPUs is getting smaller and smaller when the *maximum queue length* is larger than 6. The total computing time between 3 GPUs and 4 GPUs is almost the same. This phenomenon indicates that for the spectral calculation, 2 GPUs is powerful enough to process the request from 24 CPU cores in our test environment, and it is not very helpful for performance improvement by simply adding more GPUs.

B. Load balance

Fig.5 indicates that even the *maximum queue length* is only 2, more than 95% tasks are distributed to GPUs in the above experiments where the Simpson integration algorithm is used. This result depends on both the problem domain and hardware environment. For most cases of spectral calculation, the Simpson algorithm can provide enough accuracy just by dividing the integral range into 64 equal pieces, and this calculation is moderate for the GPUs in our experiments. Therefore, most tasks are dispatched to GPUs. On the other hand, some applications may require higher accuracy, in order to further verify the effectiveness and adaptability of the load balance strategy, another group tests with higher accuracy were performed by employing the Romberg integration [25]. Compared with Simpson algorithm, Romberg algorithm can obtain higher accuracy but without adding any extra computational complexity.

The computational complexity of Romberg integration algorithm is determined by the value of k, as described by (3), which represents the times of dichotomy. As k increases, the accuracy become more higher, and the amount of calculation of a single task will also increase exponentially by a factor of 2. Fig.6 shows the load distribution on GPU device 0 with different k value, and Table I shows the statistical result of tasks distribution between GPU and CPU.

$$T_m^{(k)} = \frac{4^m}{4^m - 1} T_{m-1}^{(k+1)} - \frac{1}{4^m - 1} T_{m-1}^{(k)} \tag{3}$$

In Fig.6, each bar represents the load percentage on one GPU during the complete execution of a test where the

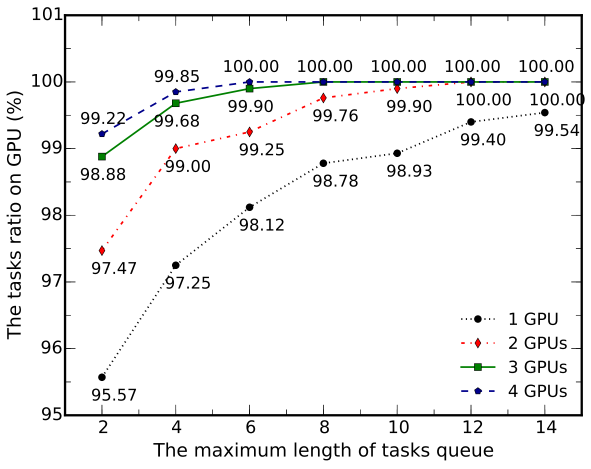


Figure 5: The task ratio on GPUs with different maximum queue length, ratio = tasks achieved by GPUs / total tasks

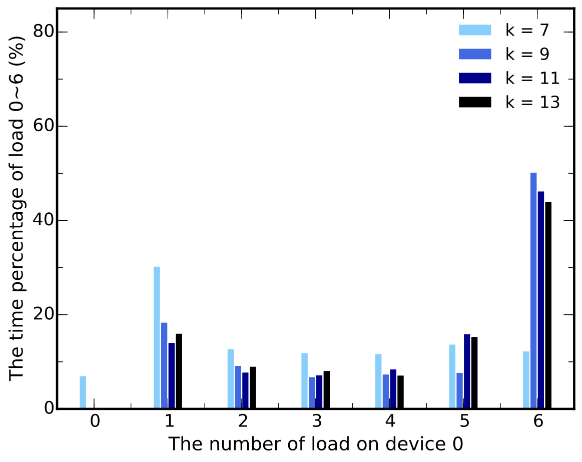


Figure 6: The load distribution on GPU during end-to-end executions with different computational complexities; the total number of GPUs is 2 and the maximum queue length is fixed at 6.

maximum queue length was fixed at 6. For example, the black bar in the bin of load number 6 indicates that when the computational complexity of single task is proportional to 2^{13} RRC integrals, the load staying at 6 takes up the 44.04% of the total running time. According to Table I, when the computation amount of single task is 2^7 , GPU can easily handle more than 98% of the overall work loads, and the loads higher than 3 take up only 37% of the total execution time. But as the computational complexity of a single task rises, the load of GPU also keeps increasing, consequently more and more tasks are dispatched to CPU. This group of experiments shows that the amount of computation contained within a single task has an import impact on the result of load balance. Therefore in order to maximize the end-to-end performance in a heterogeneous CPU/GPU system, it is necessary to choose a proper task partition strategy for specific applications.

Table I: The task distribution ratio on GPU with different computational complexities.

Computation amount/task	The number of tasks on GPU	The ratio of tasks on GPU	GPU load ≥ 3
2^7	6674	98.26%	37.85%
2^9	6344	93.40%	65.46%
2^{11}	4518	66.52%	70.76%
2^{13}	2779	40.92%	66.64%

C. Accuracy

A comparative study of the accuracy of our approach was performed. Fig.7a shows the normalized flux in a wavelength range calculated based on the output of the original serial APEC, and Fig.7b is the result obtained from the proposed hybrid parallel approach. Fig.8 illustrates the quantitative analysis of numerical errors between the two methods. According to the error distribution curve, the relative error value ranges -0.0003% to 0.0033%, and more than 99% errors are located in the interval of 0% to 0.0005%. The error analysis verified that the proposed approach can effectively accelerate spectral calculation without any obvious accuracy loss.

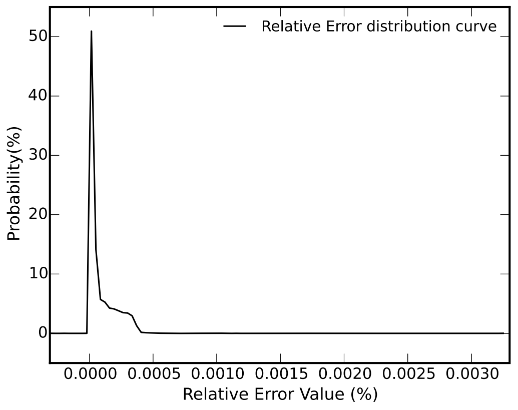


Figure 8: The distribution of numerical error

D. Adaptability

In the following experiment, Non-equilibrium ionization (NEI) problem is used to evaluate the adaptability of the proposed hybrid parallel approach to these compute-intensive applications containing a large number of similar small tasks. NEI is an important phenomenon related to many astrophysical processes, and the basic NEI model is given by the following set of ordinary differential equations (ODE):

$$\frac{\partial n_i^Z}{\partial t} = N_e[n_{i+1}^Z \alpha_{i+1}^Z + n_{i-1}^Z S_{i-1}^Z - n_i^Z(\alpha_i^Z + S_i^Z)] \quad (i = 1, \cdots, N_{spec}) \tag{4}$$

where n_i^Z is the number density of the ion i of the element Z, t is the time, N_{spec} is the total number of species, N_e is the electron number density, $\alpha_i^Z = (N_e, T)$ and $S_i^Z = S(N_e, T)$ are the recombination and ionization coefficients respectively. The details of NEI solving process can refer to our previous work [26]. The characteristics of NEI calculation are as following:

1) At every point of parameter space, there are about a dozen of ODE groups and the size of each group equals the number of ionization states of its corresponding element.
2) α_i^Z and S_i^Z are determined by the number density and temperature of electrons and need to be computed on real time. So the establishment of these ODEs will also consume lots of computing resources.
3) The ODEs of NEI are stiff and sparse.

Even with modern methods for solving the ODEs, calculating the ion abundances of a multidimentional simulation is very expensive in terms of CPU time and computer memory. The testcase used here contains one million grid points and each point evolves 1000 timesteps, and the running environment is the same with the above spectral testcases. In order to utilize the proposed hybrid approach more efficiently, a GPU-accelerated NEI solver is developed based on the classic ODE solver LSODA [27], and every ten time-dependent calculations are packed into one task for reducing the frequency of data copy between host and device.

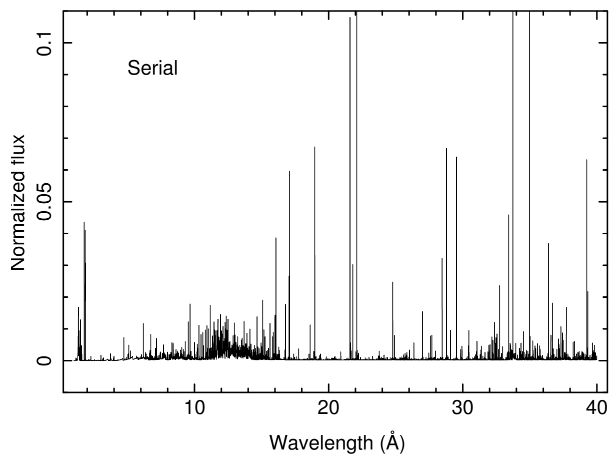
Table II: The speedup of NEI on different number of GPUs.

	1GPU	2GPUs	3GPUs	4GPUs
speedup	2.8	5.9	10.8	15.1
time (s)	3137	1494	810	582

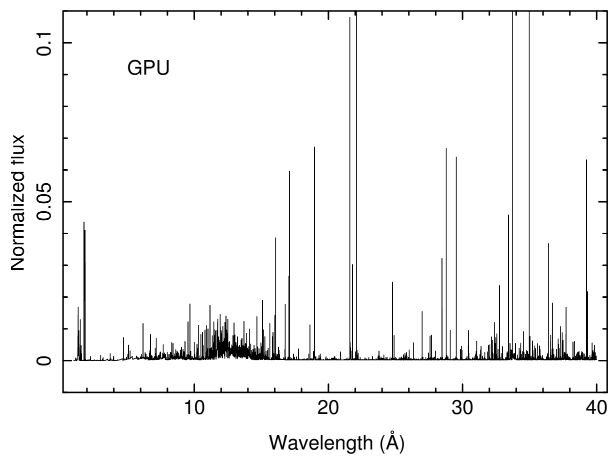
Table II shows the speedup of NEI on different number of GPU devices, and it was computed by comparing the CPU/GPU hybrid parallel method against the pure MPI version with 24 CPU cores. The maximal speedup is about 15 and occurs when the maximum queue length is 8 and the number of GPUs is 4. This experiment indicates that the proposed approach has good adaptability to other compute-intensive applications but the optimal configuration is application specific.

V. Conclusion

In this paper, we proposed a hybrid CPU-GPU parallel approach to accelerate spectral calculation. First, we



(a) The emissivity calculated by serial APEC



(b) The emissivity calculated by hybrid parallel APEC

Figure 7: Comparative result of the two spectral calculations.

offloaded the compute-intensive integral parts of the application to GPUs, and reduced the frequency of memory copy between device and host by combining many single integral operations within one ion into a coarse-grained task. Second, for a large number of small tasks in the spectral calculation, we developed a task scheduling scheme among multiple CPUs and GPUs via share memory that can avoid extra communication overhead in the traditional client-server architecture such as NVIDA's native MPS [23]. Last, comprehensive theoretical analysis and experiments were conducted to verify the efficiency and accuracy of the approach, and the experiments showed that 24 CPU cores with 3 GPU devices can speed up the computation by a factor of 300 relative to the original serial implementation, and a factor of 22 relative to the 24 CPU cores parallel version. Additionally, the approach was also adapted to NEI related application involving numerous ODEs and achieved a 15-fold speedup relative to the pure MPI implementation.

There is a limitation in the current implementation. Only synchronous mode is supported in the task scheduler, that is when a task is submitted to GPU, the CPU will be blocked until the result is back from GPU. For integral tasks in spectral calculation, the waiting time only account for a very small portion of the total time, so there is no need to use asynchronous methods such as overlapping communication and computation. But when the single task is time-consuming to GPU, some asynchronous task queuing mechanism must be introduced to keep CPUs busy and reduce the waiting time.

Our ongoing work will be focused on developing an improved scheme for load balancing and enhancing the adaptability of the approach to other more complex astrophysical applications such as solving ionization equations and nucleosynthesis reactive network.

VI. Acknowledgment

This work was supported by National Natural Science Foundation of China (NSFC) through grant U1231108.

References

[1] R. D. Cowan, *The theory of atomic structure and spectra*. Univ of California Press, 1981, vol. 3.

[2] R. Smith. (2008, Aug.) Calculating radiative recombination continuum from a hot plasma. [Online]. Available: http://atomdb.org/Physics/rrc.pdf

[3] J. Houck and L. Denicola, "ISIS: An interactive spectral interpretation system for high resolution x-ray spectroscopy," in *Astronomical Data Analysis Software and Systems IX*, vol. 216, 2000, pp. 591–594.

[4] K. Arnaud, "XSPEC: The first ten years," in *Astronomical Data Analysis Software and Systems V*, vol. 101, Qubec City, Canada, Sep 1996, pp. 17–20.

[5] M. S. Noble and M. A. Nowak, "Beyond XSPEC: Toward highly configurable astrophysical analysis," *Publications of the Astronomical Society of the Pacific*, vol. 120, no. 869, pp. 821–837, 2008.

[6] M. Bautista and T. Kallman, "The XSTAR atomic database," *The Astrophysical Journal Supplement Series*, vol. 134, no. 1, p. 139, 2001.

[7] R. K. Smith, N. S. Brickhouse, D. A. Liedahl, and J. C. Raymond, "Collisional plasma models with APEC/APED: emission-line diagnostics of hydrogen-like and helium-like ions," *The Astrophysical Journal Letters*, vol. 556, no. 2, p. L91, 2001.

[8] A. Al-Omari, J. Arnold, T. Taha, and H.-B. Schuttler, "Solving large nonlinear systems of first-order ordinary differential equations with hierarchical structure using multi-gpgpus and an adaptive runge kutta ode solver," *Access, IEEE*, vol. 1, pp. 770–777, 2013.

[9] Y. Shi, W. H. Green, H.-W. Wong, and O. O. Oluwole, "Accelerating multi-dimensional combustion simulations using gpu and hybrid explicit/implicit ode integration," *Combustion and Flame*, vol. 159, no. 7, pp. 2388–2397, 2012.

[10] K. Arumugam, A. Godunov, D. Ranjan, B. Terzic, and M. Zubair, "An efficient deterministic parallel algorithm for adaptive multidimensional numerical integration on gpus," in *42nd International Conference on Parallel Processing (ICPP 2013)*, Lyon, France, Oct. 2013, pp. 486–491.

[11] ——, "A memory efficient algorithm for adaptive multidimensional integration with multiple gpus," in *20th International Conference on High Performance Computing (HiPC, 2013)*, Bengaluru, India, Dec. 2013, pp. 169–175.

[12] D. Thuerck, S. Widmer, A. Kuijper, and M. Goesele, "Efficient heuristic adaptive quadrature on gpus: Design and evaluation," in *Parallel Processing and Applied Mathematics (PPAM 2013)*, Warsaw, Poland, Sep. 2013, pp. 652–662.

[13] R. Piessens, D. Doncker-Kapenga, C. Überhuber, D. Kahaner *et al.*, "QUADPACK, a subroutine package for automatic integration," *Springer Series in Computational Mathematics*, vol. 1, p. 301p, 1983.

[14] Intel math kernel library. [Online]. Available: http://software.intel.com/en-us/intel-mkl

[15] T. Hahn, "CUBA a library for multidimensional numerical integration," *Computer Physics Communications*, vol. 168, no. 2, pp. 78–95, 2005.

[16] B. Gough, *GNU scientific library reference manual*. Network Theory Ltd., 2009.

[17] E. Yurtesen, M. Ropo, M. Aspnas, and J. Westerholm, "SSE vectorized and gpu implementations of arakawa's formula for numerical integration of equations of fluid motion," in *19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP, 2011)*, Ayia Napa, Cyprus, Feb. 2011, pp. 341–348.

[18] S. Chen, "A hybrid gpu/cpu FFT library for large FFT problems," Master's thesis, University of Delaware, Newark, USA, 2013. [Online]. Available: http://udspace.udel.edu/handle/19716/12800

[19] G. Laccetti, M. Lapegna, V. Mele, and D. Romano, "A study on adaptive algorithms for numerical quadrature on heterogeneous gpu and multicore based systems," in *Parallel Processing and Applied Mathematics*, Warsaw, Poland, Sep. 2013, pp. 704–713.

[20] Z. Wang, L. Zheng, Q. Chen, and M. Guo, "CPU + GPU scheduling with asymptotic profiling," *Parallel Computing*, vol. 40, no. 2, pp. 107–115, 2014.

[21] A. Acosta, V. Blanco, and F. Almeida, "Dynamic load balancing on heterogeneous multi-gpu systems," *Computers & Electrical Engineering*, vol. 39, no. 8, pp. 2591–2602, 2013.

[22] B. Yang, K. Lu, J. Liu, X. Wang, and C. Gong, "Hybrid embarrassingly parallel algorithm for heterogeneous cpu/gpu clusters," in *7th International Conference on Computing and Convergence Technology (ICCCT 2012)*, Seoul, Korea, Dec. 2012, pp. 373–378.

[23] NVIDIA. (2013) Sharing a gpu between mpi processes: multi-process service (mps) overview. [Online]. Available: http://docs.nvidia.com/deploy/mps/index.html

[24] M. S. Noble, L. Ji, A. Young, and J. Lee, "Parallelizing the xstar photoionization code," in *Astronomical Data Analysis Software and Systems XVIII*, Qubec City, Canada, Sep 2009, pp. 301–304.

[25] W. H. Press, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge, UK: Cambridge university press, 2007.

[26] J. Xiao, X. Xu, J. Sun, X. Zhou, and L. Ji, "Acceleration of solving non-equilibrium ionization via tracer particles and mapreduce on eulerian mesh," in *International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2014)*, Dalian, China, Aug. 2014, pp. 29–42.

[27] L. Petzold and A. Hindmarsh, "LSODA–livermore solver for ordinary differential equations, with automatic method switching for stiff and non-stiff problems," *Computing and Mathematics Research Division, I-316 Lawrence Livermore National Laboratory*, vol. 94550, 1997.