An OpenMP-Based Algorithm for Multi-Nodes Computational of Super-Resolution

Jun Ma¹, Xiaoyong Wang¹,², FengLi², Xue Yang²

¹Spatial Data Processing Technology Laboratory of Henan University, Kaifeng, China
²Qian Xuesen Laboratory of Space Technology, Beijing, China

Abstract. A new common OpenMP based parallel programming method MPMC (multi-node paralleling model base on multiprocessor devices) is proposed and implemented for data separation based to accelerate Super-Resolution (SR) task. PanguOS, a common parallel programming system designed with MPMC, is deployed with Secure Shell (SSH) to control devices and Secure Copy (SCP) to transmit the data stream on Ubuntu 16.04, and it has a good performance for SR task with remote sensing images. Experiments with images from geostationary-orbit earth observing satellite GaoFen(GF)-4, the method proposed can achieve almost 2.95 times acceleration at PanguOS, deployed with 3 Jetson TX2s, than a single Jetson TX2.

1. Introduction

Imaging is a main way of information transmission between human beings and plays an indispensable role in our lives. However, images obtained cannot always meet the real scientific research requirements [1]. Especially, for space based remote sensing, onboard cameras are underdamping systems, and SR can help improve the resolution further. SR is a technique of reconstructing a high-resolution image from a series of low-resolution images captured within a short time slot, and it is widely used in many fields such as surveillance, medical instruments, remote sensing, etc. [1-4]. In general, there are two characteristics for SR:

(1) Time consumption for a SR task is huge, because there are many computational efforts involving a SR task.

(2) The volume of computational platform for SR is too big to be equipped onboard, because of the limitation of SWaP (size, weight, and power) with satellites.

In terms of parallel programming, it can be roughly divided into two categories: GPU based parallel programming and CPU based parallel programming [1,5]. CUDA plays an important role in GPU based parallel programming, since it has a large number of built-in highly efficient APIs [5,6]. On the contrary, MPI (Message Passing Interface) and OpenMP are widely used in CPU based parallel programming systems. MPI has become a standard for distributed computing while OpenMP is a memory sharing technology for parallel programming on multi-processor devices [5-7].

In this paper, a new OpenMP based parallel programming algorithm is proposed for distributed computing. The implementation of an MPMC algorithm, PanguOS, has been developed for a remote sensing image-based SR task on ARM devices.

2. Methodology
Nowadays, it is common to solve complex problems with heavy computations. This drives people to either develop efficient algorithms or try to accelerate the computations with parallel programming [7]. There are two strategies for parallel computing: task-based separation and data-based separation [1, 7]. Task based separation is a way for parallel computation by dividing tasks into sub-procedures which are not involved to each other. A complete task can be done by finishing each procedure in order. The data-based separation is a method for paralleling computing by cutting the data into some smaller parts and send those data to multiple computational units.

Ideally, a complex task of data-based separation can be described as follows:

$$ f(w) = \sum_{i=0}^{n} f_i(w) $$  \hspace{1cm} (1)

where \( n \) is the number of sub-procedures from a complex task, and \( f_i(w) \) is the computational cost for each sub-procedure \( i \). And only when each of the \( f_i(w) \) is finished, the task \( f(w) \) can be completed.

It is well known that parallel programming is often accompanied with some resource loss such as the data assignments, IOs, network transitions, etc. Therefore, the complex task finished with parallel programming can be modelled as the following equation:

$$ f(w) = \sum_{i=0}^{n} f_i(w) + \sum_{i=0}^{n} \Omega_i(w) $$  \hspace{1cm} (2)

where \( n \) is the number of sub-procedures from a complex task, and the \( f_i(w) \) is the cost of the sub-procedure \( i \). \( \Omega_i(w) \) is the loss when the processor processing \( f_i(w) \).

Network transmission cost and the IOs are the mainly cost for multi-devices distributed parallel programming [8], so the total loss \( \Omega_i(w) \) can be modeled as Eq. (3) shown.

$$ \sum_{i=0}^{n} \Omega_i(w) = \sum_{i=0}^{n} N(i) + \sum_{i=0}^{n} O(i) $$  \hspace{1cm} (3)

where \( N \) is cost from network transmission, and \( O \) is the consumption from IOs. From Eq. (3), we can know that, to improve computational efficiency, the best way is to reduce communication cost and IO operation.

It is well known that the more information interacted the more resources and time cost [6-9], because the network adapter cards and the switches always consume some time to understand the packages and resend the package to other devices. Therefore, MPI cannot be the most efficient method for distributed parallel programming, because it always needs more message interaction. Although, the cost listed above is almost negligible, it can affect the performance severely with the increase number of computational units.

In general, three phases of MPMC for parallel computing is: Data preparation, Data processing, Data cleaning.

![Figure 1. Steps of MPMC](image-url)
(1) Data preparation is the preparation operations before the data processing task, it can divide the raw data into some smaller parts and package those parts into some groups at a device regarded as a master node.

(2) Data Processing is the main task of MPMC. Not only some initialization operations, network checking, data transition and core functions are executed, but also some data processing control commands, and all human-computer interaction may be executed in this section.

(3) Data cleaning is the final phases of MPMC, the results can be transmitted to master devices, and those resolved data can be merged into a whole image and clean the work scene on each device. Moreover, procedures on master device and remote computational devices may be finished asynchronously. The next data preparation task can be processed while the data are running on the other remote computing devices.

As shown in Figure 1, the data separation can be finished on the master device. The master device sends some initialization commands with SSH and send the streams to remote devices by SCP commands. Then, the remote device can start to process the data, and write the result into the folder negotiated before the system deployed. After that, the master device can send SCP commands to download result from remote device to local and procedure on remote device is finished, and then the result data can be merged together on master device.

A series of computational units and the master device can be grouped together with a switch [6]. When the data is received by master device, the raw data can be divided into some smaller pieces and sent to remote computational units to process the data in a parallel model (a) shown in Figure 2.

![Figure 2. Networks](image)

MPMC, a common distributed computational algorithm, is a flexible data separation based parallel programming model for multiprocessor devices to accomplish a complex computational task. As (b) shown in Figure 2, the system can be extended flexibly with different number of nodes.

The leaves of root from 0 to N are the sub-nodes of the root device and it is also the sub-root device of its descendants. As shown in Fig. 3., every leaf node can also serve as a sub-root master device to manage all of its leaf nodes from A to N, provided that those nodes are computational nodes with a dual network card.

![Figure 3. Unlimited Leaves](image)

As shown in Figure 3, each leaf of the root device can rebuild a sub-parallel programming system with its leaves if the root equipped with a dual network card device (fortunately that almost all of current computational devices have dual or more network cards to meet the requirements of distributed
parallel programming). In this way, a parallel programming task can be extended to parallel programming with almost unlimited number of devices. Remote sensing images often have a large number of pixels, it may be \(10240 \times 10240\) pixels or even larger. It is difficult for ARM devices to accomplish the SR task on a single device because of the limited hardware resources cannot support to reconstruct the whole image in such a large volume. As the MPMC algorithm shown above, it drives us to accomplish the SR task for large remote sensing images.

As shown in Figure 4, the input low resolution remote sensing images should be cut into pieces in the first step. The average size of pieces depends on the number of available computational nodes. Those cut group data into some groups will be sent to other remote devices.

![Figure 4. Data Separation](image1.png)

Once the data preparation step of MPMC was done, and then MPMC begin to accomplish the SR task on different devices. Bear in mind, the data sent to computational devices may also exceed the maximum throughput capacity that a node can handle. If so, the data should be divided into some smaller size further, as (a) shown in Figure 5, such as \(512 \times 512\), that can finish the SR work with that block in a single run.

![Figure 5. Grid Operation](image2.png)

Every group of small pieces of remote sensing image can be processed at a time independently, and the reconstructed image files can be written into a folder. As (b) shown in Figure 5, a whole result image can be merged from those reconstructed image files by remote devices, and it can be saved into a folder negotiated before system being deployed that the master device can know where the result files are and then the master device can download the result by SCP command directly.

![Figure 6. Final Merge](image3.png)

The master device can download the reconstructed remote sensing images. As shown in Figure 6, the whole image can be eventually merged from a series of smaller image by the master device.

### 3. Experiments
Jetson TX2, as shown in Figure 7, an arm GPU device, was released from NVIDIA on 2017. It has a 64-bit dual-core Denver2 and a 64-bit quad-core ARM Cortex-A57 running at 2.0 Ghz, and a 256-core NVIDIA Pascal GPU running at 1.3 Ghz. The board has 8 GB of LPDDR4 RAM and 512 GB of storage (32 GB eMMC plus 480 GB SSD) [1]. Our SR task is designed with CUDA for Jetson TX2.
Figure 7. Jetson TX2

Speedup level \((S)\) and Parallel programming efficiency \((E)\) can be used to measure the improved performance of parallel computing \([1,12]\), as shown in Equation (4) and Equation (5).

\[
S = \frac{T_s}{T_p} \tag{4}
\]
\[
E = \frac{S}{p} \tag{5}
\]

where \(T_s\) is the time consumed by serial programming, \(T_p\) is the time consumed by parallel programming, \(p\) is the number of processors, \(S\) is the speedup ratio, \(E\) is the efficiency.

PanguOS, an implementation of MPMC, is consisted of 3 Jetson TX2 and a Gigabit router. All devices are set in the same LAN. The device 0 is set as the master and others are set as leaf nodes for computing. Since Jetson TX2 is a multi-core processor, the host 0 can also participate in the computation as a computing node. In order to have an experiment for an extension test, another Jetson TX2 is prepared for PanguOS. Our experimental data is based on GF-4 satellite data. GF-4 is the first geostationary-orbit remote sensing satellite of China, which was launched on December 28, 2015 with a visible light camera and an infrared staring optical camera onboard. The temporal resolution of GF-4 can be minute-level like a staring camera hanging above the earth \([2,3]\), so the GF-4 remote sensing image data is an ideal test data for our SR experiments. The raw image is \(10240 \times 10240\) pixels of size. The central ROI of the input images is reconstructed with PanguOS to make sure that the reconstructed ROI has the same size of the input images (By default, the reconstructed images are two times larger than that size of the original low-resolution images after super resolution). The acceleration results in average from 5 groups can be shown in Tab. 1.

| Size         | Serial | Parallel | S     | E     |
|--------------|--------|----------|-------|-------|
| 2000*2000    | 411    | 240      | 1.71  | 0.57  |
| 3000*3000    | 603    | 306      | 1.97  | 0.66  |
| 6000*6000    | 1055   | 447      | 2.36  | 0.79  |
| 6000*7000    | 2246   | 832      | 2.70  | 0.90  |
| 8000*8000    | 2572   | 890      | 2.89  | 0.96  |

Table 1. Experiments with different ROI size

We can see that with the increase of ROI area of raw image, from the Tab. 1, the computational speedup ratio and the efficiency of each board is increasing greatly.

It is well known that the performance can always be improved further by increasing of the number of devices, so we carry out another test with different number of devices. The results can be seen in Tab. 2.

| Devices | Serial | Parallel | S     | E     |
|---------|--------|----------|-------|-------|
| 1       | 2572   | 2572     | 1     | 1     |
| 2       | 2572   | 1326     | 1.94  | 0.97  |
| 3       | 2572   | 904      | 2.85  | 0.95  |
| 4       | 2572   | 732      | 3.51  | 0.88  |

Table 2. Numbers of devices of MPMC
From the above table, we can see that the speedup level is increased with the number of devices added, but the effective performance of each board has been decreased.

A raw remote sensing image from GF-4 is shown in (a) of Figure 8, we can see that it is very blurry. The lake and the river cannot be recognized easily, and the outline of the mountain is also not clear. In contrast to the low-resolution image, our reconstructed image can be seen in (b) of Figure 8. We can see that more detailed information and outlines of the lack, the river and the mountain peak can be recognized easily.

![Raw Image](image1.png) ![After SR](image2.png)

**Figure 8.** Comparison of raw image and image

In order to better evaluate the performance of MPMC, we finish a parallel application with MPI. Data separation task and data merge task are assigned to the rank 0, while data transition task and processing task are assigned to other ranks. The result is shown in Tab. 3.

| Size      | Serial | Parallel | S  | E  |
|-----------|--------|----------|----|----|
| 2000*2000 | 411    | 253      | 1.62 | 0.54 |
| 3000*3000 | 603    | 314      | 1.92 | 0.64 |
| 6000*6000 | 1055   | 459      | 2.30 | 0.73 |
| 6000*7000 | 2246   | 842      | 2.67 | 0.89 |
| 8000*8000 | 2572   | 1007     | 2.84 | 0.95 |

Comparing Tab 3 with Tab 1, we can see that the application with MPI has less speedup radio than MPMC, and the improved efficiency can reach 96% in contrast to MPI with 95%.

| Devices | Serial | Parallel | S  | E  |
|---------|--------|----------|----|----|
| 1       | 2572   | 2572     | 1  | 1  |
| 2       | 2572   | 1336     | 1.93 | 0.97 |
| 3       | 2572   | 917      | 2.80 | 0.93 |
| 4       | 2572   | 744      | 3.46 | 0.87 |

Comparing Tab. 4 with Tab. 2, we can see that MPI has less efficiency than that of MPMC, each processor of MPMC reaches 88% while that of MPI is 87%. Although there are few potentials having this difference from programming, the MPMC are more flexible than others. Supervisor is not necessary for each node of MPMC like MPI or others, and MPMC has fewer pre-steps to deploy paralleling environment.

4. Conclusion
Parallel computing is a thinking way to speed up the task in parallel, but limited within some specific algorithms or methods. In this paper, a flexible framework MPMC is proposed as a parallel algorithm...
which can finish some works in parallel with message passing by SSH and stream passing by SCP. In a word, MPMC can have communication asynchronously or communicate with other devices on demand, so it cannot occupy CPU continuously for waiting the response that can reach more higher performance. As experiments shown, the more time cost on each node, the less interaction between each of devices, and the better performance and efficiency MPMC is.

5. References

[1] Wang X, Li F, Ma J, et al. A New Parallel Scheduling Algorithm Based on MPI[C]. active media technology, 2018.
[2] Li F, Xin L, Guo Y, et al. Super-Resolution for GaoFen-4 Remote Sensing Images[J]. IEEE Geoscience and Remote Sensing Letters, 2018, 15(1): 28-32.
[3] F. Li, X. Jia, D. Fraser and A. Lambert. Super resolution for remote sensing images based on a universal hidden Markov tree model. IEEE Transactions on Geoscience and Remote Sensing, Vol 48, Issue: 3, Pages: 1270-1278, 2010.
[4] Dong C, Loy C C, He K, et al. Image Super-Resolution Using Deep Convolutional Networks[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016, 38(2): 295-307.
[5] Yang C, Huang C, Lin C, et al. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters [J]. Computer Physics Communications, 2011, 182(1): 266-269.
[6] Arai K. Secure Copier Which Allows Reuse Copied Documents with Sorting Capability in Accordance with Document Types[J]. International Journal of Advanced Research in Artificial Intelligence, 2013, 2(9).
[7] Cesar E, Cortes A, Espinosa A, et al. Introducing computational thinking, parallel programming and performance engineering in interdisciplinary studies[J]. Journal of Parallel and Distributed Computing, 2017: 116-126.
[8] Bellare M, Kohno T, Namprempre C, et al. Authenticated encryption in SSH: provably fixing the SSH binary packet protocol[C]. computer and communications security, 2002: 1-11.
[9] Broquedis F, Furmento N, Goglin B, et al. ForestGOMP: An Efficient OpenMP Environment for NUMA Architectures[J]. International Journal of Parallel Programming, 2010: 418-439.
[10] Charbonneau A, Terskikh V. SpectroGrid: Providing Simple Secure Remote Access to Scientific Instruments[C]. high performance computing systems and applications, 2008: 76-82.
[11] Chen W, Song Y, Bai H, et al. Parallel Spectral Clustering in Distributed Systems[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2011, 33(3): 568-586.
[12] J. Diaz, C. Muñoz-Caro, and A. Niño, “A survey of parallel programming models and tools in the multi and many-core era,” IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 8, pp. 1369–1386, 2012.

Acknowledgments

The third author acknowledges the support of National Key Research and Development Program of China (2016YFB0501301).