CRBLASTER: A Fast Parallel-Processing Program for Cosmic Ray Rejection in Space-Based Observations

Kenneth Mighell
National Optical Astronomy Observatory, 950 N. Cherry Ave., Tucson, AZ 85719

ABSTRACT

Many astronomical image analysis tasks are based on algorithms that can be described as being *embarrassingly parallel* – where the analysis of one subimage generally does not affect the analysis of another subimage. Yet few parallel-processing astrophysical image-analysis programs exist that can easily take full advantage of today's fast multi-core servers costing a few thousands of dollars. One reason for the shortage of state-of-the-art parallel-processing astrophysical image-analysis codes is that the writing of parallel codes has been perceived to be difficult. I describe a new fast parallel-processing image-analysis program called CRBLASTER which does cosmic ray rejection using van Dokkum's L.A.Cosmic algorithm. CRBLASTER is written in C using the industry standard Message Passing Interface library. Processing a single 800 x 800 Hubble Space Telescope Wide-Field Planetary Camera 2 (WFPC2) image takes 1.9 seconds using 4 processors on an Apple Xserve with two dual-core 3.0-GHz Intel Xeons; the efficiency of the program running with the 4 cores is 82%. The code has been designed to be used as a software framework for the easy development of parallel-processing image-anlaysis programs using embarrassing parallel algorithms; all that needs to be done is to replace the core image processing task (in this case the C function that performs the L.A.Cosmic algorithm) with an alternative image analysis task based on a single-processor algorithm. I describe the design and implementation of the program and then discuss how it could possibly be used to quickly do time-critical analysis applications such as those involved with space surveillance or do complex calibration tasks as part of the pipeline processing of images from large focal plane arrays.

1. INTRODUCTION

The writing of parallel-processing codes is frequently a challenging task frequently requiring complicated choreography of interprocessor communications. Although access to multi-core computing platforms is now readily available, few parallel-processing astrophysical image-analysis programs exist that can easily take full advantage of today's fast multi-core servers costing a few thousands of dollars.

Many astronomical image analysis tasks are based on algorithms that can be described as being *embarrassingly parallel* – where the analysis of one subimage generally does not affect the analysis of another subimage [1]. The writing of parallel-processing image-analysis codes based on embarrassingly-parallel algorithms can be a much easier task to accomplish due to their limited need for communication between compute processes.

Embarrassingly-parallel image analysis of a single image requires that the image be partitioned for processing on the compute processes. Fig. 1 shows a simple way that one may partition an image into 3 subimages for processing by embarrassingly-parallel algorithm on 3 compute processes. Partitioning the image over rows instead of columns would be logically equivalent.

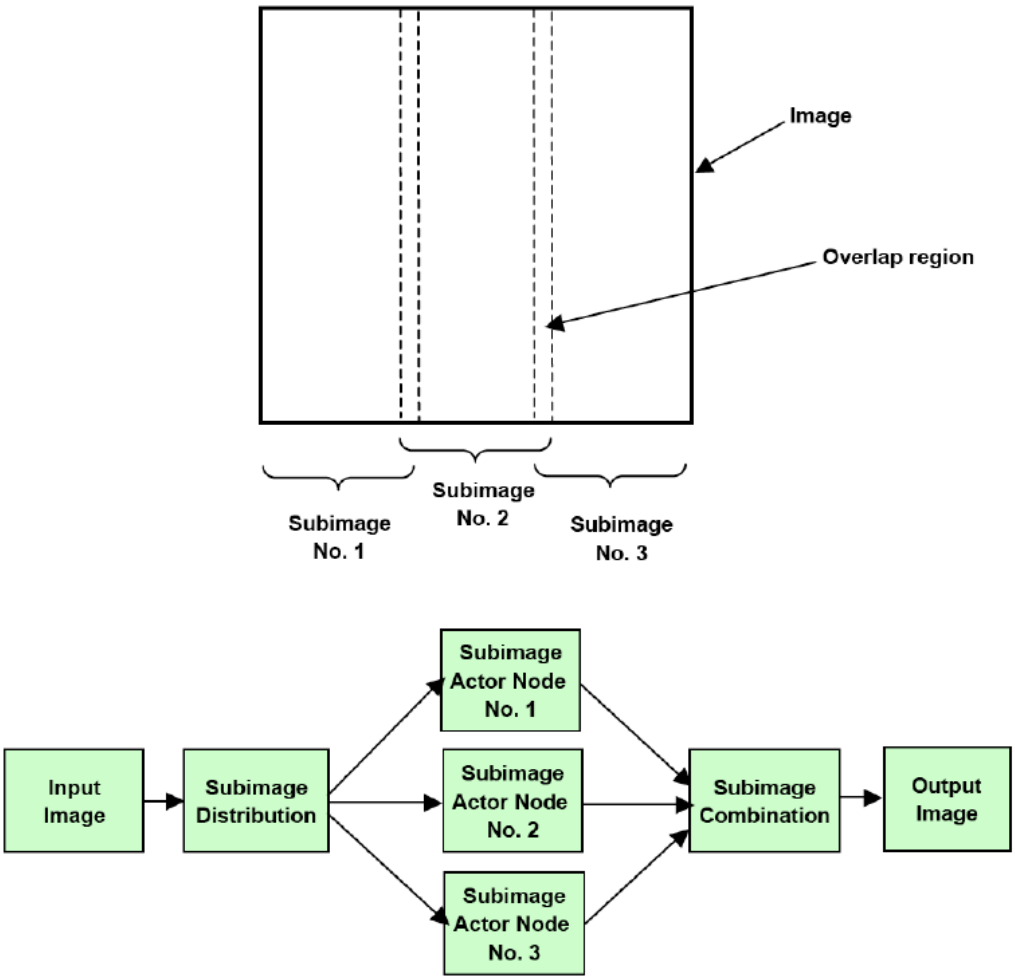

Fig. 1. A simple way to partition an image into 3 subimages for an embarrassingly parallel algorithm.

During the conversion of a traditional single-process image-analysis program to an embarrassingly-parallel program, one must consider carefully edge effects where additional data beyond the edges of a particular image partition may be required in order to do a proper computation of the algorithm. In order to qualify as an embarrassingly-parallel computation, one tries to avoid interprocess communication between compute processes whenever possible; this make the parallel-processing program much easier to write and possibly faster to execute. Consider, for example, a 7 7 x 7 median filter. An embarrassingly-parallel implementation of the 7 x 7 median filter (using the partition scheme shown in Fig. 1) would broadcast subimages that include an *overlap region of 3 pixels* so that the computation of the 4th column from a common partition edge would be correctly computed; the data in the overlap region is used but the algorithm is only applied to the non-overlapping columns. For a 600 x 600 input image, the subimages would thus have sizes of 203 x 600, 206 x 600, and 203 x 600 pixels. But what is to be done with the first and last 3 columns of the input image and the first and last 3 rows of the subimages? The 7 x 7 median filter is *not defined* in those regions and the coding of the algorithm in those areas of the input image are thus *implementation dependent*.

In this article, I describe the design, implementation, and performance of an embarrassingly-parallel image-analysis program which does cosmic ray rejection of CCD images. The code has been designed to be used as a software framework which enables the easy development of other parallel-processing image-analysis programs based on embarrassingly-parallel algorithms.

2. CRBLASTER

I have developed a new fast parallel-processing image-analysis program, called CRBLASTER[1], which does cosmic ray rejection using van Dokkum's L.A.Cosmic [2] algorithm. CRBLASTER is written in C using the industry standard Message Passing Interface (MPI)[2]. Processing a single 800 x 800 *Hubble Space Telescope (HST)* Wide Field Planetary Camera 2 (WFPC2) image takes 1.9 seconds using 4 processes on an Apple Xserve with two dual-core 3.0-GHz Intel Xeons; the efficiency of the program running with the 4 processors is 82%.

Fig. 2 shows the flowchart diagram of CRBLASTER. The director (master) process reads the input FITS image and splits it into N subimages (with overlap regions), and then sends them to the actor (slave) processes. Each actor process (including the director process) does cosmic ray rejection using the L.A.Cosmic algorithm on their own subimage, and then sends the resulting cosmic-ray cleaned subimage to the director process. The director process then collects all of the cosmic-ray cleaned subimages and combines them together to form the output cosmic-ray cleaned image which has the same size of the input image.

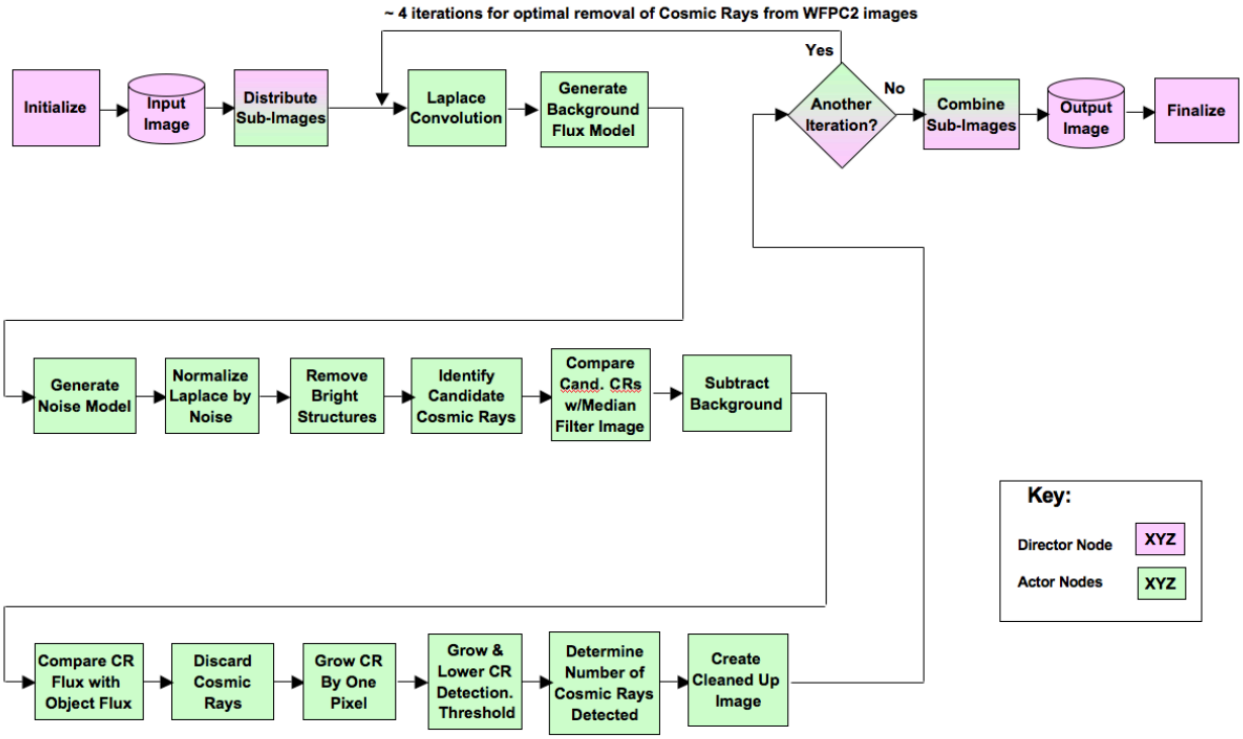

Fig. 2. Flowchart diagram of CRBLASTER.

The director process vertically partitions the input image into N subimages where N is the number of processes requested by the user. These subimages contain approximately 1/Nth of the image plus an overlap region that is BORDER pixels high above/below all joint partition edges. For the L.A.Cosmic algorithm, the optimal value of BORDER has been determined to be 6 pixels; using less than 6 pixels leaves many cleaning artifacts, while using more than six pixels does not improve the quality of the final output image.

The director/actor process(es) send/receive the subimages by using just two matching pairs of MPI_Send()/

[1] CRBLASTER is currently available at http://www.noao.edu/staff/mighell/crblaster

[2] The Message-Passing Interface (MPI) [3] was designed to be an industrial-strength message-passing environment that is portable across a wide range of hardware environments [8] . MPI is now the de facto industry standard for message-passing parallel-programming model [4][5], replacing virtually all other message passing implementationsused for production work. Most, if not all of the popular parallel computing platforms offer at least one implementation of MPI.

MPI_Recv() calls. The first pair sends/receives the subimage image structure as an array of (int)(sizeof(struct imageS_s)) bytes (MPI_Datatype MPI_CHAR). This programming hack greatly simplifies the program but it does come at a cost: CRBLASTER is assumed to be running within a homogeneous computing environment, which is a valid assumption for multi-core servers or Beowulf clusters composed of identical CPUs. The second pair of MPI_Send()/MPI_Recv() calls sends/receives the actual image data as an array of doubles (MPI_Datatype MPI_DOUBLE). This generally works well when the subimage data is being sent to another process. However, if the subimage data is being sent from the director to itself (in its role as an actor), then some implementations of MPI may hang due to some assumptions of the maximum size of message any sane user would wish to self transmit to/from a given process. Since the subimage arrays may be many megabytes in size, it is prudent to replace the MPI_Send()/MPI_Recv() calls with a simple memory copy (memcpy) whenever the director sends the subimage data to itself.

Running CRBLASTER on just one process gives a speed improvement of a factor of about 5.8 over the IRAF [6][7] implementation of the L.A.Cosmic algorithm (lacos_im.cl[3]). Fig. 3 shows the measured performance of CRBLASTER on a cluster with 20 processors. Wall time is the actual time of execution as measured on a clock on the wall. The blue circles gives the measured execution times for the core analysis functionof CRBLASTER running with 1–10 and 20 processors. The green straight line gives expected speedup based on the ideal model of a purely parallel algorithm. The red curve gives the expected speedup based on a more realistic model that accounts for the transmission time of the subimages which include redundant pixels in the overlap regions. The left image is part of the WF3 dataset of the *HST* WFPC2 observation U3060302M.C0H of the galaxy cluster MS 1137+67; the right image shows the result of cosmic ray rejection using CRBLASTER (compare Fig.~6 with Fig. 6b of [2]).

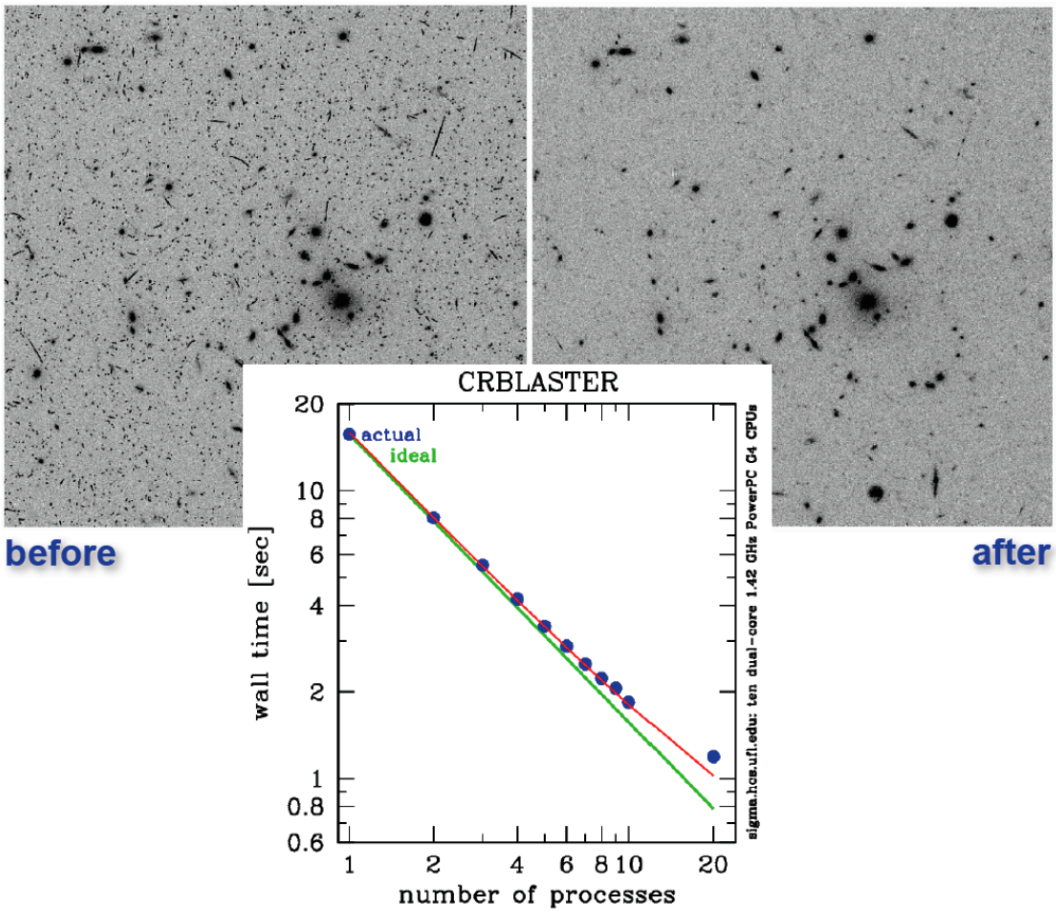

Fig. 3. The measured performance of CRBLASTER on a cluster with 20 processors.

[3] Currently available at http://www.astro.yale.edu/dokkum/lacosmic/lacos_im.cl

Note the nearly ideal improvement in the processing time of the core analysis function. The time spent in the non-parallelizable portion of CRBLASTER (mainly the reading/writing of the input/output images) could be greatly minimized by reading/writing the image data on/off a ramdisk instead of a physical hard disk drive as reading/writing from/to memory is much faster than to spinning magnetic disks.

3. SOFTWARE FRAMEWORK

The CRBLASTER code can be used as a *software framework* for easy development of parallel-processing image-anlaysis programs using embarrassing parallel algorithms. The biggest required modification is the replacement of the core image-processing function with an alternative image-analysis function based on a single-processor algorithm. If the new algorithm needs larger or smaller overlap region of the subimages, then the numerical value of the BORDER macro [#define BORDER (6)] should be modified to the appropriate value for the new algorithm. And of course, the command line options will need to be modified to provide the new algorithm information about any required parameters. Beyond these simple modifications, nothing within the main software framework needs to be touched.

Using CRBLASTER as a software framework, one should be able to implement time-critical analysis applications such as those involved with space surveillance or do complex calibration tasks as part of the pipeline processing of images from large focal plane arrays.

4. SUMMARY

Many astronomical image-analysis programs are based on algorithms that can be described as being embarrassingly parallel, where the analysis of one subimage generally does not affect the analysis of another subimage. Yet few parallel-processing astrophysical image-analysis programs exist that can easily take full advantage of today's fast multi-core servers costing a few thousands of dollars. A major reason for the shortage of state-of-the-art parallel-processing astrophysical image-analysis codes is that the writing of parallel codes has been perceived to be difficult. I described a new fast parallel-processing image-analysis program called CRBLASTER which does cosmic-ray rejection using van Dokkum's L.A.Cosmic algorithm. CRBLASTER is written in C using the industry standard Message Passing Interface (MPI) library. The code can be used as a software framework for easy development of parallel-processing image-anlaysis programs using embarrassing parallel algorithms; the biggest required modification is the replacement of the core image-analysis function (in this case the C-version of the L.A.Cosmic algorithm) with an alternative image-analysis function based on a single-processor algorithm.

I wish to thank Dr. John Samson of Honeywell Inc., Aerospace Systems for use of Figs. 1 and 2. This work has been supported by a grant from the National Aeronautics and Space Administration (NASA), Interagency Order No. NNG06EC81I which was awarded by the Applied Information Systems Research (AISR) Program of NASA's Science Mission Directorate.

5. REFERENCES

[1] Wilkinson, B. and Allen, M., *Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers*, Pearson Education Inc., 2nd edition, 2004.
[2] van Dokkum, P. G., "Cosmic-Ray Rejection by Laplacian Edge Detection," *Publications of the Astronomical Society of the Pacific,* Vol. 113, 1420–1427, 2001.
[3] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J., *MPI – The Complete Reference (Volume 1: The MPI Core)*, MIT Press, 2nd edition, 1998.
[4] Gropp, W., Lusk, E., and Skjellum, A., *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, 2nd edition, 1999.
[5] Pacheco, P. S., *Parallel Programming with MPI*, Morgan Kaufmann Publishers, Inc., 1997.
[6] Tody, D., "The IRAF Data Reduction and Analysis System," *Proceedings of SPIE,* Vol. 627, 733–739, 1986.
[7] Tody, D., "IRAF in the Nineties," *Astronomical Society of the Pacific Conference Series 52: Astronomical Data Analysis Software and Systems II* , 173–183, 1993.
[8] Dowd, K. and Severance, C. R., *High Performance Computing*, O'Reilly & Associates, Inc., 2nd edition 1998.