Task-Tree Based Large-Scale Mosaicking for Massive Remote Sensed Imageries with Dynamic DAG Scheduling

Yan Ma, *Member, IEEE*, Lizhe Wang, *Senior Member, IEEE*, Albert Y. Zomaya, *Fellow, IEEE*, Dan Chen, and Rajiv Ranjan, *Member, IEEE*

Abstract—Remote sensed imagery mosaicking at large scale has been receiving increasing attentions in regional to global research. However, when scaling to large areas, image mosaicking becomes extremely challenging for the dependency relationships among a large collection of tasks which give rise to ordering constraint, the demand of significant processing capabilities and also the difficulties inherent in organizing these enormous tasks and RS image data. We propose a task-tree based mosaicking for remote sensed imageries at large scale with dynamic DAG scheduling. It expresses large scale mosaicking as a data-driven task tree with minimal height. And also a critical path based dynamical DAG scheduling solution with status queue named CPDS-SQ is provided to offer an optimized schedule on multi-core cluster with minimal completion time. All the individual dependent tasks are run by a core parallel mosaicking program implemented with MPI to perform mosaicking on different pairs of images. Eventually, an effective but easier approach is offered to improve the large-scale processing capability by decoupling the dependence relationships among tasks from the complex parallel processing procedure. Through experiments on large-scale mosaicking, we confirmed that our approach were efficient and scalable.

Index Terms—Remote sensing image processing, parallel computing, DAG scheduling, data-intensive computing, big data computing

1 Introduction

In recent years the demands for accurate and up-to-date [1] environmental information at regional to global scale are increasing dramatically. The spaceborne sensors which conduct global observations [2] of the earth surface continuously have provided a unique and quantitative way for global monitoring. The efforts to improve the large scale capabilities of mosaicking [3] have received particular attention so to solve the problems of land use or climate changes [4], [5]. The Global Rain Forest Mapping (GRFM [5]) project has produced several semi-continental mosaics using some 13000 scenes of JERS-1 L-band SAR images. The United States Geological Survey (USGS) team has created Antarctica [6] mosaic from over 1,000 ETM+ scenes.

Mosaicking [7] of remote sensed (RS) imageries usually stitches a large collection of scenes with overlapping regions into a geometric alignment, radiometric balanced seamless one. Subsequently, a continuous view of the entire big area could be formed. The sheer volume of national-scale mosaic will be hundreds of gigabytes, and the continental mosaic could even exceed one terabyte [6]. Extremely large amounts of data need to be processed. Generally, mosaicking for remote sensed imageries mainly consists of re-projection, registration, radiometric balancing, seam line extraction and blending. This processing chain with multi-stage [7] is rather compute-intensive and also quite time-consuming. Generating a regional mosaic with a dozen of scenes would take couple of hours. When scale to large region, image mosaicking becomes extremely challenging due to the vast volume of image data, multiple processing steps with higher complexity, and also the intricate adjacent relationships among large collection of scenes with overlapping regions. Particularly, some time-critical applications like disaster assessment even require near real-time processing capabilities.

The enormous computation introduced by large-scale image mosaicking requires significant processing capabilities, which have far, outpace a single computer. Appling multi-core cluster based high-performance computing (HPC) in large-scale mosaicking is an effective solution to address these computational challenges. The parallelism is normally achieved by recursively breaking down large-scale processing issue into smaller tasks responsible for regional mosaics. In this way, certain tasks dealing with larger regions wait for the regional mosaics generated by much smaller tasks as input to be available. However, in the mosaicking scenario showed here, procedure consists of several complex processing stages, vast adjacent scenes with overlapping regions, and the tasks performing on these scenes subject to ordering constraints. The complexities inherent in organizing large numbers of ''messy''

- *Y. Ma and L. Wang are with the Institute of Remote Sensing and Digital Earth Chinese Academy of Science, Beijing 100094, China. E-mail: Lizhe.Wang@gmail.com.*
- *A.Y. Zomaya is with the School of Information Technologies, the University of Sydney, Australia.*
- *D. Chen is with the School of Computer, China University of Geosciences. E-mail: Danjj43@gmail.com.*
- *R. Ranjan is with ICT/CSIRO, Australia.*

Manuscript received 12 July 2013; revised 30 Sept. 2013; accepted 10 Oct. 2013. Date of publication 22 Oct. 2013; date of current version 16 July 2014. Recommended for acceptance by J. Chen.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.272

datasets and interdependent tasks will inevitably complicate the data slicing, task partition, also leads to trivial and frequent synchronization among tasks. Under this situation, the efficient parallel version of large-scale mosaicking algorithm would be rather difficult and error-prone, especially when implemented on low-level APIs message passing interface (MPI) enabled parallel system. Furthermore, the extra synchronization and communication overhead introduced by the interdependent tasks affect negatively on performance and scalability, when tasks are dispatched to different nodes in parallel system.

Direct Acyclic Graph (DAG) with vertexes for tasks and edges for constraints may be used to model applications consisting of interdependent tasks. DAG scheduling generates schedule with minimized overall parallel execution time or *schedule length* while without breaking the partial order of tasks that certain tasks must execute earlier than others. To properly address these aforementioned issues, we propose TTLMosaic, a novel Task-Tree based Large-scale Mosaicking for remote sensed imageries with dynamic DAG scheduling on multi-core cluster system. The main contribution of this work is that it introduces task tree together with dynamic DAG scheduling to decoupling the interdependent relationships among tasks from the complex parallel processing procedure. Through divide-and-conquer approach, the large-scale mosaicking problem as a root node recursively spawns out a collection of interdependent tasks. These tasks perform the same MPI implemented parallel mosaicking program on different datasets and generating corresponding regional mosaics. Then, the dependencies among these tasks form an abstract balanced task tree with nearly minimal height. Meanwhile, we adopt the improved dynamic DAG scheduling algorithm [8], [9], [10], [11], [12], [13] with job status queues so as to offer an optimized schedule that mapping the abstract task tree to multi-core cluster with minimal schedule length. Thereafter, a two-level parallelism is provided, the first level is the parallelism among tasks achieved by DAG scheduling, and the second level inside the tasks is parallel implemented by MPI on symmetric multiprocessors (SMP) nodes. Accordingly, this would offer higher parallel efficiency and more excellent scalability. In addition, the developing of efficient large-scale mosaicking program would also be much easier, since the MPI-implemented parallel program for individual tasks could no longer concern for the ordering constraint among interdependent tasks.

The rest of this paper is organized as follows. The following section discusses the related work, and the problem definition is addressed in Section 3. In Section 4, we go into details in the design and implementations of task-tree based large-scale mosaicking for RS imageries with dynamic DAG scheduling. Section 5 discusses the experimental analysis of program performance, and finally the last section summarizes this paper.

2 Related Work

The SMP clusters characterized by increasing scale as well as Graphics Processing Units (GPUs) are boosted employed in supercomputing domain. Several widely accepted parallel paradigms developed for hierarchical architecture of SMP cluster are used in image mosaicking algorithms, namely, OpenMP, MPI, and also MPI+OpenMP [16] hybrid paradigm which exploits multiple levels of parallelism. Work in [14] comparatively experimented the mosaicking algorithms with these three different parallel paradigms and all led to noticeable performance improvement. Work in [29] proposed an optimized image mosaic algorithm with parallel I/O and dynamic grouped parallel strategy on MPI-enabled cluster. In this method, the computing nodes are grouped for sub mosaics (tasks), while the master node carefully controls the whole mosaicking procedure of dependent tasks. The drawback of these works is that for an efficient implementation, a properly dealing with the frequent synchronization and communication among nodes with low-level APIs is required. But, this will lead to discouraging scalability. In addition, GPUs with CUDA programming have become a competitive accelerator [5]. Samargo *et al.* [28] presented CUDA accelerated mosaicking for unmanned aircraft system. The problem is that the complicated control logical of dependent tasks complicates the CUDA implementing. Moreover, to achieve excellent efficiency, a proper allocate of different memory types is required.

Nevertheless, the image mosaicking is challenged with collections of data-driven dependent tasks with ordering constraint. Thus, to develop parallel program for it solely with low-level but efficient programming models like MPI or CUDA would still be rather difficult and error-prone. In spite of the outstanding performance improvement, these kinds of approaches are only suitable for generating small regional mosaics, but not yet specialized in large-scale mosaicking with large collections of remote sensing images.

The DAG scheduling algorithms are employed for scheduling the dependent tasks of a parallel program onto cluster nodes to achieve minimal completion time. There are many choices of DAG scheduling solutions with trade-offs between time complexity and quality such as list-based scheduling heuristics including HLFET [17], MCP [18], ETF [19], and DLS [20], and the clustering scheduling approaches, e.g., EZ [8] and DCP [11].

The DAG scheduler is responsible for mapping task graphs to concrete processors, job execution and monitoring [21], [22], [23], [24]. Google's Pregel [25] a distributed computing framework with supersteps and message passing is suitable for large-scale graph processing. Recently, great efforts have laid on the study of algorithms [2], [7], [26], [27] to tackle variations in geometric and discontinuous in radiometric incurred telemetry characteristics when in large scale mosaicking scenario. On the contrary, the processing efficiency together with parallelization strategy of mosaicing for RS imageries at large scale is rarely pay attention to. Some limited related works go Merzky *et al.* [30] effectively retooling Montage [31] an existing astronomical image mosaicking software on distributed cyberinfrastructure through adopting DAGMan as a DAG scheduler and mDAG for describing Montage tasks as an abstract DAG.

The TTLMosaic for remote sensed imageries proposed in this paper aims at addressing the above challenges in large-scale capability and its performance efficiency. This

solution relies on the DAG scheduling technique as Montage does to provide efficient schedule of data-driven tree structured task graph (DAG) of mosaicking at large scale on parallel cluster systems. TTLMosaic consists of two modules: 1) DTTSchedular in charge of task tree construction and scheduling onto SMP processors; and 2) PMosaic a MPI-implemented parallel version of mosaic algorithm for automatically mosaicking two images or regional mosaics into a seamless single one.

3 Problem Definition

This section discusses the key issues related to the parallel mosaicking for remote sensed imageries at large scale on MPI-enabled multi-core cluster system. There are three aspects of this problem: large numbers of RS senses with overlapping regions and dependent tasks perform on them, difficulties of parallel implementation, along with data processing speed and scalability. The first issue is related to the complexities inherent in organizing tons of ''messy'' scenes with overlapping regions for task partition as well as the processing order determination (Section 3.1). The second issue is related to the difficulties lie in the parallel implementation with great concerning for trivial synchronization and communication among tasks incurred by their intricate dependency relations (Section 3.2). The third issue refers to the performance and scalability, namely how to sufficiently exploit parallelism and take advantage of hierarchical parallel architecture and especially data I/O (Section 3.3).

3.1 Large Numbers of RS Imageries and Tasks

When scale to large area, the image mosaicking will be overwhelmed with tons of RS datasets. The regional-scale mosaic covering Central Africa region (6° S-8° N and 5° E-26° E) [1] consists of more than 700 SAR scenes. Also, the Antarctica [6] mosaic merging over thousands of scenes taken by ETM+ sensor adds up to about more than one terabyte of data. However, these large numbers of RS images make the traditional mosaic on basis of scene-by-scene no longer inapplicable on parallel system. The reason for that is the intolerable time consumption and inevitable poor scalability with increasing processors.

The data acquired for large scale mosaicking are a mess of georeferenced RS images that most of the adjacent images have overlapping regions. Nevertheless, the intricate adjacent relationships among these images give rise to the complexities inherent in task partition and their processing order determination. Firstly, for a more fine-grained partition, the task partition does need be conducted recursively according to the intricate adjacent relations among images. In this way, the tasks produce larger mosaics depends on the smaller mosaics generated by other tasks as input images. As a result, the task partition is also a problem of properly representing large scale mosaicking in the form of data-driven task graph (DAG) which consists of a large collection of interdependent tasks. Withal, for performance efficiency, to express program as a nearly balanced task tree with almost minimal height is also paramount but also rather cumbersome. Secondly, to arrange these large numbers of interdependent tasks into an efficient processing order so as to gain low completion time could also be critical important.

3.2 Difficulties of Parallel Implementation

With the increasing demand for large-scale mosaic in regional to global-scale research, the mosaicking for RS imageries is complicated by discontinuous in radiometric and variations in geometric. The remote sensed image mosaicking would normally be a complex multi-stage processing procedure consists of re-projection, registration, radiometric balancing, seam line extraction and blending in overlapping regions. Of course, this complex processing procedure will make the parallel implementation difficult. To complicate the situation is that the parallel implementing have to tremendously concern for the dependency relationships among large numbers of tasks. The tasks should be executed with respect to the ordering constraint that some certain tasks have to wait for the smaller mosaics produced by other tasks to be available. This would subsequently lead to frequent and trivial process synchronization and data communication with MPI for processing sequence controlling. The efficient implementation of parallel mosaicking algorithm would be extremely difficult and error-prone.

Assume that if the dependency among tasks along with processing order controlling could be decoupled from the MPI implementation details, then the parallel implementation of mosaic would be much easier and efficient.

3.3 Performance and Scalability

When scale to large area, the image mosaicking turn out to be quite time-consuming and requires huge processing capability. Furthermore, some time-critical applications like disaster monitoring pose a challenge on performance. So, we should draw attention to exploiting the parallelism among tasks while abide by the ordering constraints, and also sufficiently benefiting from the hierarchical parallel architecture of cluster. Withal, the loading and exporting large amount of datasets introduce intensive data I/O operations and also undesirable I/O overhead. For performance and scalability consideration it is urgent to take I/O performance into account.

4 Design and Implementation

We demonstrate TTLMosaic a dynamic task-tree based mosaicking for remote sensed imageries with DAG scheduling. Here term ''Dynamic'' refers to the DAG scheduling method TTLMosaic adopts that the precedence constraint among the dependent tasks could be dynamically assigned during execution. TTLMosaic provides an efficient and easy scalable solution to greatly improve the large-scale processing capacity and performance of mosaicking on MPI-enabled multi-core clusters. This approach relies on the improved dynamic DAG scheduling algorithm with job status queues to decouple the ordering constraint among dependent tasks from the complex parallel processing procedure. TTLMosaic composes of DTTSchedular for task tree creation and efficient scheduling with minimal completion time (Section 4.2) and MPI-implemented PMosaic run by the individual tasks to perform mosacking on different images or mosaics (Section 4.1).

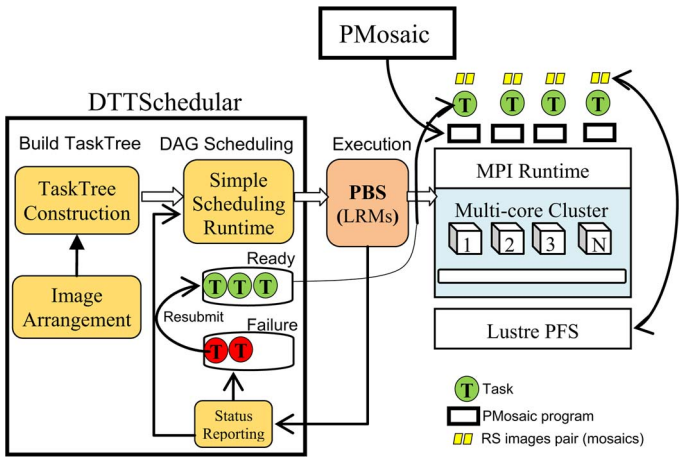

Fig. 1. System architecture diagram of DTTLMosaic.

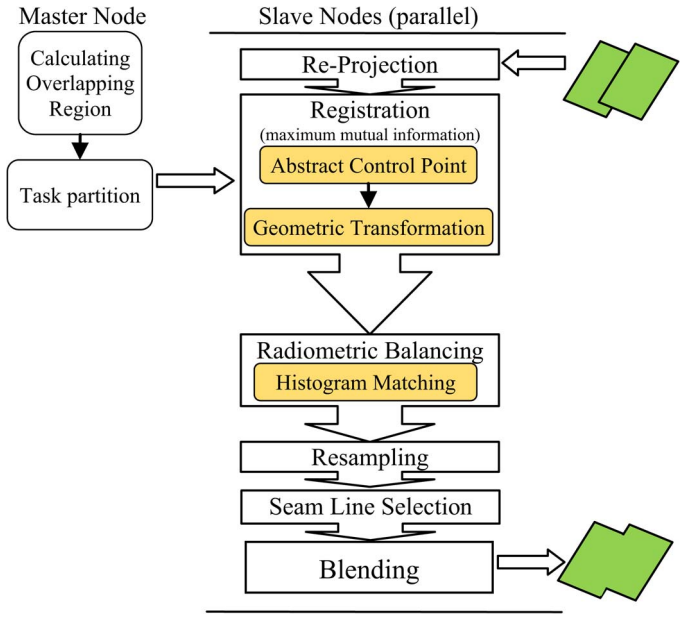

Fig. 2. Flow chart of the PMosaic program.

The system architecture diagram of TTLMosaic is showed in Fig. 1. We introduce a two-level parallel pattern in TTLMosaic. DTTSchedular exploits the parallelism among dependent tasks with parallel scheduling of task tree, while PMosaic achieves fine-grained parallelism inside the tasks with MPI.

To address the issues discussed in Section 3.1, we arrange the ''messy'' images into partial ordered sequence, and then express the large-scale mosaicking problem as a nearly balanced task tree based on this image sequence. Once the large collections of ''messy'' RS images are acquired for mosaicking, they will firstly be arranged into a partial row-path-major order sequence with the path and row number in geographic metadata of scenes. Then DTTSchedular build a nearly balanced binary task tree from the arranged image sequence by recursively cut the sequences into two subsequences with almost equal amount of images.

To solve the issues raised in Sections 3.2 and 3.3, DTTSchedular offers an effective and dynamical schedule for task tree and then submit the tasks to the LRMs (Local Resource Managers) of cluster like PBS for job execution. When a task tree is build, a simple DAG scheduling runtime is offered to produce a precedence-constrained and computation resource assigned concrete DAG with a Critical Path based Dynamical Scheduling solution with job Status Queue (CPDS-SQ). Then, the ''free'' tasks (no unscheduled children or preceding nodes) with high precedence will be put into ready queue and submit to PBS for mapping tasks to real processer resource in multi-core cluster. The scheduling runtime also checks PBS for job status. Once then a task is finished, the precedence of unscheduled task nodes would be dynamically reassigned, and the succeeding tasks waiting for input images to be available would be put into ready queue if ''free''. Thereafter, the DTTSchedular could execute these mosaic tasks in parallel while do not break the dependency among tasks. Accordingly, the parallel implementation of PMosaic with MPI could be much easier without concerning for the ordering constraints among tasks. In addition, a job resubmission mechanism is also provided in the simple scheduling runtime in case of failure. To improve the I/O performance discussed in Section 3.3, a SAN storage system equipped with high performance parallel file system Lustre is adopted for fast image data staging in and out among tasks. This SAN storage is fully connected with all computation resource. Also, the mechanism of overlapping I/O with computation is implemented in PMosaic.

4.1 Image Mosaicking Algorithm Implemented with MPI

When scaling to large area, the image mosaicking is complicated by the radiometric variations and radiometric discontinuities caused by telemetry characteristics [2]. Moreover, some of the processing steps are not fully automated in most common used commercial software for remote sensing image processing, like seam line extraction. Therefore, we adopt an image mosaicking approach consists of multiple stages, including re-projection, registration, radiometric balancing, seam line extraction and blending. The image registration is provided for automatic control point abstraction and geometric transformation between a pair of images with overlapping region. Then the geometric variation between images could be corrected. Meanwhile, the histogram match is used for radiometric balancing. Also, the seam line selection together with blending is employed for eliminating the artificial edges in the overlapping region introduced by the radiometric discontinuous between images. In addition, each individual processing step will follow an automatic way without manual intervention.

The flow chart of the PMosaic program implemented with MPI is showed in Fig. 2. The master computing node is responsible for data slicing and data partition. While the slave nodes follows a multi-stage processing procedure. Firstly, we re-project all the input images into a global projection coordinate system by choosing a proper projection method. Secondly, we adopt a maximum mutual information [32] based registration to abstract control points and establish geometric transformation automatically. Since the remote

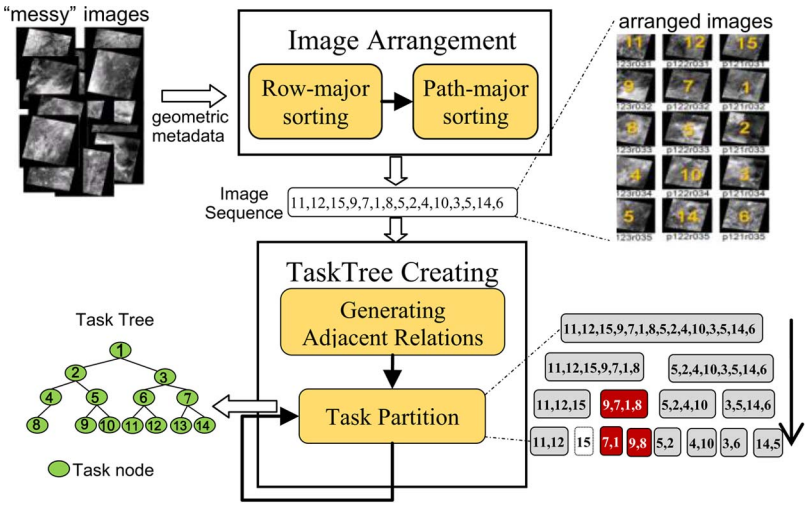

Fig. 3. Task tree creating method.

sensing images acquired for mosaicking are always geometric rectified, so it is ok to use a stable registration algorithm with lower quality. Thirdly, the radiometric balancing is achieved by histogram match between slave images and the selected referencing image. Then, the slave images are resampled with the geometric transformation function established in registration. Finally, we perform seam line selection and blending in overlapping region. For a fine quality of mosaic, we choose the laplacian pyramid based image blending algorithm [15]. For two images A and B, the lapalacian pyramid of overlapping regions in image A would be LA, and the he lapalacian pyramid of overlapping regions in image B is LB. Also GR the lapalacian pyramid of the overlapping region mask with optimized seam line is constructed as weight value for blending. Then through a weighted pyramid blending of LA and LB with weight GR in each pyramid layer and add up all the fusion layers to form a seamless mosaic.

For the I/O performance consideration, we adopt data pre-fetch approach that the data staging thread is offered for data staging in an out with data pre-fetching and writing with asynchronous parallel I/O. Accordingly, the I/O overhead could sufficiently overlap with the computation overhead. More detailed description could be found in our previous paper [29].

4.2 Dynamic Task Tree Scheduling for Large-Scale Mosaicking

DTTSchedular provides a dynamic task tree scheduling for large-scale mosaicking so as to improve its large scale capability. It represent large-scale mosaicking program in the form of task tree which is also a tree structured direct acyclic graph consisting of large numbers of tasks. DTTSchedular then relies on a dynamic MCP DAG scheduling algorithm to efficiently schedule the task tree onto multi-core cluster in parallel for a fast generating of the final mosaic with large scale.

4.2.1 Task Tree Construction Method

The large numbers of RS images for mosaicking at regional-scale to even global-scale are normally input in random. Therefore, the building of task tree would also be a procedure of image sorting and task partition. The main problem lays on that what kind of tree we should build and also how to build. To offer a fine-grained task partition along with a most simplified parallel processing procedure for a single task, each individual task could only generate a mosaic with two images. Namely, the task tree should be a binary tree that each non-leaf nodes have two children. Another important thing is the shape of this binary tree. The unbalanced tree which is also higher would lead to a long completion time, since fewer tasks could run in parallel at once. That is to say, it's good that each non-leaf node has two children. So a more flat shaped balanced task tree would be desirable.

In this paper, we adopt a simple task tree creating method which is also illustrated in Fig. 3. Firstly, a mess of images are arranged into a partial order sequence according to their path and row numbers which also indicate the approximate geographical location. In the geographical coordination system, the image scenes located at the top left corner would have a smaller row and larger path number. These scenes will be arranged in front of the sequence. This image sequence is arranged through a two-step sorting approach with a first row-major order sorting and a second path-major order sorting. Thus the images would be arranged in both ascending row order and descending path order.

Afterwards, we will build a task tree from this sequence through reclusive task partition while with respect to the adjacent relation among images. In this paper, we use adjacent table to represent the adjacent relation among images. If image k and image j have overlapping region then we mark this image pair as adjacent related in table. The adjacent table is generated by checking for each image pair whether an overlapping region is existed by comparing the range of image in projection coordinate system. Through a top-down task partition approach, we recursively divide this image sequence into a collection of sub sequences which corresponding to the nodes of task tree. In each step of the recursive partition, the sequence (corresponding to n_i) will be equally split into two sub sequences with almost

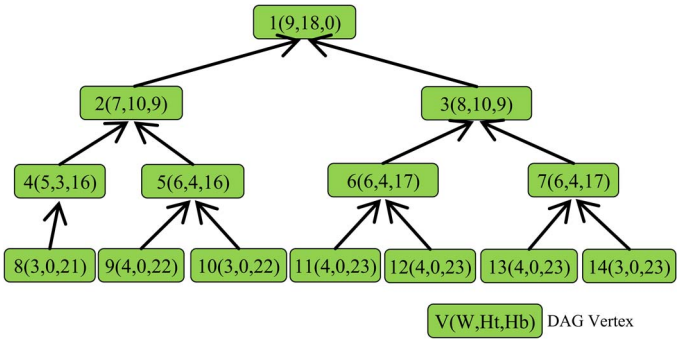

Fig. 4. DAG representation of task tree.

same number of images. One of the sub sequence called A consists of the images arrange in front of the middle image and the other sub sequence B includes the rest of the images. Then, the sub sequence A and B will be packaged as task nodes whose father is node n_i.

However, the above partition approach may introduce ''fake'' isolated images in sub sequence. Here ''fake'' isolated images refer the non-isolated images with adjacent images but do not have any adjacent relation with images in current sub image sequence. In case that a ''fake'' isolated image k is produced in sub sequence A, then we would select an image from B which do not adjacent to image k and also have the most adjacent relation with the images in A. Then, image k will swap with this selected image, the example of this is the red-colored sub sequence (9, 7, 1, 8) showed in Fig. 3. Following this way, we will make sure that each tasks dealing with two images or mosaics with overlapping region.

4.2.2 Task Tree Scheduling with CPDS-SQ

The task tree for large-scale mosaicking can also be treated as a tree structured direct acyclic graph. There are many choices for existing DAG scheduling algorithm with trade-offs between quality and time-complexity. In this paper, the data staging in and staging out is improved by a parallel file system Lustre enabled SAN storage fully connected with all computation resources. Whether the succeeding and previous nodes are run on same processor or not, the succeeding tasks waiting for the output data of preceding task to be available could staging in the input data from Lustre without explicit data communication. So, the data communication consumption among tasks is not considered in this paper. Moreover, PMosaic run the individual tasks is implemented with MPI among multiple processors. While the high quality clustering or task-duplication based algorithms with high time-complexity always aim at minimizing communication time and suppose that tasks are run on single processor. Therefore, instead of putting forward a new DAG scheduling algorithm, we adopt an improved critical path based dynamic scheduling solution with job status queue, short for CPDS-SQ, which is also base on dynamic list scheduling technique.

In this paper, we offer a simple DAG scheduling runtime to implement the above CPDS-SQ scheduling approach. It also dynamically submits the ready tasks to the LRMs of multi-core cluster PBS with computation resource assignment, along with the specification of data and task arguments.

4.2.3 Task Tree Representation with DAG Model

Following a Divide-and-Conquer way, the task tree is executed from bottom to top, since the father task nodes demand for the output smaller mosaic of their child nodes. So this task tree could be represented as a join DAG. The node n_i which is the child of task node n_k is also the preceding node of n_k. The DAG model $G = (V, E, W, Ht, Hb)$ is employed for task tree representation. Take the task tree built above in Fig. 3 for example, the vertex set $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$, the edge set $E = \{(1, 2), (1, 3), (2, 4), (2, 5), (3, 6), (3, 7), (4, 8), (5, 10), (5, 9), (6, 11), (6, 12), (7, 13), (7, 14)\}$. Where Ht of vertex V_i is the length of the longest path from an entry vertex to V_i, while Hb is the longest path from V_i to an exit vertex. The length of path is the sum of all node weights W along the path. Ht highly related to earliest start time of vertex, *while* $Hb(V_i)$ *corresponding to how many time left for processing the all the succeeding of vertex* (V_i). The definition of *Ht and Hb* is showed in following equations. In addition, the $pre(V_i)$ stands for the preceding vertex set of V_i, while $suc(V_i)$ stands for the succeeding vertex set of V_i

$$Ht(V_i) = \begin{cases} W(V_j) + \max_{V_j \in pre(V_i)}\{Ht(V_j)\}, & \text{if } pre(V_i) \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$Hb(V_i) = \begin{cases} \max_{V_j \in suc(V_i)}\{Hb(V_j)\} + W(V_i), & \text{if } suc(V_i) \neq \emptyset \\ W(V_i), & \text{otherwise.} \end{cases} \quad (2)$$

Run time Estimation offers $W(V_i)$ the estimated execution time of tasks V_i with the data amount $d(V_i)$ and the unit completing time of PMosaic program t when processing unit amount of data on single processor. As the tasks perform mosaicking at larger regions with larger amount of data would be more exhausted, so more computation resources will be accommodated. Assume that the number of processors assigned to task V_i is $n(V_i)$, and the responding performance speedup is $speedup(n(V_i))$. The unit completing time and performance speed up could be empirical value. Then the run time of task V_i could be estimated as follows:

$$W(V_i) = t * d(V_i)/speedup(n(d(V_i))).$$

DAG Representation of task tree built in Fig. 3 is depicted in Fig. 4. Where the execution time estimation value is given by $W = \{9, 7, 8, 5, 6, 6, 6, 3, 4, 3, 4, 4, 4, 3\}$. The directions of edges indicate the execution partial order.

4.2.4 CPDS-SQ DAG Scheduling

CPDS-SQ scheduling solution uses an earliest start time of a task as a priority. The earliest start time of a task is namely the Ht of the corresponding vertex or node. Initially, all the nodes in the DAG are assigned with an initial priority according to their Ht values. In case when some nodes have

a same earliest start time, then ties are broken by considering the $Ht + Hb$ of the nodes. Since the higher $Ht + Hb$ of node is, the more possible that the node would be on the critical path. So, by taking critical path into concern, the nodes with higher $Ht + Hb$ would be assigned with a higher priority.

After initialization of task priority, the CPDS-SQ solutions begin to scheduling the free tasks nodes recursively. Here free nodes refer to the tasks with no predecessors. This means that these tasks could be executed immediately without waiting for the dependent tasks to be ready. The scheduling starts with the entry nodes in the precedence-constraint DAG which also corresponding to the leaf nodes of task tree. Initially, all the entry nodes V_j are packaged into task packages modeling with $T_j = \langle V_j, D_j, Arg_j, N_j \rangle$ by assigning the amount of computation resources N_j, specifying the input image data $D_j$and processing arguments Arg_j. These task packages T_j are then constructed as a list in descending order of priority by the sorting algorithms and insert into a *ready queue*.

For each task packages in the ready queue, it will be submitted to a local resource manager of SMP cluster system name PBS for concrete computation resource accommodation and job execution. Once the task T_j is submitted to PBS, T_j becomes a scheduled task. Then CPDS-SQ scheduler moves scheduled task package T_j from ready queue to *running queue* for execution status reporting.

In case when the task scheduled task package T_j is finished, then CPDS-SQ will remove this finished task package T_j out from running queue and update the Ht value of the corresponding nodes with the real runtime of the task. Then a bottom to up priority recalculating procedure starts from task node V_j in task DAG would be invoked. Firstly, the priority of the unscheduled succeeding task nodes of V_j named $V_k(V_k = \mathrm{suc}(V_j))$ would be recalculated with the updated Ht value $Ht(V_k)$. Then, the priority of the unscheduled succeeding task nodes of V_k will be recalculated recursively until that node V_k has no succeeding tasks namely exit task node. Subsequently, CPDS-SQ will free task V_k the succeeding nodes of this finished node V_j by remove the dependent relationship edge (V_j, V_k) between task V_j and V_k from task DAG. If the succeeding node is a free node with no preceding tasks and ready for scheduling that it will also be packaged as task package and insert into ready queue.

While, when a failure of the task package is encountered, the task package will be moved from running queue to *Failure Queue* and mark the number of failures. The failure task packages will be rescheduled to ready queue with at most twice. This status updating and priority recalculation procedure will be looped until all the tasks are scheduled for execution.

The CPDS-SQ DAG scheduling solution could be expressed as follows. Where the T_j is the corresponding task package of node V_j, it also consists of data specification file D_j, argument file for execution Arg_j and the demand of computation resources N_j.

A simple scheduling runtime of DTTScheduler is offered to implement the CPDS-SQ DAG scheduling solution. The scheduling runtime diagram is presented in Fig. 5.

The CPDS-SQ DAG scheduling solution

```
1:  //Priority Initialization
2   for each node Vj in direct acyclic graph G do
3      P(Vj) ← Ht(Vj) priority assignment
4     for each node Vi < Vj do
5       if P(Vj) = P(Vi) then
6         if Ht(Vj) + Hb(Vj) > Ht(Vi) + Hb(Vi) then
7           P(Vj) ← higher priority
8         Else
9           P(Vj) ← lower priority
10    end for
11  end for
12
13  //Schedule free nodes
14   List = ∅
15  for each node Vj in direct acyclic graph G do
16    if node Vj is a free node then
17       Tj = ⟨Vj, Dj, Argj, Nj⟩
18       List ∪ Tj   //add task to list
19  end for
20  Sorting(P(List)) in descending order
21  ReadyQueue ∪ List   //insert List to ready queue
22  for each task Tj in ReadyQueue do
23     SubmitJob(Tj)
24     remove Tj from ReadyQueue
25     RunningQueue ∪ Tj
26  end for
27
28  // Status Updating and Priority Recalculating
29  for each task Tj in RunningQueue do
30    if Status (Tj) = finished then
31       Ht(Vj) ← realruntime(Tj)
32       // Priority Recalculating
33       Vk = suc(Vj)
34       while Vk ≠ ∅ then   //update from bottom to up
35         if Vk is un-scheduled then
36            Ht(Vk) ← recalculated value
37            P(Vk) ← recalculated value
38            Vk ← suc(Vj)
39       end while
40       remove Tj from RunningQueue
41       free suc(Vj) //free the succeeding nodes
42       for each node Vk in suc(Vj)
43         if Vk is ready then
44           ReadyQueue ∪ Vk //insert into readyqueue
45       end for
46    else if Status (Tj) = failure then
47       remove Tj from RunningQueue
48       FailureQueue ∪ Tj
49       if Tj have resubmit more than twice then
50         ReadyQueue ∪ Tj //resubmit failure jobs
51  end for
52  If exists unscheduled node in G and
53     RunningQueue ≠ ∅
54    and ReadingQueue ≠ ∅ then
55    goto line 14 for looping
```

The example schedule produced by CPDS-SQ scheduling solution on cluster with five processors is showed in Fig. 6. Assume that the task T_1 to T_3 which generate mosaic at larger region are assigned with two processors, and others are assigned with only one processor.

At the beginning, the $(V_j, W(V_j), Ht(V_j), Hb(V_j s))$ parameter of all the vertexes in DAG model would be (1, 9, 18, 0), (2, 7, 10, 9), (3, 8, 10, 9), (4, 5, 3, 16), (5, 6, 4, 16), (6, 6, 4, 17), (7, 6, 4, 17), (8, 3, 0, 21), (9, 4, 0, 22), (10, 3, 0, 22), (11, 4, 0, 23), (12, 4, 0, 23), (13, 4, 0, 23), (14, 3, 0, 23).

Firstly, the free nodes V_8 to V_{14} are arranged with the descending order of priority is $\{V_{11}, V_{12}, V_{13}, V_{14}, V_9, V_{10}, V_8\}$. So the node V_{11}, V_{12}, V_{13}, V_{14}, and V_9 are packaged into tasks $\{T_{11}, T_{12}, T_{13}, T_{14}, T_9\}$ and put into ready queue for scheduling. At the time of ''3'', task T_{14} finished, and the Task V_{10} will be submitted to PBS for execution. Then at the time of ''4'', task T_9, T_{11}, and T_{12} finished, the Task T_8 will be submitted to PBS for execution. At the time of ''6'', the task T_5 is becoming a free task node with no preceding tasks are inserted to ready queue for scheduling. Consequently, at the time of ''9'', the T_3 is free and scheduled to two processors by PBS. At the time of ''12'', the task T_2 is ready and executed on two processors. Finally, the root task T_1 is scheduled. Thus a total scheduling length of 27 is achieved.

5 Experiments

The task tree based large scale mosaicking algorithm TTLMosaic presented above offers an effective solution for fast mosaicking at large scale. It has successfully applied in the cluster-based PIPS (Parallel Image Processing System for remote sensing) for generating the regional-scale or even national mosaic using scenes taken by TM and ETM+ sensors.

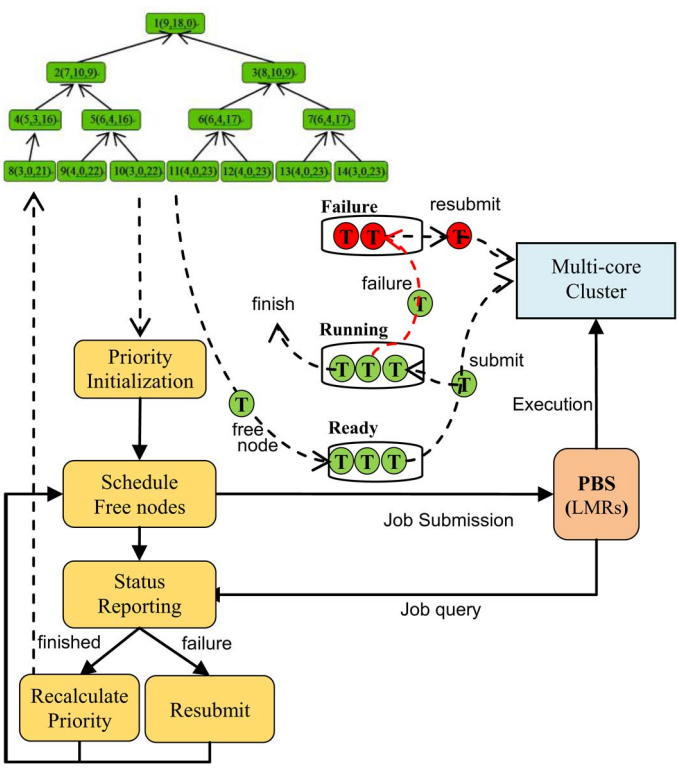

Fig. 5. Runtime diagram of CPDS-SQ DAG scheduling solution.

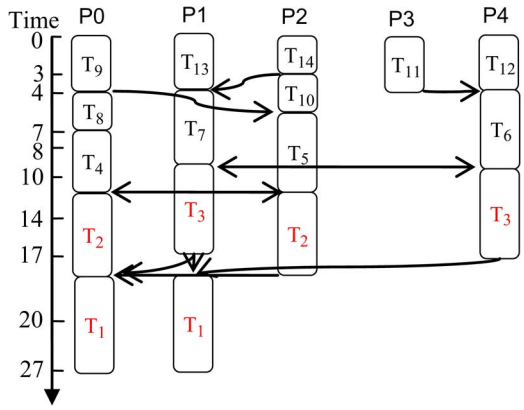

Fig. 6. Example schedule result of CPDS-SQ DAG scheduling.

The comparative experiment on regional mosaicking covering Bohai Bay (34° N-42° N and 115° E-120° E) was taken respectively with TTLMosaic and the commercial remote sensing image processing software ENVI. The final 30-meter regional mosaics consist of 15 scenes of datasets with single band taken by 30-meter TM sensor of Landsat 5 are demonstrated in Fig. 7, and the data amount of them are up to 2 gigabytes. Where the mosaic showed on the left side was generated by TTLMosaic on cluster using two SMP node (each node equipped with 8 cores) task about 18 minutes, and the other was generated by ENVI on workstation take about 50 minutes. From the experimental result we can see that the mosaic produced by TTLMosaic is radiometric balanced seamless one. While the mosaic produced by ENVI remains unhandled radiometric discontinuities along with explicit seam boundaries among images. Therefore, TTLMosaic algorithm not only greatly improves the visual quality of the final mosaic with a continuous view of the whole region, but also the time consumption is also reduced by using more processors.

The following experiments were carried out on a multi-core cluster with 10 multi-core nodes connected by a 20 gigabyte Infiniband using RDMA protocol. Each node is a blade server with dual Intel(R) Quad core CPU (3.0 GHz) and 8 GB memory. The operating system was Cent OS5.0, the C++ compiler was the Intel C/C++ Compiler with optimizing level O3, and the MPI implementation was Intel MPI.

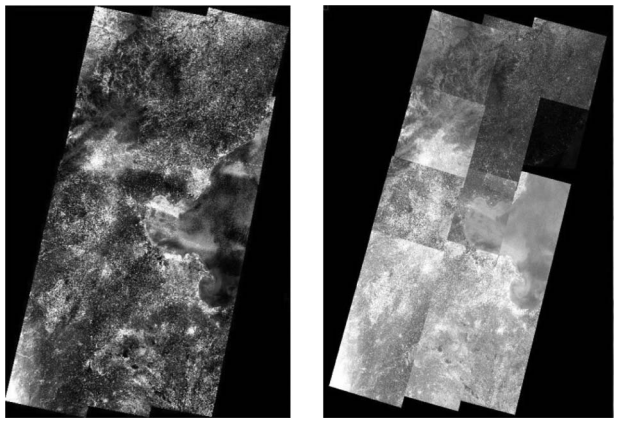

Fig. 7. Bohai bay mosaics produced by TTLMosaic (a) & ENVI (b).

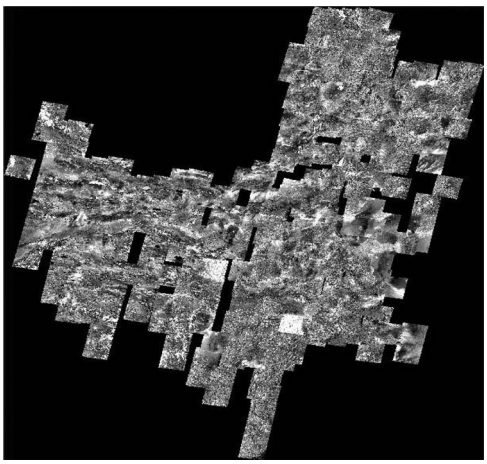

Fig. 8. Nation-wide mosaic produced by TTLMosaic.

The nation-wide mosaicking experiment covering most area of china (20° N-47° N and 84° E-180° E) was implemented on 10 nodes (80 cores) and took about 5 hours. While to complete the similar work, the common commercial software would take days or even weeks for extra manual intervention. The final nation-wide mosaic showed in Fig. 8 consists of s2 15-meter panchromatic ETM+ scenes of Landsat 7. The data amount of this mosaic adds up to be more than 120 gigabytes. In addition, we choose lambert conformal conic projection for this china-wide mosaic with two standard parallels of 25° N and 47° N, central meridian of 105° E.

The performance experiment is conduct on task tree based mosaicking program TTLMosaic (TTM) and traditional MPI-implemented parallel mosaicking program (PM) [28] both with increasing computation resources and with increasing amount image data respectively. The performance experiment with increasing processors chose the case of mosaicking 200 15-meter panchromatic ETM+ [6] scenes.

The traditional parallel mosaicking algorithm named PM is implemented on a basis of scene-by-scene that the parallelism is achieved by partition of the images into blocks. The master node arranges the mosaicking sequence of these scenes by generating a Minimum Spanning Tree. Following this way, only one scene of remote sensing image could be processed in parallel by computing nodes at one time and eventually stitched to form a seamless larger mosaic. The most common situation is that the computing nodes have to wait for each other with MPI synchronization primitives to abide by the order constraints of these scenes. Eventually, frequent synchronization and waiting overhead among nodes will result in poor efficiency and scalability, and also make the parallel program rather complicated.

Compared with TTM with two-level parallelism, the parallelism of PM may be relatively poor. When comes to large scale mosaicking with enough computing resources, lots of scenes could be mosaicked in parallel by TTM. While, PM could only dealing with one scene once, this will lead to some idle computing nodes. From proc and on, the advantage of PM algorithm is that each scene only have to be resampled once which retains the original information of pixels at a large extend.

TABLE 1
Performance with Increasing Node Scale

Nodes (Cores)	PM		TTM	
	RunTime(m)	Speedup	RunTime(m)	Speedup
1(1)	1130.580	1.000	1500.000	1.000
1(8)	340.075	3.325	292.797	5.123
2(16)	202.803	5.575	124.647	12.034
3(24)	154.745	7.306	79.407	18.890
4(32)	124.317	9.094	67.295	22.290
5(40)	101.793	11.107	61.784	24.278
6(48)	93.835	12.049	57.924	25.896
7(56)	88.064	12.838	49.836	30.099
8(64)	86.706	13.039	49.702	30.180
9(72)	88.239	12.813	50.150	29.910
10(80)	95.796	11.802	53.362	28.110

The results of the performance experiment with increasing processors which scaling form 1 node (8 cores) to 10 nodes (80cores) are listed in Table 1. While the performance metrics run time and speedup with increasing nodes are also respectively plotted in Figs. 9a and 9b.

From the experimental results plotted in Fig. 9 can see that DTTLMosaic (TTM) proposed in this paper presents more outstanding performance and scalability than the traditional parallel mosaicking program (PM) does with increasing amount of processors. The TTM parallelization of nation-wide remote sensing image mosaicking with DAG scheduling and MPI reduces the one processor time of 1500 minutes down to 49 minutes on 64 cores, for a speedup of 30. Compared with PM which purely implementation with MPI, TTM have decreases the PM runtime of 95 minutes to 53 minutes on 80 cores, while improves the PM speedup of 11.8 to 28. This means that the TTM reduces nearly a half of the runtime of PM with increasing processors or cores, while the speedup doubles. The main performance improvement would probably result from the efficient CPDS-SQ DAG scheduling solution. CPDS-SQ decouples the complex dependence relationships among tasks with ordering constraints from the complex parallel processing procedure and finally schedules the dependant tasks at a relatively minimal scheduling length.

However, when fewer processors are accommodated (less than 8 cores in Fig. 9a), TTM performs worse than PM. While, once enough computation resources are provided, TTM then outpaces PM immediately with fewer run time overhead. Note that TTM provides a two-level of parallelism, on is the parallelism among tasks through DAG scheduling and the other level is the parallelism among processors implemented by MPI. Naturally, the task tree construction, task tree scheduling and also the job submission and scheduling with PBS would inevitably introduce some extra time overhead which effect negatively on performance. When increasing processors are employed, the extra time overhead would not be a problem anymore, since the parallel processing time of TTM undergoes an obviously decrease.

The speedup of TTM is nearly linear with increasing nodes. Nevertheless, when more than 7 nodes (56 cores) are accommodated, the speedup of TTM declines. The reason for it is that when more than enough processors are offered, the processors required by free tasks ready for scheduling are less than the total resources, so there would be some

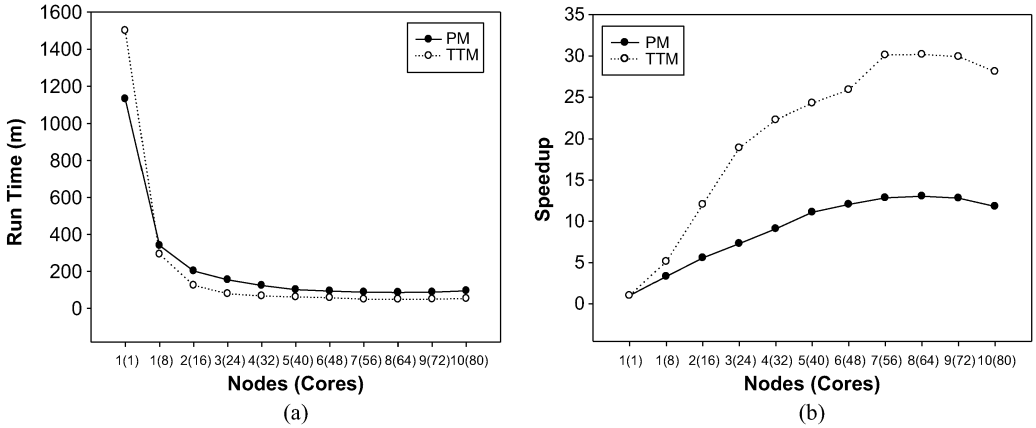

Fig. 9. Performance experimental result with increasing nodes.

idle processors then. The main reason for the TMM and PM all fail to scale linearly as the number of cores increase to more than 56 would be the I/O. The frequent data transferring among the dependent tasks and the data staging from one processing step to another within the multi-stage processing chain of mosaicking task will all result in quite impressive I/O time overhead. With better data cache or staging strategy and improved parallel I/O, we would expect to obtain a better speedup.

With the amount of panchromatic ETM+ scenes used for mosaicking increased from 2 to 800, the data amount increased from 0.7 to 273 gigabytes. The corresponding experimental results with the increasing region size of mosaic are listed in Table 2. Also the performance metrics run time and throughput with increasing amount of data are also respectively plotted in Figs. 10a and 10b.

Fig. 10a gives the picture of the run time comparing of the above two programs that DTTLMosaic perform explicitly more excellent scalability than PM with increasing amount of data which also means more amount of tasks for scheduling. From the curves we could know that the run time of DTTLMosaic decreases much faster than PM does. In case of generating large scale mosaic with 800 scenes (270 gigabytes), the TTM solution reduces the PM processing time of 1019 minutes down to 422 minutes, for a processing speed of 11 megabytes per second verses 4.5 megabytes per second of PM. Namely, when dealing with more image data or tasks, the run time overhead of DTTLMosaic (TTM) could just be less than a half of the time overhead of PM, while the processing speed triples. The reason for this is that with the increasing of data or tasks, more idle processors would be assigned to tasks, then the run time overhead could be greatly decreased with more tasks could be run in parallel. This is also the reason why that the data throughput performance of TTM far outpaces the performance of PM with increasing data amount.

Obviously, the data throughput performance of TTM goes worse than PM with limited cores less than 8, since the building and scheduling of task tree would task extra overhead as mentions in above speedup experiment. But when increasing cores are provided, the data throughput rate soars up sharply. Meanwhile, the discourage linear scalability with more than 700 scenes which adds up to a total of more than 230 gigabytes of data, would also dues to the rather considerable I/O overhead mentioned in previous experiment.

6 Conclusion

Remote sensed imagery Mosaicking at large scale turns out to be quite challenging for the large collection of interdependent tasks together with the significant processing capability requirements. As demonstrated above in this paper, we put forward DTTLMosaic a task-tree based mosaicking algorithm for remote sensed imageries which achieves a two-level parallelism with DTTSchedular for parallelism among tasks and PMosaci for inside tasks. DTTSchedular expresses large-scale mosaicking as a data-driven task tree with nearly minimal height and offers an optimized schedule on multi-core cluster with a critical path based dynamical DAG scheduling solution named CPDS-SQ. PMosaic the parallel image mosaicking program implemented with MPI is offered to perform mosaicking on different images by the individual tasks. This task-tree based approach offers an effective but easy way to improve the large-scale processing capability by decoupling the dependence relationships among tasks which waiting for input to be available from the complex parallel processing procedure. The experimental result shows that the final mosaic not only is radiometric balanced and but also greatly improved with no explicit seam boundaries among

TABLE 2
Performance with Increasing Data Amount

Image Scenes	Data (GB)	PM		TTM	
		Runtime (m)	Speed (MB/s)	Runtime (m)	Speed (MB/s)
2	0.7	3.828	3.048	6.780	1.721
12	4.1	15.996	4.376	16.280	4.300
25	8.5	33.240	4.387	25.470	5.726
50	17.1	64.600	4.515	37.900	7.696
130	44.4	158.184	4.794	59.000	12.853
210	71.8	256.102	4.783	93.000	13.172
300	102.5	397.852	4.399	143.000	12.238
370	126.5	476.772	4.527	176.000	12.263
530	181.2	646.714	4.781	257.000	12.030
700	239.3	873.291	4.676	345.000	11.836
800	273.4	1019.922	4.576	422.000	11.058

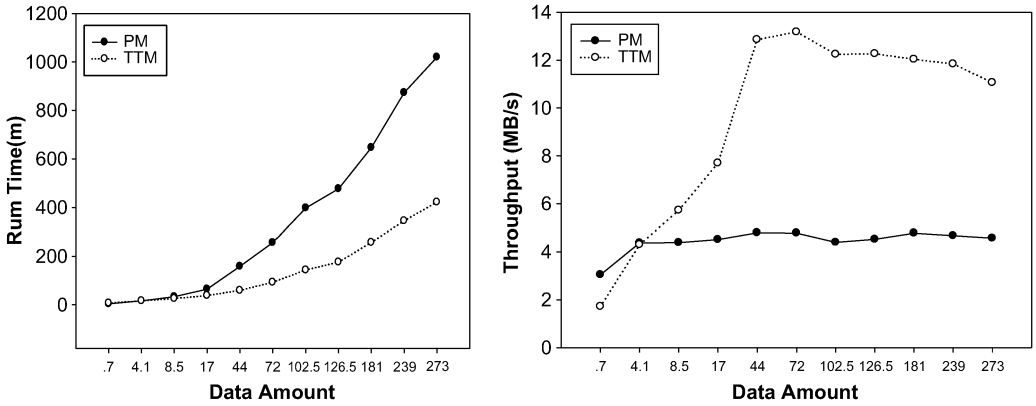

Fig. 10. Performance experimental result with increasing data amount.

images. The large-scale processing capability is also improved to be efficient by a china-wide mosaicking. Furthermore, DTTMosaic also perform a more outstanding performance and high scalability than traditional way of parallel mosaicking both with increasing computation resources and amount of image data or tasks. To draw the conclusion that the task-tree based mosaicking algorithm for remote sensed imageries with DAG scheduling offers an effective but easier approach to improve the large-scale processing capacity with higher scalability.

ACKNOWLEDGMENT

Dr. L. Wang and D. Chen are the corresponding authors. Dr. L. Wang's work is supported by National Natural Science Foundation of China (61361120098). Dr. Y. Ma's work is supported by Foundation for Young Scholars of Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences.

REFERENCES

[1] P. Mayaux, G.F. De Grandi, Y. Rauste, M. Simard, and S. Saatchi, ''Large-Scale Vegetation Maps Derived from the Combined L-Band GRFM and C-Band CAMP Wide Area Radar Mosaics of Central Africa P,'' *Int'l J. Remote Sens.*, vol. 23, no. 7, pp. 1261-1282, Apr. 2002.

[2] M. Shimada and T. Ohtaki, ''Generating Large-Scale High-Quality SAR Mosaic Datasets: Application to PALSAR Data for Global Monitoring,'' *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 3, no. 4, pp. 637-656, Dec. 2010.

[3] G. De Grandi, P. Mayaux, Y. Rauste, A. Rosenqvist, M. Simard, and S. Saatchi, ''The Global Rain Forest Mapping Project JERS-1 Radar Mosaic of Tropical Africa: Development and Product Characterization Aspects,'' *IEEE Trans. Geosci. Remote Sens.*, vol. 38, no. 5, pp. 2218-2233, Sept. 2000.

[4] Y. Guo, M. Shi, Y. Li, and D. Liu, ''Research on Fast Image Mosaic Based on CUDA,'' in *Proc. 4th ISCID*, Oct. 2011, vol. 1, pp. 198-201.

[5] A. Rosenqvist, M. Shimada, B. Chapman, A. Freeman, G.F. De Grandi, S. Saatchi, and Y. Rauste, ''The Global Rain Forest Mapping Project—A Review,'' *Int. J. Remote Sens.*, vol. 21, no. 6/7, pp. 1375-1387, Apr. 2000.

[6] B. Robert, V. Patricia, and F. Andrew, ''The Landsat Image Mosaic of Antarctica,'' *Remote Sens. Environ.*, vol. 112, no. 12, pp. 4214-4226, Dec. 2008.

[7] C. Bielski, J. Grazzini, and P. Soille, ''Automated Morphological Image Composition for Mosaicing Large Image Data Sets,'' in *Proc. IEEE IGARSS*, July 2007, pp. 4068-4071.

[8] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*. Cambridge, MA, USA: MIT Press, 1989.

[9] S. Kim and J. Browne, ''A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures,'' in *Proc. Int'l Conf. Parallel Process.*, 1988, vol. 3, pp. 1-8.

[10] T. Wajdi and A. Imitaz, ''Optimal Algorithm for Tree Scheduling with Unit Time Communication Delays,'' *Proc. Inst. Elect. Eng.—Comput. Digit. Tech.*, vol. 148, no. 2, pp. 79-88, Mar. 2001.

[11] Y.-K. Kwok and I. Ahmad, ''Dynamic Critical Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 5, pp. 506-521, May 1996.

[12] Y. Chung and S. Ranka, ''Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed-Memory Multiprocessors,'' *Proc. Supercomput.*, pp. 512-521, Nov. 1992.

[13] M.-Y. Wu, W. Shu, and J. Gu, ''Efficient Local Search for DAG Scheduling,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 617-627, June 2001.

[14] H. Wang, ''Parallel Algorithms for Image and Video Mosaic Based Applications,'' M.S. thesis, University of Georgia, Atlanta, GA, USA, 2005.

[15] P. Burt and E. Adelson, ''The Laplacian Pyramid as a Compact Image Code,'' *IEEE Trans. Commun.*, vol. COM-31, no. 4, pp. 532-540, Apr. 1983.

[16] R. Rabenseifner, G. Hager, and G. Jost, ''Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes,'' in *Proc. PDP Netw.-Based*, 2009, pp. 427-436.

[17] T.L. Adam, K.M. Chandy, and J. Dickson, ''A Comparison of List Scheduling for Parallel Processing Systems,'' *Commun. ACM*, vol. 17, no. 12, pp. 685-690, Dec. 1974.

[18] M.-Y. Wu and D.D. Gajski, ''Hypertool: A Programming Aid for Message-Passing Systems,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 3, pp. 330-343, July 1990.

[19] J.J. Hwang, Y.C. Chow, F.D. Anger, and C.Y. Lee, ''Scheduling Precedence Graphs in Systems with Interprocessor Communication Times,'' *SIAM J. Comput.*, vol. 18, no. 2, pp. 244-257, Apr. 1989.

[20] G.C. Sih and E.A. Lee, ''A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 175-187, Feb. 1993.

[21] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, ''Swift: Fast, Reliable, Loosely Coupled Parallel Computation,'' in *Proc. IEEE Congr. Services*, 2007, pp. 199-206.

[22] Karajan Workflow System. [Online]. Available: http://wiki.cogkit.org/wiki/Karajan

[23] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, ''Falkon: A Fast and Light-Weight tasK executiON Framework,'' in *Proc. ACM/IEEE Conf. SC*, Nov. 2007, pp. 1-12.

[24] DAGman Directed Acyclic Graph Manager, 2008. [Online]. Available: http://research.cs.wisc.edu/htcondor/dagman/dagman.html

[25] G. Malewicz, M.H. Austern, A.J.C. Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, ''Pregel: A System for Large-Scale Graph Processing,'' in *Proc. Int'l Conf. Manage. Data*, 2010, pp. 135-146.

[26] J. Kropáček, G. De Grandi, and Y. Rauste, ''Geo-Referencing of Continental-Scale JERS-1 SAR Mosaics Based on Matching Homologous Features with a Digital Elevation Model: Theory and Practice,'' *Int'l J. Remote Sens.*, vol. 33, no. 8, pp. 2413-2433, Apr. 2011.
[27] K. Kim, K.C. Jezek, and H. Liu, ''Orthorectified Image Mosaic of Antarctica from 1963 Argon Satellite Photography: Image Processing and Glaciological Applications,'' *Int'l J. Remote Sens.*, vol. 28, no. 23, pp. 5357-5373, Nov. 2007.
[28] A. Camargo, R.R. Schultz, Y. Wang, R.A. Fevig, and Q. He, ''GPU-CPU Implementation for Super-Resolution Mosaicking of Unmanned Aircraft System (UAS) Surveillance Video,'' in *Proc. IEEE SSIAI*, May 2010, pp. 25-28.
[29] Y. Wang, Y. Ma, P. Liu, D. Liu, and J. Xie, ''An Optimized Image Mosaic Algorithm with Parallel IO and Dynamic Grouped Parallel Strategy Based on Minimal Spanning Tree,'' in *Proc. 9th Int'l Conf. GCC*, Nov. 2010, pp. 501-506.
[30] A. Merzky, K. Stamou, S. Jha, and D.S. Katz, ''A Fresh Perspective on Developing and Executing DAG-Based Distributed Applications: A Case-Study of SAGA-Based Montage,'' in *Proc. 5th IEEE Int'l Conf. e-Science*, Dec. 2009, pp. 231-238.
[31] G.B. Berriman, J.C. Good, D. Curkendall, J. Jacob, D.S. Katz, T.A. Prince, and R. Williams, ''Montage: An On-Demand Image Mosaic Service for the NVO,'' in *Proc. ADASS XII*, 2002, p. 343.
[32] D. Mattes, D.R. Haynor, H. Vesselle, T.K. Lewellen, and W. Eubank, ''PET-CT Image Registration in the Chest Using Free-Form Deformations,'' *IEEE Trans. Med. Imaging*, vol. 22, no. 1, pp. 120-128, Jan. 2003.

Yan Ma received the DEng degree from Chinese Academy of Sciences, in 2013. She is an Assistant Professor at Institute of Remote Sensing & Digital Earth, Chinese Academy of Sciences (CAS), Beijing, China. Her research interests include high performance geo-computing and parallel remote sensing image processing. She is a member of the IEEE.

Lizhe Wang is a Professor at Institute of Remote Sensing & Digital Earth, Chinese Academy of Sciences (CAS), Beijing, China and a ''ChuTian'' Chair Professor at School of Computer Science, China University of Geosciences, Wuhan, China. Prof. Wang is a Fellow of IET and Fellow of BCS. He is a Senior Member of the IEEE.

Albert Y. Zomaya is currently the Chair Professor of High Performance Computing & Networking and Australian Research Council Professorial Fellow in the School of Information Technologies, The University of Sydney. He is also the Director of the Centre for Distributed and High Performance Computing. He is an IEEE Fellow.

Dan Chen is currently a Professor, Head of the Department of Network Engineering, and the Director of the Scientific Computing lab with School of Computer Science, China University of Geosciences, Wuhan, China. He was a HEFCE Research Fellow with the University of Birmingham, U.K. His research interests include computer-based modelling and simulation, high performance computing, and neuroinformatics.

Rajiv Ranjan received the PhD degree in engineering from the University of Melbourne, Australia, in 2009. He is a Research Scientist and a Julius Fellow in CSIRO Computational Informatics Division (formerly known as CSIRO ICT Centre). His expertise is in datacenter cloud computing, application provisioning, and performance optimization. He is a member of the IEEE.