Terr. Atmos. Ocean. Sci., Vol. 27, No. 4, 577-592, August 2016 doi: 10.3319/TAO.2016.03.10.01(ISRS)

SAR Image Simulations Using the LBM Algorithm on MPI-GPU

Chuan-Li Sun[1, *], Lung-Chih Tsai[1, 2], and Cheng-Yen Chiang[2, 3]

[1] *Institute of Space science, National Central University, Taoyuan City, Taiwan, R.O.C.*
[2] *Center of Space and Remote Sensing Research, National Central University, Taoyuan City, Taiwan, R.O.C.*
[3] *Department of Computer Science & Information Engineering, National Central University, Taoyuan City, Taiwan, R.O.C.*

Received 1 June 2015, revised 3 March 2016, accepted 10 March 2016

ABSTRACT

Synthetic Aperture Radar (SAR) is a powerful tool for studying natural environments under all-weather and day-and-night conditions. SAR system design and data-processing algorithm simulation is noted for its controllable parameters. The satellite SAR echo signal simulation framework has been successfully applied to target recognition based on Radarsat-2 and TerraSAR-X images and in strip map mode. However, such SAR image simulation works only on CPU or GPU (graphics processing units) and requires huge calculations. We developed a "Load-Balancing Model (LBM)" algorithm that uses Message Passing Interface GPU (MPI-GPU) to reduce the inner loop load and improve the computational performance. The LBM algorithm uses MPI-GPU technology to build the simple GPU cluster system. The LBM algorithm is used to separate the intensive computing and controlling tasks for each node, and exploit the contemporary GPU computation capability to accelerate the computing tasks. We conducted a relevant experiment on a target radar cross section (RCS) and improved the performance by a factor of > 40 compared to a 4-core CPU accelerated program.

Key words: SAR, MPI, GPU, Load-balancing model

Citation: Sun, C. L., L. C. Tsai, and C. Y. Chiang, 2016: SAR image simulations using the LBM algorithm on MPI-GPU. Terr. Atmos. Ocean. Sci., 27, 577-592, doi: 10.3319/TAO.2016.03.10.01(ISRS)

1. INTRODUCTION

Synthetic Aperture Radar (SAR) image databases have an important role in target recognition and identification system development (Lee 1980; Rihaczek and Hershkowitz 2000; Margarit et al. 2006; Kasim et al. 2008; Lee and Pottier 2009; Guo et al. 2010; Huang and Lee 2010; Tian et al. 2011; Wang et al. 2011). It is mainly dependent on the operator's prior knowledge. The operating load, experiments and target types are the uncertainties. Hence, SAR simulation is one potential alternative to alleviate this problem. Developing a full blown SAR image simulation scheme with high verisimilitude including the sensor and target geo-location relative to the Earth, SAR sensor movement, SAR system parameters, target radiometric and geometric characteristics, and environment clutter is highly desirable.

The SAR image simulation algorithm includes three loops. The first loop is the sensor movement. The second loop is the calculation for each polygon on the target. The third loop is the echo signal integration for each slant range element. The complexity is obviously $O(n^3)$. Because the simulation run time drastically increases corresponding to the data resolution, it is necessary to find approaches to reduce the simulation time.

Most desktop computers today are equipped with fully programmable graphics processing units. These chips contain many powerful Single Instruction Multiple Data (SIMD) processors that can support parallel data processing and high-precision computation (Bai et al. 2009)—a practice known as general-purpose computing on graphics processing units (GPGPU). Newer general-purpose interfaces include NVIDIA's Compute Unified Device Architecture (CUDA) and the new multivendor standard OpenCL. CUDA is gaining position as the choice for the high performance computing (Vasiliadis et al. 2008) community. There is a growing amount of work being carried out on high performance computing around the world (Harish and Narayanan 2007). Message Passing Interface (MPI) has been the choice for high performance computing for more than a decade and it has proven its capability in delivering higher performance

* *Corresponding author*
E-mail: 956403009@cc.ncu.edu.tw

in parallel applications. CUDA and MPI use different programming approaches but both of them depend on the inherent parallelism of the application to be effective. CUDA runs on the GPU and the GPU is a magnitude order faster than the common CPU. However, the GPU performance depends on which application is executed by the GPU.

MPI typically runs on CPU clusters so that it does not have the hardware level performance acceleration support that CUDA has. However, using MPI, we can execute different components from different programs in different CPUs in the cluster, whereas we can only run one kernel at a time inside the GPU while we are using CUDA.

We propose the Load-Balancing Model (LBM) algorithm that uses hybrid programming technologies with MPI and CUDA to build a parallel computing environment to reduce the run time and improve parallel program performance, the LBM algorithm can reduce the inner loop load and solve the dynamic balancing problem in homogeneous computing.

In the LBM algorithm is divided into MPI strategy mechanism and GPU strategy mechanism. In the MPI strategy mechanism is mainly a combination of static and dynamic loads, with three balanced stage parts, including a pre-evaluation stage, management-work stage, and adjust-strategy stage. In the GPU strategy mechanism, a task-parallel model strategy, where CUDA is used to build a parallel computing environment on a single node and data can be moved directly from one GPU to another using Peer to Peer (P2P) communication that by-passes the main memory, which includes feedback and calculation stages.

The rest of this paper is organized as follows. In section 2, we introduce the CUDA, MPI, and SAR image simulation algorithm framework. In section 3, the LBM algorithm method on MPI-GPU clusters and the LBM algorithm approach are described. Our system configuration is specified in section 4. The evaluation results for our work on a test bed are presented. Concluding remarks and future work are given in section 5.

2. BACKGROUND

2.1 CUDA and MPI

The CUDA integrated technology introduced by the NVIDIA Company is the official name for this kind of GPU product. In CUDA hardware the threads are grouped into 32-thread units called *warps*. Each warp works on a task, which we call a *worker*. The threads are organized through a "Thread Block" (CUDA, http://en.wikipedia.org/wiki/CUDA). A Thread Block can have up to 512 threads. Threads that belong to the same Thread Block can share data through a shared memory. These threads execute intensive computing tasks in SIMD. A CUDA program consists of one or more portions that are executed on either the host or the GPU (CUDA C Programming Guide, http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz4EBzi6mCo). GPU nodes are connected as peripheral devices on the I/O bus (PCI express). Data can be moved directly from one GPU to another using P2P communication that by-passes the main memory. However, communication between GPU buffers used by different processes must go through the main memory. GPU task-parallel models have recently become a popular topic in GPU research. Aila and Laine (2009) presented the idea of persistent threads for handling irregular ray generation in ray tracing and this was further developed in a GPU ray tracer OptiX (Parker et al. 2010).

The stated goals of MPI are high performance, scalability and portability. Achieving low latency and high throughput is important for computing clusters, since a lack of shared memory implies large amounts of network data transfer. In the MPI model data must be explicitly transmitted between processors using *Send*() and *Recv*() primitives. MPI can cooperate with Fortran, C, C++, and other languages for developers' selection (Zheng 2002, http://moodle.ncku.edu.tw/pluginfile.php/685069/mod_resource/content/0/mpic_2002.pdf). At present, the most common MPI version in the academic field is S-MPI, MPICH, OpenMPI, IBM MPL, CHIMP, and LAM (Local Area Multicomputer) which all made based on MPI standards. Many literatures related to MPI are published, such as performance realization of Mixed Model MPI, Noaje mixed researches of CUDA and MPI Speeding up matrix multiplication and conjugate gradient (Noaje et al. 2010) and so on. Henty (2000) compares the performance achieved by the hybrid model with that achieved by a pure MPI. Chang et al. (2009) developed an algorithm to compute pair wise Pearson correlation coefficients on a single GPU using CUDA. The hybrid CUDA and MPI programming has also been studied by Noaje et al. (2010). Yang et al. (2010) discussed the performance gain via hybrid CUDA and MPI programming through several applications including matrix multiplication, MD5 and Bubble sort on GPU clusters. Though the hybrid CUDA and MPI demonstrates good combination the load balancing problem among GPUs has not been addressed in the aforementioned works.

Considering that the OpenMPI is equipped with strengths such as easy transplant and cross-platform (Karunadasa and Ranasinghe 2009), the paper adopts OpenMPI, CUDA and cluster-based computers to serve as the testing environment of parallel computing of echo signals of simulation radar images. Furthermore, for OpenMPI is one of suits of Linux and is used for testing efficiency by the high-efficiency evaluation website TOP 500 (OpenMPI, Open Source High Performance Computing, http://www.open-mpi.org/), it is suitable for testing efficiency of various supercomputers or high-efficiency operation cluster-based system.

OpenMPI uses the mpirun instruction to execute programs. Parameters of mpirun instruction are divided into two

parts, one is parameter options of mpirun (mpirun-options), and the other one is the name of program. The complete command is mpirun (mpirun-options) (program name). Users can set parameter options of mpirun according to their own demands, which can be referred to Table 1 for common settings.

2.2 Satellite SAR Echo Signal Simulation Framework

We announced an algorithm to simulate a complex SAR echo signal reference in (Loper and Parr 2007). It is suitable for use in establishing a procedure for calculating the altitude of all targets at the same time. The algorithm's framework can be seen in Fig. 1. The simulation processing flow is basically adapted from Refs. (Curlander and McDonough 1991; Cumming and Wong 2005; Wang et al. 2011). We divided this flow chart into three partitions. The first partition is the data collection including parameter setting (label 1A) and database (label 1B). The next partition is preprocessing including scene location calculation, object dimension estimation, SAR echo signal raw data dimension estimation and fast time series generation. The final stage is the main calculation consisting of three loops, sensor/azimuth position loop, polygon loop, and slant range loop.

Table 1. Parameter List of mpirun Instruction.

Parameter Options	Illustrations
--app	Notifying file names of app files. Other parameters will be ignored with the presence of this parameter.
-bind-to-board	Binding the process to a motherboard and being executed by CPU of the motherboard (this setting is applicable for the case that there are more than one motherboard on single node).
-bind-to-core	Binding the process to a core and being executed by the core.
-bind-to-none	Do not bind the process (default).
-byboard	Setting that the process performs looping execution by taking a motherboard as a unit (the setting is applicable for the case that there are more than one motherboard on single node).
-bycore	Setting that the process performs looping execution by taking a core as a unit.
-bynode	Setting that the process performs looping execution by taking a node as a unit.
-c-n-np	The quantity of used processes when executing a program.
-cpus-per-proc	Setting the quantity of CPU when dealing with single process.
-h--help	Displaying illustrations of instructions.
-H--host	Using which operational resources of nodes when setting processes.
-machinefile	Notifying file names of profiles of a system.
-path	Setting positions of program execution.

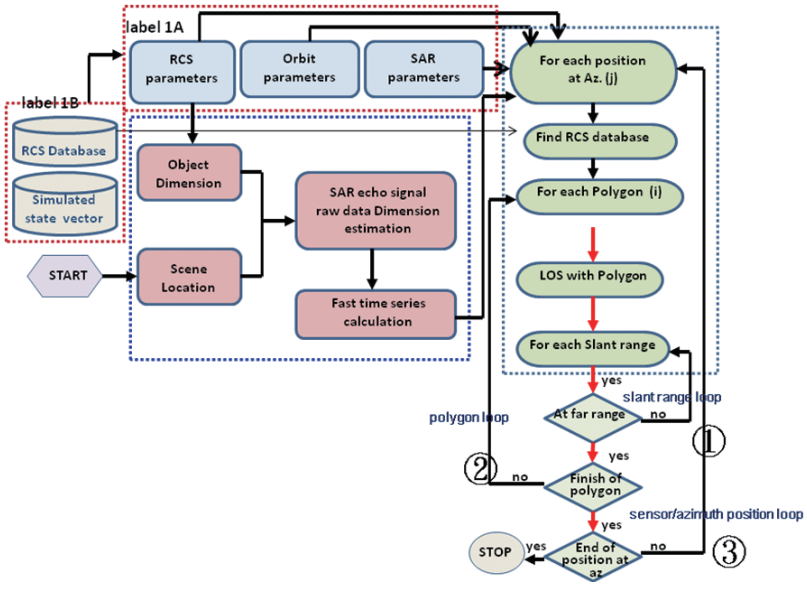


Fig. 1. SAR echo signal simulation flowcharts. (Color online only)

There are two divisions, parameter setting and database collection. Parameter setting is separated into three sections, the radar cross section (RCS), Orbit, and SAR system. The important SAR system parameters including satellite name, look angle range in degree, transmitted frequency in Hz, pulse repeat frequency (PRF) in Hz, ADC sampling rate in Hz, duty cycle, effective antenna dimensions in meters (they can be replaced with beam width for elevation and azimuth direction in degrees), chirp bandwidth in Hz and squint angle in degrees.

The satellite orbit state vector in the orbit database including position, velocity and time can be simulated from two-line elements or collected from real historical satellite header files. The RCS database simulates from 3D CAD models using physical-optic (PO) / electro-optic (EO) methods. There are differences in the incidence and aspect angles at discrete degree intervals between the satellite and target object.

The main kernel can be divided into three parts based on the loop type. The first loop is the azimuth sampling. The satellite position derived from the state vector at the azimuth direction is the input in this stage.

In the polygon loop some values are calculated sequentially including the instantaneous slant range, angle between light-of-sight and polygon central location, synthetic antenna pattern at the azimuth direction and the RCS database file search. This loop needs to calculate for each polygon on the target's surface.

The final loop, slant range loop, is the SAR echo signal equation and integration for each polygon and each slant range line.

3. HYBRID MPI/CUDA PROGRAM WITH LBM

In Fig. 2, The time complexity in calculation kernel is $O(n^3)$ originally because the three loops. According to the number of calculations in this procedure for SAR target images, 2520 altitudes exist under incidence angle sampling from 20 - 50 degrees with 5 degree and aspect angle sampling from -180 - 180 degrees with 1 degree. The polygon numbers for each target model can reach orders of magnitude of more than 20000. From the test results using 25600 polygons, 2200 azimuth position samples, 7500 slant range samples under Intel 2.33 GHz (Quad cores), 8 Gbytes RAM and 64 bits Linux 2.6.18 kernel environment, requires almost 10.3 months to calculate with one thread for all altitudes.

We developed an efficient LBM algorithm which uses MPI+GPU-based cluster structure model, and combining the advantages of both static and dynamic load balance in the LBM (Willebeek-LeMair and Reeves 1993; Hui and Chanson 1999; Arora et al. 2001; Tzeng et al. 2010; Cederman and Tsigas 2012). Therefore, we refer to (Colajanni et al. 1998; Pai et al. 1998; Srisuresh and Gan 1998; Bunt et al. 1999; Cardellini et al. 2002; Padhy and Rao 2011) research papers, and design LBM operating mechanism based on Queuing Theory, which is divided into MPI strategy mechanism and GPU strategy mechanism. The MPI strategy mechanism can handle the information collection, performance evaluation and work distribution adjustment. Besides, GPU strategy mechanism mainly applies CUDA procedure to call the kernel function, to execute by GPU, for the convenience of applying large amount of threads in GPU for parallel processing work. So the LBM operating mechanism can make each computer of the resource used effectively, and reduce the working time, and improve each computer working of load balance by simulation radar image echo signal, as shown in Fig. 3 of the LBM overall operation schematic diagram.

3.1 In the MPI Strategy Mechanism

MPI provides abundant functions for message passing and related operations. MPI adopts the signal process signal data program model, in other words, each process carries out the same MPI program. The MPI program can obtain a number of current programs, with each program differing from the other programs by the number. Each program can carry out different tasks while communicating with other processes (Song and Dongarra 2012; Shi et al. 2013). In the MPI Strategy Mechanism divided into pre-evaluation stage, management-work stage and adjust-strategy stage are discussed separately as follows.

3.1.1 Pre-Evaluation Stage

In worker node network, we have established interconnect communication between master node and other worker nodes, each node's back-pass CPU queue length and GPU memory, thread and cores, etc. As CPU load is most influenced by queue length (Medhi 1991), we've thereby only classified CPU's queue length as its operational performance reference indicator. Besides, to reduce message passing and waiting time during work assignment, in this stage we've also combined azimuth angle, side look angle, and polygon indicator into Matrix Pool area, as shown in Fig. 4.

As shown in Fig. 5, the operation schematic diagram of result back-pass master node after each worker node finished calculating 1000 polygons. This is to facilitate cutting quantity for each worker node's computation, and attach each worker node computational task with independence, thereby to avoid increasing back-pass reaction time due to busy work, which would be enormously helpful for enhancing LBM overall working efficacy.

3.1.2 Management-Work Stage

After the aforementioned stage finished, the system starts to execute computational task, which is divided into "work assignment" and "performance assessment", as follows:

(1) **Work Assignment**: master node receives each node's (including itself) back-pass assessment result, and coverts to performance indicator based on equation for work assignment according to performance indicator size. Performance indicator indicates each work node's computing ability, and also represents the maximum computing quantity of each work node before computation, so as to ensure each node is with same work load when startup. The result is each node's assignment work quantity ratio, as shown in below:

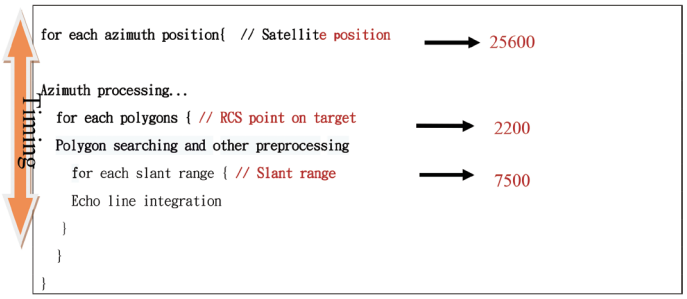


Fig. 2. The original time complexity for the echo signal simulation. (Color online only)

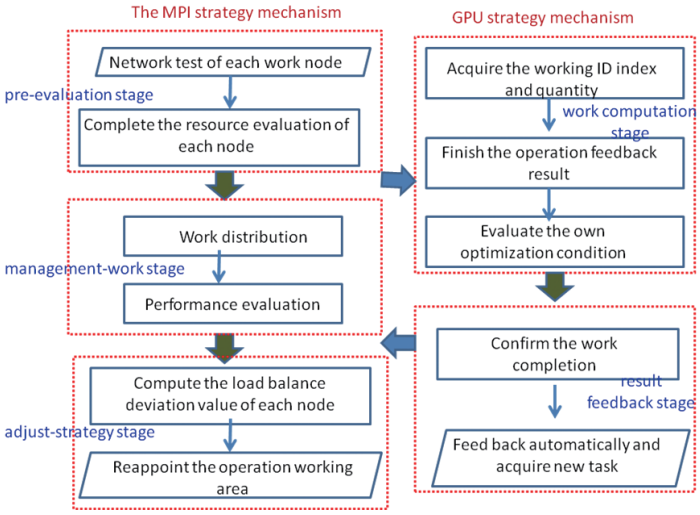


Fig. 3. The overall operation schematic diagram of LBM. (Color online only)

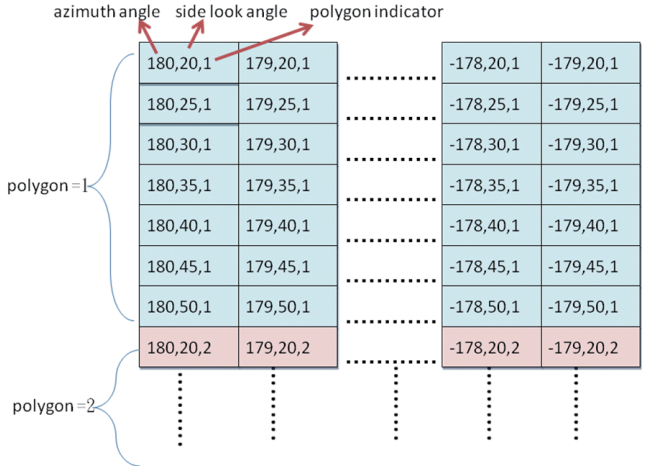


Fig. 4. Matrix pool area combined by azimuth angle, side look angle, and polygon. (Color online only)

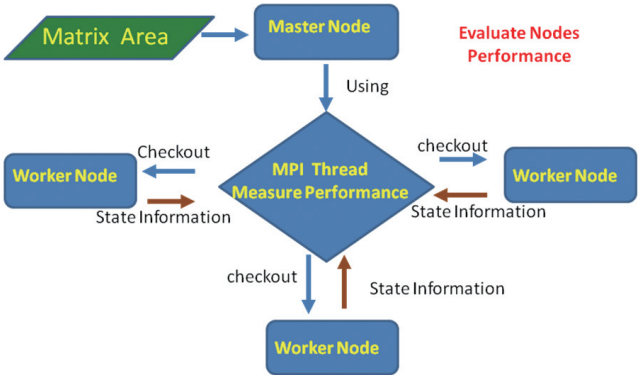


Fig. 5. Operation schematic diagram of pre-evaluation stage. (Color online only)

$$A_{T(\text{master node})} = \text{processing time} / 1000 \text{ polygons} \quad (1)$$

$$B_{T(\text{worker node})} = \text{processing time} / 1000 \text{ polygons} \quad (2)$$

$$R_{AB} = B_T / A_T \quad (3)$$

Now we regard 1000 polygons as the testing condition for each work node's performance indicator, the result is such node's assignment work quantity ratio. R_{AB} is a such simple average processing time ratio, each worker node divided by master node's specific value, can be approximately work out each work node's initial performance indicator, to decide each node's suitable computational polygon quantity (azimuth angle and horizontal angle) region range, and equalize each work node's average load when startup.

(2) **Performance Assessment**: In actual operating process, due to various node resources (such as memory size, main board efficacy, etc.), there will exist inconsistent information. So, after referred (Shi and Tang 1999; Liu and Shi 2007), we are able to design a load trigger condition that is suitable for this paper's environment, so as to facilitate master node's (MPI node) command for work node operating status, and avoid scheduling congestion and other worker nodes' idle conditions, thereby to achieve the best effect, with trigger condition equation as below:

$$m = \frac{\sum_{i=1}^{N} n_i}{N}, \quad \sigma = \sqrt{\frac{\sum_{i=1}^{N} (n_i - m)^2}{N}} \quad (4)$$

In Eq. (4), m is the overall work group network's computational average time value, σ is variance, N is refers to worker node number, n_i represents the time of finishing K times of polygon (K is multiple of 1000) on the No. i node.

Therefore, m indicates a required average time of each work node's K times of polygon computation, while variation degree refers to the size between each node's operation time and average operation time difference. As such node's variation degree increasing, this node is perhaps with busy or idle computational amount. Figure 6 is the schematic diagram of management-work Stage's "performance assessment", of which, tolerance (threshold) is a result after compared each node's average back-pass time value with variation degree according to Eq. (4), and the threshold value is σ / m.

3.1.3 Adjust-Strategy Stage

After each node finished work computation, and when passing result back to master node, master node will compare each node's average back-pass time value with variation degree, and figure out the tolerance value, which is established according to experimental environment's assessment results. As a result, unbalanced load will occur in this group when exceeding such range value, which also refers to other nodes are in idle. Therefore, MPI will then readjust work assignment, and launch "central decision" mechanism to execute work assignment, to assign unfinished work regions to worker nodes that not exceeding tolerance value, as shown in Fig. 7, the schematic diagram of adjust-strategy stage.

3.2 GPU Strategy Mechanism

We've utilized CUDA to construct GPU worker node's strategy mechanism, and applied GPU multi-task parallel feature to jointly execute computation by using several threads (Kunz 1991; Willebeek-LeMair and Reeves 1993; Wang et al. 2008). Besides, through employing share memory communication, we are able to execute SIMD intensive computer task. GPU strategy mechanism in this part can be mainly divided into "work computation stage" and "result feedback stage", as follows.

3.2.1 Work Computation Stage

From Fig. 5, matrix computational region is managed

by master node (MPI node), each worker node will upload corresponding input value (range) to GPU for computing. Furthermore, by referring (Guim et al. 2010) papers, we divided computational task into several subtasks, and allocate them separately to CPU or GPU for processing, together with resource allocation modes in (OSCAR, Observing Systems Capability Analysis and Review Tool, http://www.wmo-sat.info/oscar/satellites) papers related CPU/GPU systems, bandwidth conditions and power consumption, etc., to reduce power consumption and increase available resources. Optimized work load condition during GPU computational process is established as Eq. (5).

$$T_{GPU} = \frac{L\alpha}{P_{GPU}},\ T_{CPU} = \frac{L(1-\alpha)}{mP_{CPU}},\ T = \max(T_{CPU}, T_{GPU}) \quad (5)$$

Of which, α is load rate, L is total load quantity, m is CPU executing cores, and P represents GPU or CPU's thread size. Therefore, such node's reasonable load time should be the maximum computational value between CPU and GPU, and when two times are roughly the same, it indicates that current load balance is the best status. Due to this paper's utilized experimental load rate is 0.6, and we can compute CPU and GPU's reasonable time according to assigned work quantity.

3.2.2 Result Feedback Stage

After GPU node finished all assigned tasks, it will pass result back to master node, and start new computational task. Besides, when there is no other computational task, it will then activate its feedback mechanism and notify master node to conduct work assignment requisition. As a result, in the MPI Strategy Mechanism, not only MPI can actively inspect each node's work condition, but each node is also able to feed master node's current condition back during GPU strategy mechanism, to compensate the blank of master node due to busy work. Figure 8 refers to the schematic diagram of MPI and GPU's work reassignments under GPU strategy mechanism.

4. EXPERIMENT EVALUATION

4.1 Load Balance Trigger Condition

To establish load balance trigger condition, we take multi-node and single GPU card environment as the testing benchmark, and conduct more than 10 times of computational tests, to decide its average back-pass time value. Testing results are respectively shown in Tables 2 and 3 and we thereby to know GTX660 and C2050 GPU cards' collected computational resources under different polygons computation. In Table 2 the GTX660 GPU card, which polygon number exceeding over 4000, and in Table 3 the C2050 GPU card, which polygon number exceeding over 8000, we adjust and increase both GPUs thread to one time. We found their average memory access time and computing time are increased to more than one time, there were to indicate the GPU's loading is busy.

4.2 Single-Node and Multi-GPU Work Environment

To test LBM load's equilibrium mechanism efficacy under single-node and multi-GPU, and after considered computer's main board space and slot size, we select two pieces of NVIDIA GeForce 470 video cards as the testing objects, and relevant NVIDIA GeForce 470 GPU specifications are shown in Table 4.

We take the polygon size, which is 26500 generated from common CAD chart file as the benchmark, to simulate it is within satellite synthetic aperture radar flight direction's squint angle range, and make statistic on the computing time required by 1 - 30000 polygons of SAR echo signal simulated, thereby assessing single-node MPI-2GPU's actual computational efficacy, whose average operation time is shown in Fig. 9.

4.3 Multi-Node and Single GPU's Work Environment

Under multiple worker nodes network environment, we utilize two pieces of NVIDIA GTX660 card and two Tesla C2050 cards to actually create high-performance computing cluster system through D-LINK DGS-3100-24 switcher to connect each work node, with relevant specifications are shown in Table 5.

Due to each work node only has one GPU card, we also adjust CUDA's thread number and increase GPU computational amount, and one polygon is only finished in 0.07 sec as indicated in Fig. 10, 32 polygons are finished in 2.339 sec, with average back-pass operation time being 0.073 sec.

Under LBM modal multi-node MPI-GPU environment, we input the same computational condition to it is within satellite synthetic aperture radar flight direction's squint angle range, and make statistic on the computing time required by 1 - 30000 polygons of SAR echo signal simulated from as shown in Fig. 11. It can thus be seen that the more GPU quantity is, load equilibrium effect is better to save overall operation time.

In Fig. 11, operation time is each node's average value, which is neither directly related with GPU card's sequence, nor indicates that No. 4 piece's efficacy is higher than that of the previous three. The key to upgrade computational efficacy is maintain the overall load balance within tolerance value range. Base on the SAR echo signal simulation framework, Table 6 shows it is need almost 10.3 months calculated under Intel 2.33 GHz (Quad cores) reduce to 1.06 days calculated under LBM-4GPU with one thread for all altitudes.

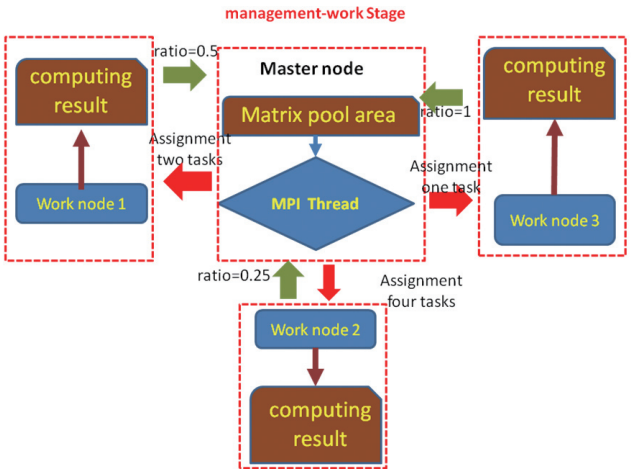


Fig. 6. Schematic diagram of management-work stage. (Color online only)

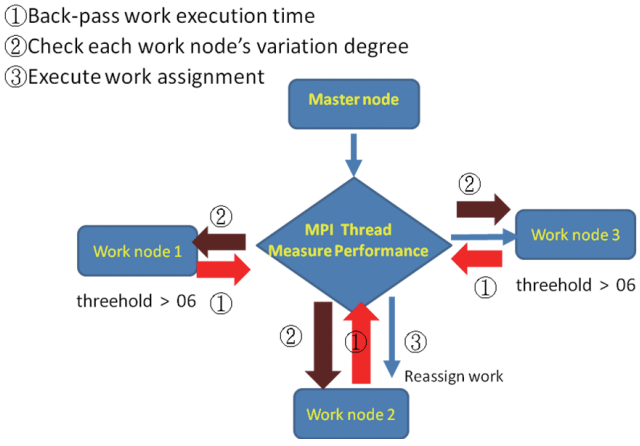


Fig. 7. Schematic diagram of adjust-strategy stage. (Color online only)

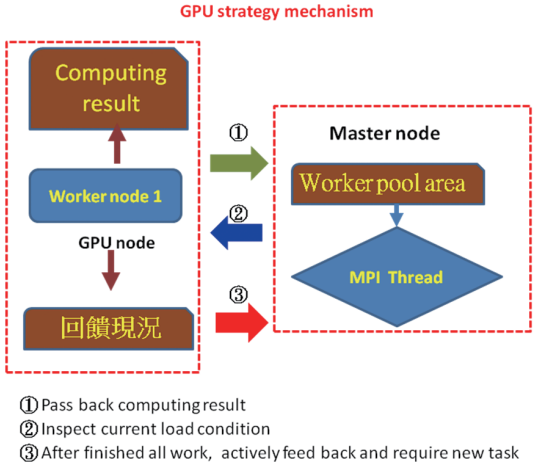


Fig. 8. Schematic diagram of GPU strategy mechanism. (Color online only)

Table 2. GTX660 computational task testing situation. (Color online only)

Polygon number / 資源條件	1	1000	2000	4000	8000	16000	20000	25000	30000	35000
最高CPU使用率 (%)	1	1	25	80	96	99	99	99	99	99
調整GPU執行緒 (大小)	16	16	16	32	32	32	32	32	32	32
平均計算時間 (sec)	0.0128	16.6	37.86	72.86	166.15	428.97	521.42	820.67	1023.58	1436.25
平均記憶體存取時間 (sec)	0.044	60	166	552	1423	3675	7253	9594	13634.5	21262

Table 3. C2050 computational task testing situation. (Color online only)

Polygon number / 資源條件	1	1000	2000	4000	8000	16000	20000	25000	30000	35000
最高CPU使用率 (%)	1	1	25	30	85	96	99	99	99	99
調整GPU執行緒 (大小)	16	16	16	16	32	32	32	32	32	32
平均計算時間 (sec)	0.0122	15.21	30.89	61.25	138.21	398.12	488.26	756.89	865.32	1058.98
平均記憶體存取時間 (sec)	0.028	56.6	166.5	462.99	951.55	3210.2	5624.58	7412.56	9351.03	18306.47

Table 4. NVIDIA GeForce 470 GPU specification.

Specifications	GeForce 470
Size	9.5-inch PCIe × 16 slot
CUDA core	448
CUDA core frequency	0.607 GHz
Total memory capacity	1.6 GB GDDR5
Memory speed	1.28 GHz
Memory port	320 bit
Memory bandwidth	133.9 GB sec^{-1}
Displayer support	Double-linked DVI-I × 1
Displayer's minimum resolution ratio	2560 × 1600

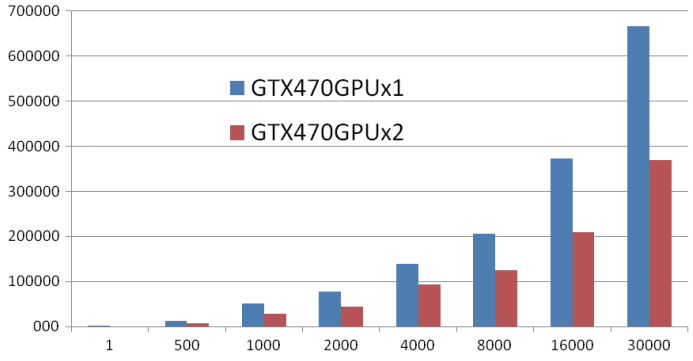


Fig. 9. Single-node MPI-2GPU's operation time. (Color online only)

Table 5. LBM model multi-node MPI-GPU's simulated environment.

	Master node	Worker node	
		Node 2	Node 3/Node 4
Processors	Intel Xeon/NVIDIA Tesla C2050 × 1	NVIDIA Tesla C2050 × 1	GeForce GTX660 × 1
Clock frequency	3.33 GHz	1.15 GHz	980 MHz
Cores per processor	4	448	960
Memory size	16 GB	3 GB	2 GB
GFLOPs/sec		515	486
OS	nUbuntu Linux 12.0.4		

```
run MPI-4GPU版本(TeslaC2050 x 2,GTX660 x2 , CUDA SDK 5.0):

#define DEVNUM 4            // 在 IDL2C++16_RH_cuda3.cu 檔案標頭指定所要
#define TESTRUN 32          // 採用 GPU處理個數及批次處理的 scan line 數目

cd /home/ascychiang/V3_SAR_Teslax4\bin\linux\release
sh run.bat
(./IDL2C++16_RH_cuda3 ./data/TerraSAR-X.par ./data/TER_SV_TW.txt ./data/ ./out/ test HH 30 0 1 0 500 500 0)

Total number of selected RCS=9238,     Polygon=25672,
Total effective number of polygon=1
grid.x= 482    grid.y= 70    grid.z= 1;     num_rg=7709        num_az=2222

Result of the center pixel:     ss0[num_az/2*num_rg + num_rg/2] =
                                (0.73128172859683049367,-1.21046562669925972244)

run 32 scan lines 時間 Escaped: 2.33999999999999985789  sec
平均每條 scan line 的處理時間約為 0.073125 sec
```

Fig. 10. Average time of MPI-GPU computing one polygon.

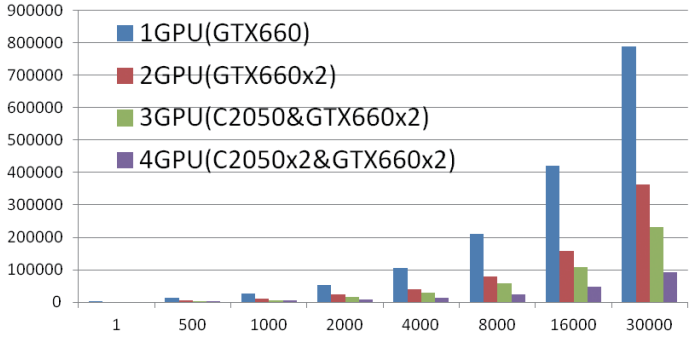


Fig. 11. MPI-1GPU~4GPU operation time. (Color online only)

Table 6. Each node process time result for SAR echo signal simulation (day).

Polygons = 26500	4 Core	MPI-1GPU	MPI-2GPU	MPI-3GPU	MPI-4GPU
Time	309.7	7.7	4.19	2.68	1.06
Speedup	1	40.22	73.93	115.56	292.17

4.4 The Result of SAR Echo Signal Simulation for MD80

As described, the LBM method can strengthen the SAR database establishment at all target's altitudes. The simulated SAR images can be obtained at different sequential incidence angles and aspect angles. In this study, we applied the LBM method in SAR echo signal framework to simulate *MCDONNELL DOUGLAS MD-80* image target and compared the results with real TerraSAR-X Satellite

and Radarsat-2 Satellite image. The simulated SAR echo signal is certificated by the developed SAR processor evaluated using real satellite SAR echo signal data. The main processing system produced a modelled *MD-80* 3D CAD image containing numerous grids or polygons with computed RCS as a function of the incident and aspect angles used for a given set of radar parameters. The number of polygons can be determined by the target's geometry complexity and its electromagnetic size. To realize the imaging scenario, each polygon must be properly oriented and positioned based on Earth Centered Rotating (ECR) coordinates.

Table 7 shows the parameter values of TerraSAR-X and Radarsat-2 satellites (Hsu et al. 2008) for the SAR echo signal simulation program. Figures 12a and b show simulated SAR images of the *MCDONNELL DOUGLAS MD-80* target model, in which the polygon numbers can reach orders of magnitude greater than 20000. For these simulations the incidence angle was fixed at 35 (35) degrees and the aspect angles were from 2 - 56 (0 - 54) degrees at intervals of 2 degrees for TerraSAR-X (Radarsat-2) satellite images. Each image contains the amplitude and phase information and some attributions such as corner longitude/latitude location, time, Doppler parameters, and other values processed by the processor. To produce simulated Radarsat-2 and TerraSAR-X images for *MCDONNELL DOUGLAS MD-80* aircraft target we use the SAR processing software, Multi-Sensor Processor (MSP) (Bailey and Werdell 2006), which was verified by the general satellite data (Bailey and Werdell 2006). Both the simulated Radarsat-2 and TerraSAR-X images are used to compare and evaluate the results using the LBM method.

The comparison principle for the real radar images and simulation image for the selected target object about the approximation point (RCS area smaller than the size of a real radar image resolution), is evaluated based on the real point target and 3 dB width simulation image. Because the real image is projected onto the ground, the slant range resolution direction is changed. Therefore, the analysis is based only on the flight direction. Figure 13 shows the *MD-80* CAD model and real SAR image and simulated SAR image targets where the incidence angle is fixed at 30 degrees and the aspect angles are from -8^0 to -12^0 degrees at intervals of 1 degree and the aspect angles are from -162^0 to -166^0 degrees at intervals of 1 degree. Because the RCS target is quite difficult to estimate and normally determined by measurement, it depends on the airplane's physical geometry and exterior features, the direction of the illuminating radar, the radar transmitter frequency, and the aircraft material types etc.

Based on real TerraSAR-X satellite data, we obtained the image center coordinates (119.395060, 24.894620) and compared it to the simulated image center coordinates (119.395090, 24.894470). The difference is in 1.5×10^{-4} degress or less accuracy, i.e., about 0.06 m on the ground and about 16.97 m on the satellite flight direction. The location results are also shown in Table 8. The proposed working flows and algorithms were validated by evaluating the image quality including geometric and radiometric accuracy using simple point targets first, followed by simulating *MD-80* aircraft. According to the TerraSAR-X image products, the flight direction resolution is 4.51 m, with the simulation results in 3 dB width at 3.5 m, with about 1 m difference. The product of the general situation will be similar to the pixel interpolation (for the sake of pixel Founder), so differences will exist. TerraSAR-X target point of the analysis results shown in Fig. 14, simulate TerraSAR-X point target image of the analysis results shown in Fig. 15, and Fig. 16 shows the position accuracy is almost 10^{-7} degrees for both simulate and real TerraSAR-X point target image of error point ration. The 3 dB width error evaluated from PSLR/ISLR index in the free clutter comparing to the real satellite product is less than 10%.

Table 7. TerraSAR-X and Radarsat-2 Satellite parameters.

Characteristics	TerraSAR-X Satellite	Radarsat-2 Satellite
Resolution	StripMap: 3 m	5 - 50 m
Band	Active X band Microwave	Active C band Microwave
Nominal swath Width	StripMap: 30 km × 50 km	500 km
Look angle (deg)	33.8 (beam center)	30.78
Squint angle (deg)	0	0
Transmitted freq. (Hz)	9.63 G	5.405 G
PRF (Hz)	3798	1000 - 3800
ADC Sampling rate (Hz)	165 M	31.67 M
Antenna size (m)	3.37 m^2 (0.704 m × 4.784 m)	9.43 m^2 (1.37 m × 6.88 m)
Chirp bandwidth (Hz)	150 M	100 M
Transmit duty cycle (%)	16.5	28

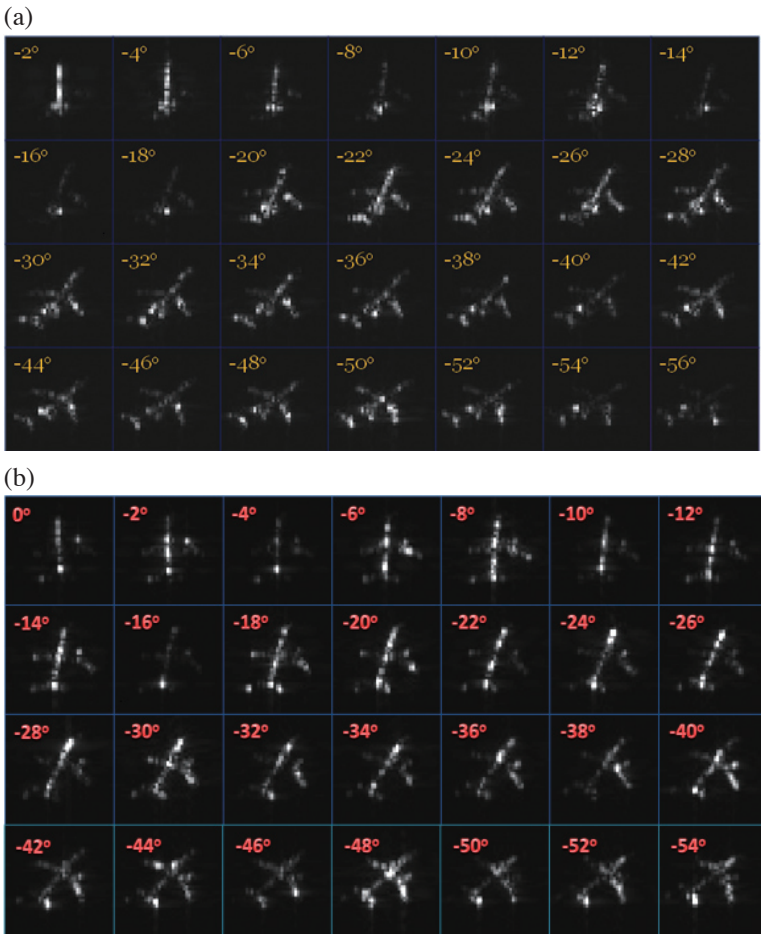


Fig. 12. (a) SAR image simulation results in TerraSAR-X Satellite parameters. (b) SAR image simulation results in Radarsat-2 Satellite parameters. (Color online only)

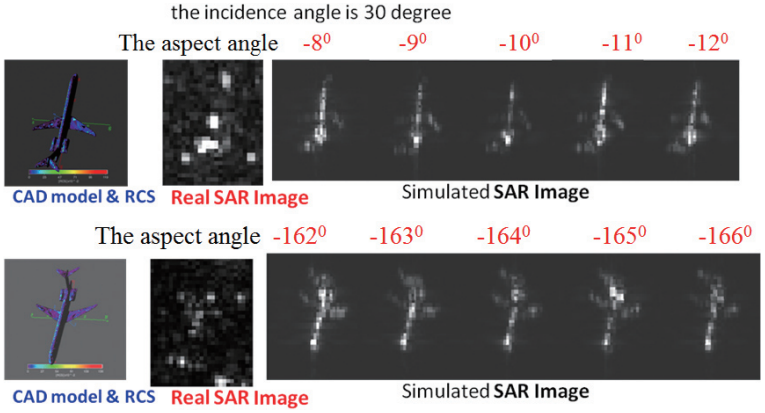


Fig. 13. Real and simulated SAR image combine with CAD model image results based on LBM method. (Color online only)

Table 8. Target point location verification results for SAR image.

Index		Real TerraSAR-X image	Simulate results	Difference
Scene Center Location	Geodetic (longitude, latitude)	119.395060°E, 24.894620°N	119.395090°E, 24.894470°N	-3×10^{-5}, 1.5×10^{-4} (degree)
	ECR(x, y, z) (m)	-2817397.45 X, 5608996.11 Y, 2830630.29 Z	-2817396.77 X, 5608995.60 Y, 2830631.97 Z	distance: 0.06 (m) flight direction: 16.97 (m)

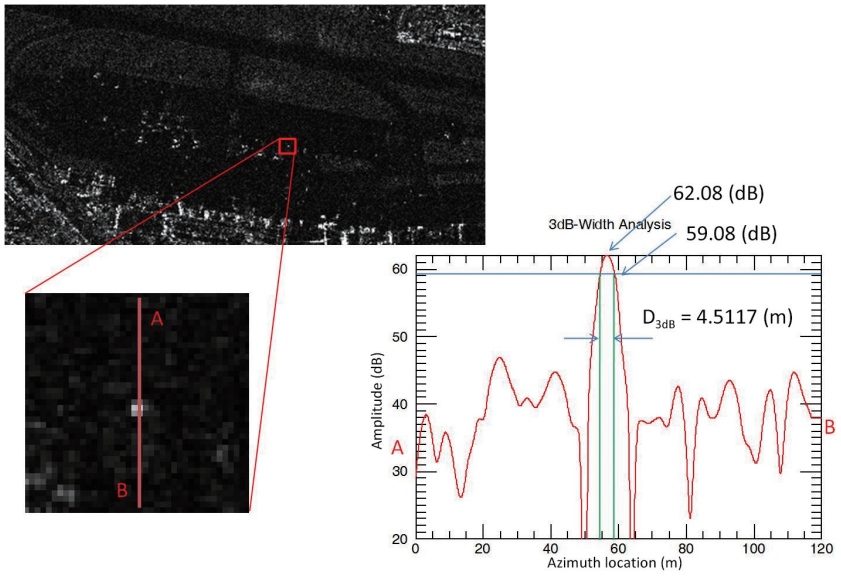


Fig. 14. Real TerraSAR-X point target image results for 3 dB width analysis. (Color online only)

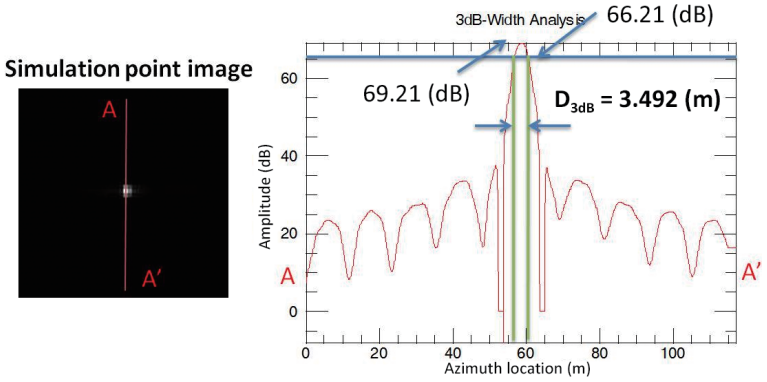
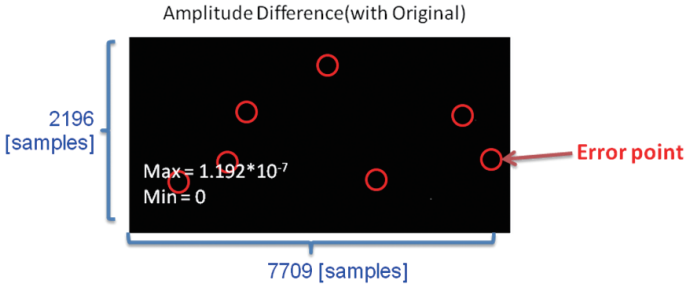


Fig. 15. Simulate TerraSAR-X point target image results for 3 dB width analysis. (Color online only)



- Overall accuracy:
  - [I,Q] → [10^{-11}, 10^{-13}] (Consistent with double precision)
  - Error point ration ≅ **0.0004% (max ≅ 10^{-7})**

Fig. 16. Accuracy verification of error point ration for TerraSAR-X point target image. (Color online only)

5. CONCLUSION AND FUTURE WORK

We introduced a framework of SAR image database simulation produced the amplitude and phase at the same time. The simulated SAR echo signal is certificated by the developed SAR processor evaluated by real satellite SAR echo signal data.

In this paper, we combined CPU with GPU, owned an integration of multi-programming environment based on MPI and CUDA, built a LBM cluster system to separate computing tasks from scheduling tasks. According to the results, the time complexity contributed from the calculation kernel is $O(n^3)$ reduced to $O(n^2)$, and it needs almost 10.3 months calculation under Intel 2.33 GHz (Quad cores) reduced to 1.06 days calculated under LBM-4GPU with one thread for all altitudes. Figure 13 shows the maximum differential amplitude value is almost 10^{-7} order. The error point ratio related to the entire image is less than 0.006%. According to these results the floating point precision using the LBM algorithm is the SAR image database product.

Our future work includes how to extend LBM for multi-GPU clusters in which each node may have more than one GPU installed and further excavate the LBM computation capability in heterogeneous multi-CPU based cluster systems.

REFERENCES

Aila, T. and S. Laine, 2009: Understanding the efficiency of ray traversal on GPUs. HPG ,09 Proceedings of the Conference on High Performance Graphics 2009, ACM New York, NY, USA, 145-149, doi: 10.1145/1572769.1572792. [Link]

Arora, N. S., R. D. Blumofe, and C. G. Plaxton, 2001: Thread scheduling for multiprogrammed multiprocessors. *Theor. Comput. Syst.*, **34**, 115-144, doi: 10.1007/s00224-001-0004-z. [Link]

Bai, H., L. He, D. Ouyang, Z. Li, and H. Li, 2009: K-means on commodity GPUs with CUDA. 2009 WRI World Congress on Computer Science and Information Engineering, Vol. 3, 651-655, doi: 10.1109/CSIE.2009.491. [Link]

Bailey, S. W. and P. J. Werdell, 2006: A multi-sensor approach for the on-orbit validation of ocean color satellite data products. *Remote Sens. Environ.*, **102**, 12-23, doi: 10.1016/j.rse.2006.01.015. [Link]

Bunt, R. B., D. L. Eager, G. M. Oster, and C. L. Williamson, 1999: Achieving load balance and effective caching in clustered Web servers. Proceedings of the Fourth International Web Caching Workshop, San Diego, California, 159-169.

Cardellini, V., E. Casalicchio, M. Colajanni, and P. S. Yu, 2002: The state of the art in locally distributed Web-server systems. *ACM Comput. Surv.*, **34**, 263-311, doi: 10.1145/508352.508355. [Link]

Cederman, D. and P. Tsigas, 2012: Dynamic load balancing using work-stealing. In: Hwu, W. W. (Ed.), GPU Computing Gems Jade Edition, Morgan Kaufmann, 485-499, doi: 10.1016/B978-0-12-385963-1.00035-6. [Link]

Chang, D. J., A. H. Desoky, M. Ouyang, and E. C. Rouchka, 2009: Compute pairwise Manhattan distance and Pearson correlation coefficient of data points with GPU. SNPD '09, 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 501-506, doi: 10.1109/SNPD.2009.34. [Link]

Colajanni, M., P. S. Yu, and D. M. Dias, 1998: Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Trans. Parallel Distr. Syst.*, **9**, 585-600, doi: 10.1109/71.689446. [Link]

Cumming, I. G. and F. H. Wong, 2005: Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation, Artech House Print on Demand, 660 pp.

Curlander, J. C. and R. N. McDonough, 1991: Synthetic Aperture Radar: Systems and Signal Processing, Wiley-Interscience, 672 pp.

Guim, F., I. Rodero, J. Corbalan, and M. Parashar, 2010: Enabling GPU and many-core systems in heterogeneous HPC environments using memory considerations. 2010 12th IEEE International Conference on High Performance Computing and Communications (HPCC), 146-155, doi: 10.1109/HPCC.2010.29. [Link]

Guo, K. Y., Q. Li, and X. Q. Sheng, 2010: A precise recognition method of missile warhead and decoy in multi-target scene. *J. Electromagn. Waves Appl.*, **24**, 641-652, doi: 10.1163/156939310791036250. [Link]

Harish, P. and P. J. Narayanan, 2007: Accelerating large graph algorithms on the GPU using CUDA. In: Aluru, S., M. Parashar, R. Badrinath, and V. K. Prasanna (Eds.), High Performance Computing – HiPC 2007: 14th International Conference, Goa, India, December 18-21, 2007. Proceedings, Springer Berlin Heidelberg, 197-208, doi: 10.1007/978-3-540-77220-0_21. [Link]

Henty, D. S., 2000: Performance of hybrid message-passing and shared-memory parallelism for Discrete Element Modeling. Proceedings of the IEEE/ACM SC2000 Conference (SC'00), IEEE, 9 pp, doi: 10.1109/SC.2000.10005. [Link]

Hsu, S. M., C. T. Wang, K. S. Chen, C. Y. Chiang, and J. R. Kao, 2008: ALOS/PALSAR Mission Operation in Taiwan. 4th Asian Space Conference & FORMOSAT-3/COSMIC International Workshop, Taipei, Taiwan.

Huang, C. W. and K. C. Lee, 2010: Frequency-diversity RCS based target recognition with ICA projection. *J. Electromagn. Waves Appl.*, **24**, 2547-2559, doi: 10.1163/156939310793675763. [Link]

Hui, C. C. and S. T. Chanson, 1999: Hydrodynamic load balancing. *IEEE Trans. Parallel Distr. Syst.*, **10**, 1118-1137, doi: 10.1109/71.809572. [Link]

Karunadasa, N. P. and D. N. Ranasinghe, 2009: Accelerating high performance applications with CUDA and MPI. 2009 International Conference on Industrial and Information Systems (ICIIS), 331-336, doi: 10.1109/ICIINFS.2009.5429842. [Link]

Kasim, H., V. March, R. Zhang, and S. See, 2008: Survey on parallel programming model. In: Cao, J., M. Li, M. Y. Wu, and J. Chen (Eds.), Network and Parallel Computing: IFIP International Conference, NPC 2008, Shanghai, China, October 18-20, 2008, Proceedings, Springer Berlin Heidelberg, 266-275, doi: 10.1007/978-3-540-88140-7_24. [Link]

Kunz, T., 1991: The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Trans. Software Eng.*, **17**, 725-730, doi: 10.1109/32.83908. [Link]

Lee, J. S., 1980: Digital image enhancement and noise filtering by use of local statistics. *IEEE Trans. Pattern Anal. Mach. Intell.*, **PAMI-2**, 165-168, doi: 10.1109/TPAMI.1980.4766994. [Link]

Lee, J. S. and E. Pottier, 2009: Polarimetric Radar Imaging: From Basics to Applications, CRC Press, 438 pp, doi: 10.1201/9781420054989. [Link]

Liu, B. and F. Shi, 2007: Research on Dynamic Load Balancing Algorithm Based on Message Passing Mechanism. *Comput. Eng.*, **33**, 58-60.

Loper, J. and S. Parr, 2007: Energy efficiency in data centers: A new policy frontier. *Environ. Qual. Manag.*, **16**, 83-97, doi: 10.1002/tqem.20144. [Link]

Margarit, G., J. J. Mallorqui, J. M. Rius, and J. Sanz-Marcos, 2006: On the usage of GRECOSAR, an orbital polarimetric SAR simulator of complex targets, to vessel classification studies. *IEEE Trans. Geosci. Remote Sensing*, **44**, 3517-3526, doi: 10.1109/TGRS.2006.881120. [Link]

Medhi, J., 1991: Stochastic Models in Queueing Theory, Academic Press, 450 pp.

Noaje, G., M. Krajecki, and C. Jaillet, 2010: MultiGPU computing using MPI or OpenMP. 2010 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 347-354, doi: 10.1109/ICCP.2010.5606414. [Link]

Padhy, R. P. and P. G. P. Rao, 2011: Load Balancing in Cloud Computing System, Department of Computer Science and Engineering National Institute of Technology, Rourkela, 1-56.

Pai, V. S., M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, 1998: Locality-aware request distribution in cluster-based network servers. Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, ACM New York, NY, USA, 205-216, doi: 10.1145/291006.291048. [Link]

Parker, S. G., J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, 2010: OptiX: A general purpose ray tracing engine. *ACM Trans. Graph.*, **29**, doi: 10.1145/1778765.1778803. [Link]

Rihaczek, A. W. and S. J. Hershkowitz, 2000: Theory and Practice of Radar Target Identification, Artech House Publishers, 738 pp.

Shi, R., S. Potluri, K. Hamidouche, X. Lu, K. Tomko, and D. K. Panda, 2013: A scalable and portable approach to accelerate hybrid HPL on heterogeneous CPU-GPU clusters. 2013 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 1-8, doi: 10.1109/CLUSTER.2013.6702619. [Link]

Shi, W. and Z. Tang, 1999: Dynamic computation scheduling for load balancing in home-based software DSMs. Proceedings of Fourth International Symposium on Parallel Architectures, Algorithms, and Networks, 1999 (I-SPAN '99), 248-253, doi: 10.1109/ISPAN.1999.778947. [Link]

Song, F. and J. Dongarra, 2012: A scalable framework for heterogeneous GPU-based clusters. Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM New York, NY, USA, 91-100, doi: 10.1145/2312005.2312025. [Link]

Srisuresh, P. and D. Gan, 1998: Load Sharing using IP Network Address Translation (LSNAT), RFC Editor, United States, No. RFC 2391.

Tian, B., D. Y. Zhu, and Z. D. Zhu, 2011: A novel moving target detection approach for dual-channel SAR system. *Progress In Electromagnetics Research*, **115**, 191-206, doi: 10.2528/PIER10120107. [Link]

Tzeng, S., A. Patney, and J. D. Owens, 2010: Task management for irregular-parallel workloads on the GPU. In: Doggett, M., S. Laine, and W. Hunt (Eds.), High Performance Graphics, Eurographics Association, 29-37.

Vasiliadis, G., S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, 2008: Gnort: High performance network intrusion detection using graphics processors. In: Lippmann, R., E. Kirda, and A. Trachtenberg (Eds.), Recent Advances in Intrusion Detection: 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008, Proceedings, Springer Berlin Heidelberg, 116-134, doi: 10.1007/978-3-540-87403-4_7. [Link]

Wang, L., Y. Huang, X. Chen, and C. Zhang, 2008: Task scheduling of parallel processing in CPU-GPU collaborative environment. ICCSIT '08, International Conference on Computer Science and Information Technology, IEEE, 228-232, doi: 10.1109/ICCSIT.2008.27. [Link]

Wang, X. F., J. F. Chen, Z. G. Shi, and K. S. Chen, 2011: Fuzzy-control-based particle filter for maneuvering target tracking. *Progress In Electromagnetics Research*,

118, 1-5, doi: 10.2528/PIER11051907. [Link]

Willebeek-LeMair, M. H. and A. P. Reeves, 1993: Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. Parallel Distr. Syst.*, **4**, 979-993, doi: 10.1109/71.243526. [Link]

Yang, C. T., C. L. Huang, C. F. Lin, and T. C. Chang, 2010: Hybrid parallel programming on GPU clusters. 2010 International Symposium on Parallel and Distributed Processing with Applications (ISPA), 142-147, doi: 10.1109/ISPA.2010.97. [Link]

Zheng, S. C., 2002: C program and MPI Hybrid Parallel Programming. Available at http://moodle.ncku.edu.tw/pluginfile.php/685069/mod_resource/content/0/mpic_2002.pdf. (in Chinese)