SDP Memo 065: Fast Implementation of SKA Algorithm Reference Library

Document Number ..... SDP Memo 065
Document Type ..... MEMO
Revision 1
Author ..... Haihang You, Runkai Yang, Tianzheng Wang, Tianchen Liu, Sheng Shi
Release Date 2018-08-20
Document Classification ..... Unrestricted
Status ..... Draft

Lead Author	Designation	Affiliation
Haihang You	Professor	Institute of Computing Technology, Chinese Academy of Science
Signature & Date:		

SDP Memo Disclaimer

The SDP memos are designed to allow the quick recording of investigations and research done by members of the SDP. They are also designed to raise questions about parts of the SDP design or SDP process. The contents of a memo may be the opinion of the author, not the whole of the SDP.

Table of Contents

1 Introduction

As the largest synthetic aperture radio telescope project in the world, Square Kilometre Array (SKA) has huge challenges for data processing task package, which demands tremendous computing power. Algorithm Reference Library (ARL) is the algorithm reference library of SKA-SDP, which systematically covers the data model and different stages of the whole radio telescope imaging process. ARL is written in Numpy based Python language. The advantage is that it is easy to understand many complex processes and is easy to use, but the performance of ARL is far from meeting the needs of astronomical data processing. To this end, we developed the optimized version of ARL, migrating the core and time-consuming algorithms in ARL, such as prediction, inversion and cleaning, to C++ platform, and use BLAS library to improve performance, and maintain transparent algorithm calls to the upper level. At the same time, we use the automatic parallel optimization tool Simulation Tuning Automatic Runtime (STAR) developed in shared memory environment for fine granularity partition and dependency assignment, so as to achieve more efficient dynamic scheduling and task processing. Finally, we use self- developed hybrid mode big data platform BATON to achieve efficient scheduling and computation of multiple imaging tasks in distributed environment for large-scale and efficient ARL algorithm imaging task processing.

2 Algorithm Reference Library

ARL is a reference algorithm library of the imaging processing developed for SKA, which include data model, pipeline, sky components such as a plurality of modules and calibration, imaging, FFT converter and other algorithms of radio astronomy data processing. The algorithm reference library is written with Python and Numpy, where Numpy is used for vector and matrix operations. As stated by Tim Cornwell, the developer of ARL, “The Algorithm Reference Library is designed to present calibration and imaging algorithms in a simple Python-based form”, and optimization is not the primary concern.

3 Fast implementation of ARL

However, in order to make ARL a practical tool for astronomical imaging processing, our optimization effort is to migrate the core algorithms of ARL from Python to C++, and further optimization applied according to the characteristics of the algorithms. Most importantly, we keep the transparency of the upper level algorithm calls written in Python.

4 Experimental results

We carried out four types experiments: single image processing pipeline, computational kernels and imaging algorithms.

4.1 Single Image Processing Pipeline

The experimental image processing pipeline includes prediction, inversion and cleaning. The experiments compare the performance of each stage of the process before and after the optimization. The experiment uses 10 channel, 7 groups of times and LOWDB2-CORE configuration (a total of 166 antennas). Results are shown in Table 1 and Fig 1.

Table 1: The performance of single image processing pipeline

	Predict(s)	invert(s)	Deconv(s)	Total(s)
Origin	8.65	13.98	23.34	45.97
Optimized	1.64	1.25	9.37	12.26
Speedup	x5.3	x11.2	x2.5	x3.7

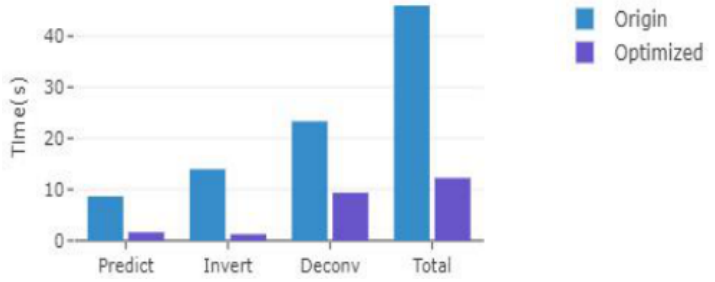

Figure 1: The performance of single image processing pipeline

It can be seen that after the migration to the c++ platform optimization, the overall image processing pipeline achieved 3.7 times speedup, and the performance of each stage of the pipeline improved respectively, where predict 5.3 times, invert 11.2 times, and deconvolution 2.5 times.

4.2 Computational Kernel

We compare the optimization effects of gridding, defridding, FFT and IFFT algorithm. The results are shown in Table 2 and Table 3.

Migrating from Numpy to C++, the degridding and gridding can achieve about 14 and 48 times performance enhancement, and after further optimization, the performance improvement reach 24 and 54 times respectively. Because of the low efficiency of python, and there are a lot of vector operations in the gridding/degridding algorithm, the preliminary optimization results are obvious. But we also note that gridding is not a parallelization friendly algorithm due to its strong data dependency, the algorithm restructuring and new implementation is necessary for further optimization.

4.3 Imaging Algorithm

We tested the performance of prediction and inversion algorithms in the image processing pipeline. These algorithms include base, wstack, timeslice, facets, wprojection and so on. Besides, w-snapshot algorithm is not evaluated here. The time comparison of each algorithm before and after optimization are shown in Table 4 and Table 5:

Table 2: The performance of gridding algorithm

	Gridding(s)	Defridding(s)
Origin	8.08	4.38
Optimized	0.55	0.09
Speedup	x14.7	x48.7

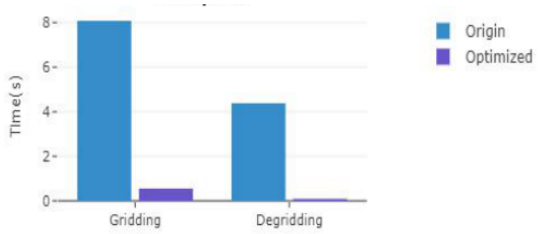

Figure 2: The performance of gridding algorithm

Table 3: The performance of FFT algorithm

	FFT(s)	IFFT(s)
Origin	0.63	0.37
Optimized	0.4	0.17
Speedup	x1.6	x2.2

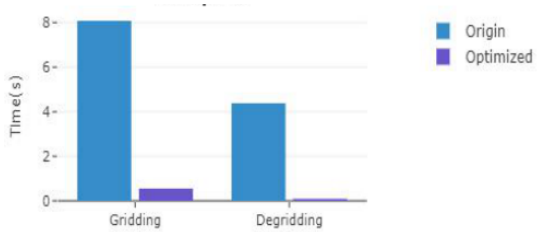

Figure 3: The performance of FFT algorithm

Table 4: The performance of Predict algorithm

	2D(s)	Wstack(s)	Timeslice(s)	Facets(s)	Wproject(s)
Origin	1.09	1.07	9.14	3.92	17.4
Optimized	0.28	0.3	8.64	0.77	9.22
Speedup	3.9	3.6	1.1	5.1	1.9

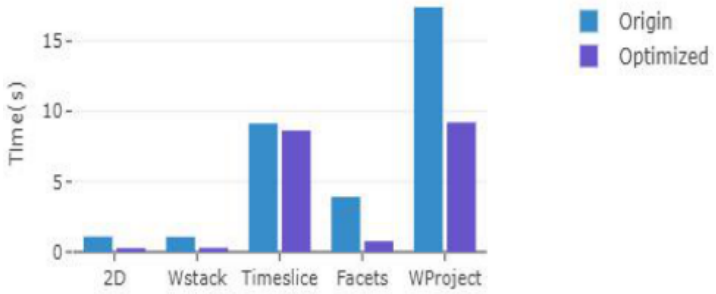

Figure 4: The performance of Predict algorithm

Table 5: The performance of Invert algorithm

	2D(s)	Wstack(s)	Timeslice(s)	Facets(s)	Wproject(s)
Origin	1.52	0.88	12.79	8.56	8.78
Optimized	0.15	0.23	12.14	3.21	6.41
Speedup	10.1	3.8	1.1	2.7	1.4

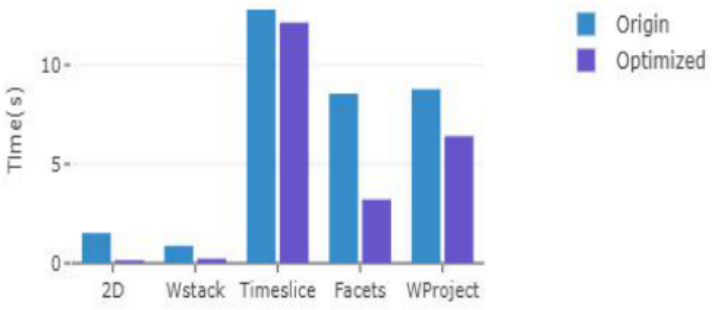

Figure 5: The performance of Invert algorithm

We can see good performance improvement for algorithms contain gridding and FFT operations such as base and facets; for wstack, timeslice, the performance is dwarfed by frequent function calls to small scale operation; wprojection has other time- consuming algorithms besides the gridding algorithm, and those algorithms need further optimization.

5 Future Work

The current work optimizes the ARL, and improves the performance of the image processing pipeline through the optimization of the computational kernels. It is unrealistic to rewrite all functions in ARL algorithm. The follow-up work is mainly to carry out performance analysis and explore the computational kernels for further optimization. We also plan to utilize dataflow programming model to enhance the performance and scalability on distributed computing platform for high throughput data processing.

we plan to use the automatic parallel optimization tool Simulation Tuning Automatic Runtime (STAR) developed in shared memory environment for fine granularity partition and dependency assignment, so as to achieve more efficient dynamic scheduling and task processing. Then, we plan to use self- developed hybrid mode big data platform BATON to achieve efficient scheduling and computation of multiple imaging tasks in distributed environment for large-scale and efficient ARL algorithm imaging task processing.

List of Figures

List of Tables

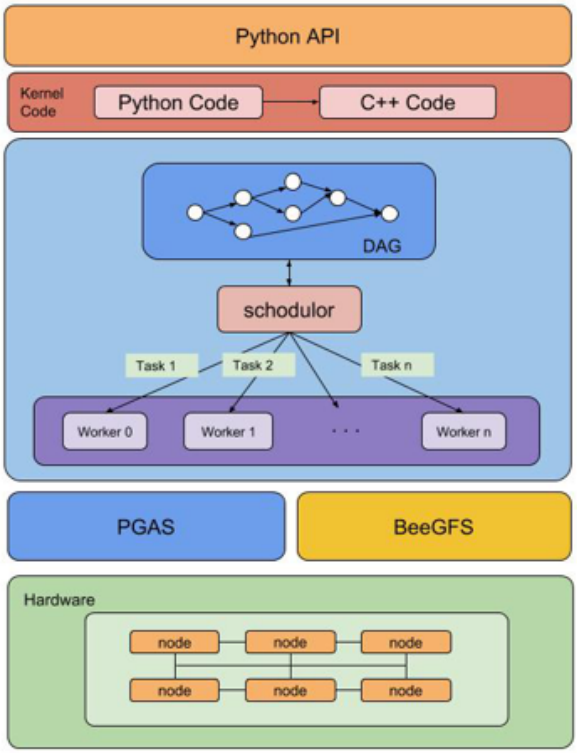

Figure 6: Self-developed hybrid mode big data platform