CRBLASTER: Benchmarking a Cosmic-Ray Rejection Application on the Tilera 64-core TILE64 Processor

Kenneth John Mighell
mighell at noao dot edu
National Optical Astronomy Observatory
950 North Cherry Avenue, Tucson, AZ 85719

Abstract

While the writing of parallel-processing codes is, in general, a challenging task that often requires complicated choreography of interprocessor communications, the writing of parallel-processing image-analysis codes based on embarrassingly-parallel algorithms can be a much easier task to accomplish due to the limited need for communication between compute processes. I describe the design, implementation, and performance of a portable parallel-processing image-analysis application, called CRBLASTER, which does cosmic-ray rejection of CCD (charge-coupled device) images using the embarrassingly-parallel L.A.COSMIC algorithm. CRBLASTER is written in C using the high-performance computing industry standard Message Passing Interface (MPI) library. A detailed analysis is presented of the performance of CRBLASTER using between 1 and 57 processors on a low-power Tilera 700-MHz 64-core TILE64 processor. The code has been designed to be used by others as a parallel-processing computational framework that enables the easy development of other parallel-processing image-analysis programs based on embarrassingly-parallel algorithms.

Introduction

One reason for the shortage of state-of-the-art parallel-processing astrophysical image-analysis codes is that the writing of parallel codes is perceived to be difficult. When writing parallel-processing software, a programmer must learn to deal with challenges associated with the choreography of multiple processes which typically can have complex interaction rules and requirements. Some of these interactions can cause problems never encountered with serial-processing software. Professional-grade parallel-processing debugging software tools are complex and sophisticated because they must detect these and other dynamic programming errors in real time; not surprisingly, these tools can easily cost many thousands of dollars for a single-user license.

Programming algorithms can be classified as being "embarrassingly parallel" if they have computational work loads that can be divided into a number of (nearly) independent parts that are executed on separate processes; each compute process does its work independently with no (or little) communication between the other compute processes [1].

Embarrassingly-parallel image analysis of a single image requires that the input image be segmented (partitioned) for processing by separate compute processes. Figure 1 shows a simple way that one may segment an image into 3 sub-images for processing by embarrassingly-parallel algorithm on 3 compute processes. Segmenting the image over rows instead of columns would be logically equivalent.

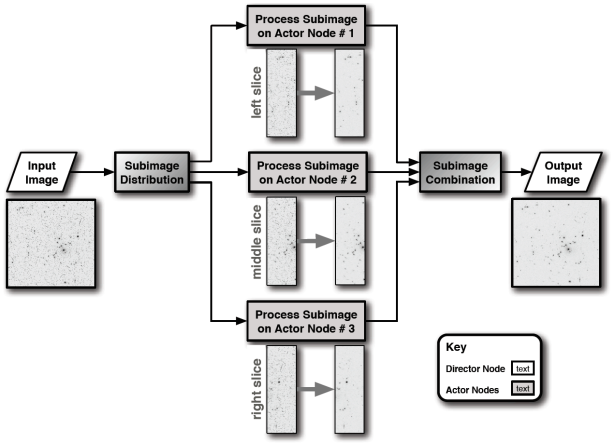

Figure 1: A simple way to segment an image into 3 subimages for an embarrassingly parallel algorithm.

Many low-level image processing operations involve only local data with very limited (if any) communication between areas (regions) of interest. Simple image processing tasks such as shifting, scaling, and rotation are ideal candidates for embarrassingly-parallel computations.

During the conversion of a traditional sequential (single-process) image-analysis program to an embarrassingly-parallel program, one must carefully consider 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. Astrophysical image-analysis programs are potential candidates for embarrassingly-parallel computation if the analysis of one subimage does not affect the analysis of another subimage.

L.A.COSMIC Algorithm

The L.A.COSMIC algorithm for cosmic-ray rejection is based on a variation of Laplacian edge detection; it identifies cosmic rays of arbitrary shapes and sizes by the sharpness of their edges and can reliably discriminate between poorly undersampled point sources and cosmic rays. The L.A.COSMIC algorithm is described in detail in reference [2] and is and is briefly outlined in Figure 2 (grey rectangles). The process is iterative and typically requires 4 iterations for the optimal removal of cosmic rays from *Hubble Space Telescope* Wide-Field Planetary Camera 2 (WFPC2) observations. Van Dokkum's IRAF script for cosmic-ray rejection in images, lacos_im.cl, is robust and requires very few user-defined parameters. Although lacos_im.cl does an excellent job in removing cosmic ray

defects in WFPC2 images, it has one major drawback -- it is slow. The L.A.COSMIC is an embarrassingly parallel algorithm and is ideally suited to being implemented as a parallel-processing image-analysis application.

CRBLASTER Application

I have written a portable parallel-processing image-analysis pro-gram, called CRBLASTER[1] which does cosmic ray rejection of CCD images using the L.A.COSMIC algo-rithm. CRBLASTER is written in C using the high-per-formance computing industry standard Message Passing Interface (MPI) library [3 – 5].

CRBLASTER uses a two-dimensional (2-D) image partitioning algorithm which segments an input image into *N* rectangular subimages of nearly equal area. CRBLASTER initially used a one-dimensional (1-D) image partitioning algorithm that segmented the input image into *N* subimages that were horizontal slices of the input image of nearly equal area. The original 1-D partitioning algorithm can be simulated as a 1 x *N* segmentation with the current 2-D partitioning algorithm.

The CRBLASTER code has been designed to be used by others as a parallel-processing computational framework that enables the easy development of other parallel-process-ing image-analysis programs based on embarrassingly-par-allel algorithms.

Figure 2 shows the flowchart diagram of CRBLASTER. An outline of CRBLASTER follows. The director process reads the input cosmic-ray-damaged FITS [6] image from disk, splits it into *N* subimages, and then sends them to the actor processes. Each actor process (sometimes 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 collects all of the cosmic-ray cleaned subimages and combines them together to form the cosmic-ray-cleaned output image which is then written to disk as a FITS image. The nitty gritty details of the computational framework of CRBLASTER will be discussed in much greater detail in the oral presentation.

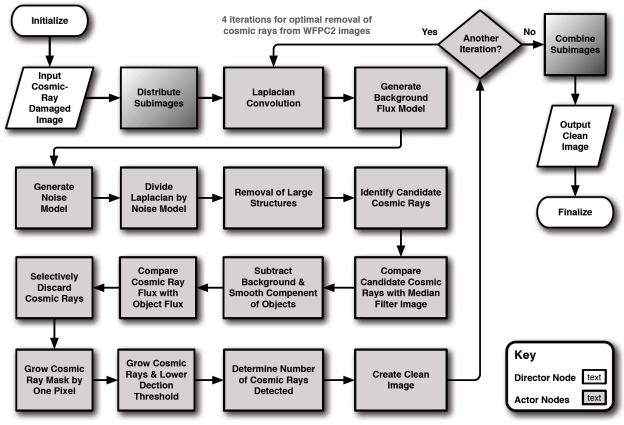

Figure 2: A flowchart diagram of CRBLASTER.

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

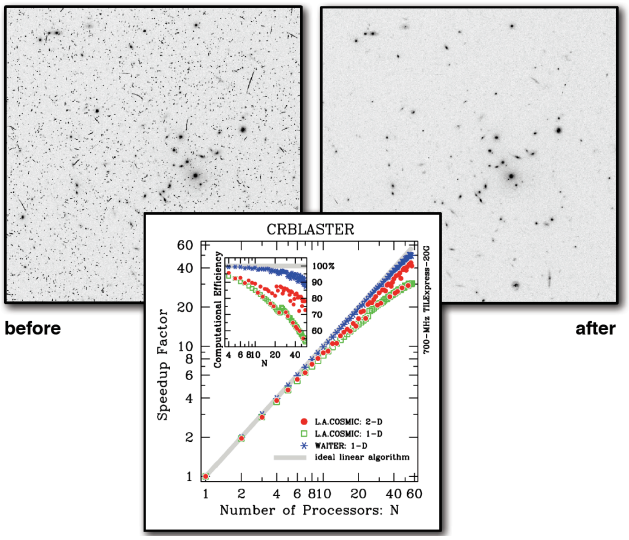

Figure 3: The measured performance of CRBBLASTER with 1 to 57 processors using a Tilera 64-core TILE64 processor.

The computational efficiencies of CRBLASTER with the nonlinear L.A.COSMIC, the nearly-linear POISSON, and linear WAITER work functions using a Tilera 64-core 700-MHz TILE64 processor with 1 to 57 processors is shown in Figure 3. The left image ("before") in the figure is part of a *HST* WFPC2 observation of the galaxy cluster MS 1137+67. The application of the L.A.COSMIC algorithm to the input image produces the image on the right ("after"). Note that almost all of the cosmic rays seen in the input image have been removed. The outer graph of the Figure 3 gives the speedup factor as a function of the number of processors on a log-log plot and the inner graph shows the computational efficiency as a function of the number of processors on a log-linear plot.

Acknowledgements

I am grateful to Dagim Seyoum for loaning me a Tilera TILExpress-20G PCIe card along with the supporting software. This work has been supported by a grant from the National Aeronautics and Space Administration (NASA), Inter-agency Order No. NNG06EC81I which was awarded by the Applied Information Systems Research (AISR) Program of NASA's Science Mission Directorate.

References

[1] B. Wilkinson, and M. Allen, *Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers*, Second Edition, Pearson Education Inc., 2004.

[2] P. G. van Dokkum, P. G. 2001, "Cosmic-Ray Rejection by Laplacian Edge Detection", *Publications of the Astronomical Society of the Pacific*, **113**, 1420–1427, 2001.

[3] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI – The Complete Reference (Volume 1: The MPI Core),* Second Edition, MIT Press, 1998.

[4] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, Second Edition, MIT Press, 1999.

[5] P. S. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann Publishers, Inc., 1997.

[6] D. C. Wells, E. W. Greisen, and R. H. Harten, "FITS – a Flexible Image Transport System", *Astronomy and Astrophysics Supplement Series*, 44, 363–370, 1981.