Astronomical Data Analysis Software and Systems XXI
ASP Conference Series, Vol. 461
Pascal Ballester, Daniel Egret, and Nuria P. F. Lorente, eds.

Using all of your CPU's in HIPE

Jeffery Douglas Jacobson and Dario Fadda

IPAC/NHSC, California Institute of Technology, 1200 E. California Blvd, Pasadena, California, 91125, USA

Abstract. Modern computer architectures increasingly feature multi-core CPU's. For example, the MacbookPro features the Intel quad-core i7 processors. Through the use of hyper-threading, where each core can execute two threads simultaneously, the quad-core i7 can support eight simultaneous processing threads. All this on your laptop! This CPU power can now be put into service by scientists to perform data reduction tasks, but only if the software has been designed to take advantage of the multiple processor architectures. Up to now, software written for Herschel data reduction (HIPE), written in Jython and JAVA, is single-threaded and can only utilize a single processor. Users of HIPE do not get any advantage from the additional processors. Why not put all of the CPU resources to work reducing your data? We present a multi-threaded software application that corrects long-term transients in the signal from the PACS unchopped spectroscopy line scan mode. In this poster, we present a multi-threaded software framework to achieve performance improvements from parallel execution. We will show how a task to correct transients in the PACS Spectroscopy Pipeline for the un-chopped line scan mode, has been threaded. This computation-intensive task uses either a one-parameter or a three parameter exponential function, to characterize the transient. The task uses a JAVA implementation of Minpack, translated from the C (Moshier) and IDL (Markwardt) by the authors, to optimize the correction parameters. We also explain how to determine if a task can benefit from threading (Amdahl's Law), and if it is safe to thread. The design and implementation, using the JAVA concurrency package completions service is described. Pitfalls, timing bugs, thread safety, resource control, testing and performance improvements are described and plotted.

1. HCSS, HIPE and the Data Reduction Pipelines.

The Herschel Common Science System includes the Herschel Interactive Processing Environment (HIPE). HIPE provides frameworks for reducing data taken from the three Herschel instruments, PACS, HIFI and SPIRE. This includes support for data reduction pipelines, calibration files, a numeric package, using Jython as its scripting language. The PACS Instrument Control Centre (ICC) provides pipelines for reducing data for the instrument AOTs.

2. The Problem

Transients occur in the signal whenever a sudden change in flux happens. Looking at Figure 1, you can see transients as the observation moves from start to calibration block, then to the on and off positions and off positions. Two types of transient effects

are present, an up-transient when the flux values increase and a down transient when the flux values decrease.

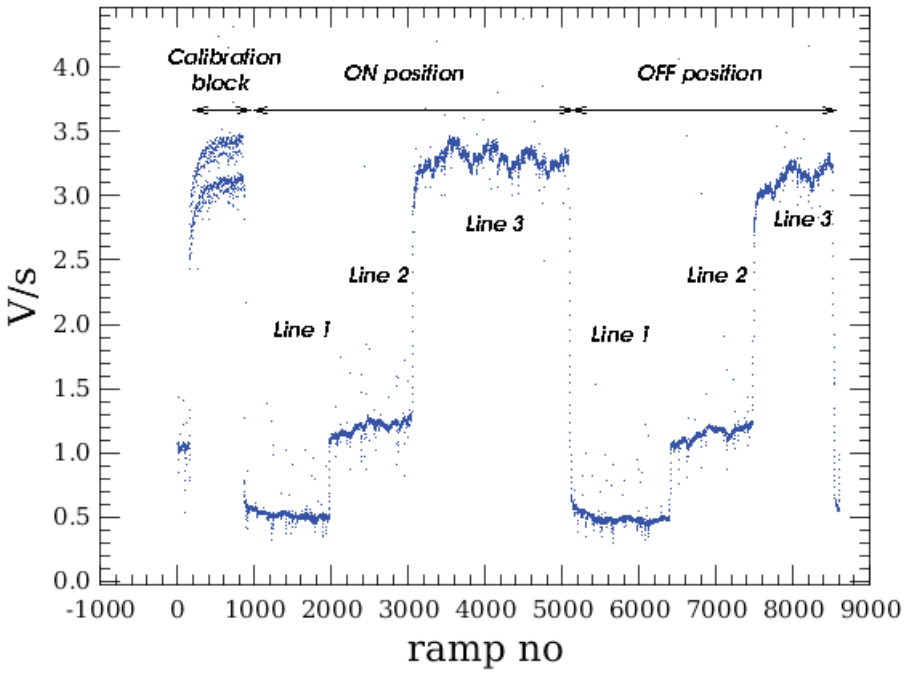

Figure 1. Sliced-Frames diagram, showing the way time ordered data are sliced, identifying calibration, on, and off slices. The science data slices are labeled by line number.

2.1. SpecLongTermTransient

The method to correct the long-term transients in the un-chopped line spectroscopy mode (see poster P042) is computationally intensive, applying a three parameter exponential function to up-transients:

$$1. + a[0] * EXP(-t/a[3]) + a[1] * EXP(-t/10.) + a[2] * EXP(-t/2000.), \quad (1)$$

and a one parameter exponential function:

$$1. + a[0] * EXP(-t/250.) \quad (2)$$

to the down transients.

The choice of functions is made by calculating a negative or positive gap in the median values of the flux for each time-ordered science frame, starting with the calibration block. The correction is implemented with the task SpecLongTermTransientTask which uses the Minpack package for minimization. This computationally intensive task served as the driver for the introduction of threading in HIPE. A JAVA version of Minpack has been introduced in HCSS mainly based on a C version (Moshir) and the IDL version of Markwardt.

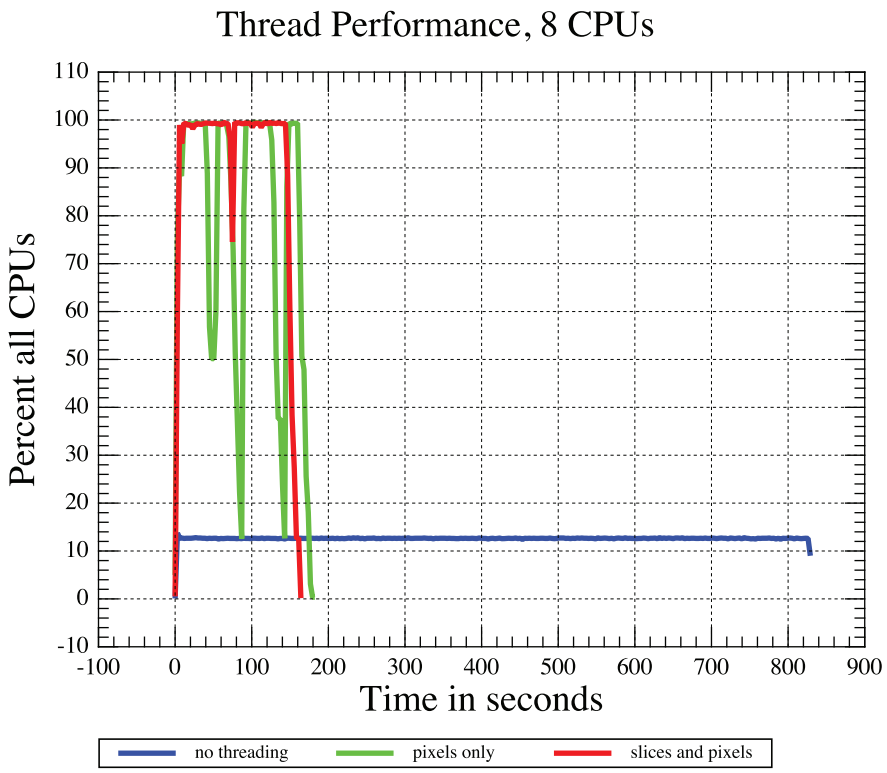

Figure 2. Plot of CPU usage: no threading, threading by pixels only, threading by pixels and slices.

2.2. How SpecLongTermTransient Was Threaded

First, we identified the computational units that can be run in parallel. PACS spectroscopy pipelines operate on Frames, or time-ordered sets of signal, wavelength, etc., which are organized into Cubes of 18 spectral pixels, 25 spatial pixels, and n-time. These Frames are contained in a type of ListContext called SlicedFrames, which supports iterators, and a master blocktable to index and describe the contents of each of the Frames. SpecLongTermTransient task looks at time and signal for each spatial pixel, independently for each scan by iterating through the sliced frames and applying the appropriate correction (up or down). Each of the 25 spatial pixels is independent, therefore all 25 can be processed in parallel. In addition, each set of Frames in slices Frames can be be processed independently. Figure 2 shows how these frames are sliced, slice 0 being the calibration block, slice 1 the on position and slice 2 is the off position of line 1, and so on.

We also examined the class libraries used for the transient corrections. Classes need to be applicable for use in threaded applications, requiring thread safety at a minimum. The level of thread safety was easy to determine due to the robust, clean design of the numerical package of HIPE.

SpecLongTermTransientTask uses two pools of threads for processing. One pool is for threads dedicated to processing the slices, the other for the actual correction of

each spatial pixel. Resource usage can be controlled by specifying the pool size as task parameters. The default is to create two threads for processing slices and NCPUs/2 threads for the spatial pixels. Thread synchronization is done through the JAVA concurrent package from which we use the JAVA ExecutorCompletionService for synchronization, and the Executors' fixedThreadPool.

The CompletionsService combines the function of a threaded Executor to allow the code to submit Callable tasks to the service and use take and poll methods to retrieve results. In the processing of spatial pixels or slices, Callable tasks are submitted to the completions service, the code synchronizes on task completion by using the BlockingQueue to retrieve completed tasks, or Futures. For a description of this mechanism, please see Goetz (2006) (section 6.3.5 "Completion Service: Executor meets BlockingQueue").

2.3. Performance Improvements

The first version threaded only the end of processing spatial pixels for each slice, but the CPU usage dropped when the number of unfinished spatial pixels became lower than the number of CPUs. Then we added another level of parallelization over the sliced frames, giving us an overlap to achieve increased throughput. Figure 3 shows the performance improvements gained from threading. A correction that takes 832 seconds without threading has been reduced to 167 seconds threading slices and pixels, for a 500% improvement.

2.4. When Does It Make Sense to Thread by Amdahl's Law ?

If F is the fraction of the calculation that must be executed serially, then Amdahl's Law says that on a machine with N processors we get a performance speedup (S) of at most:

$$S = \frac{1}{(F + (1 - F)/N)}. \tag{3}$$

As $\boldsymbol{N}$ approaches infinity, the maximum speedup is at most $\mathbf{1/F}$.

Optimize the number of threads you use by $\boldsymbol{N_{threads} = N_{CPU} * U_{CPU} + (1 + w/c)}$, where $\boldsymbol{N_{CPU}}$ is the number of CPUs, $\boldsymbol{U_{CPU}}$ is utilization, $\mathbf{0 \leq U_{CPU} \leq 1}$ and $\boldsymbol{W/C}$ is waiting time over computing time.

Acknowledgments. We would like to thank P. Appleton for his help, advice and testing, C. Borys and B. Ali (NHSC) for their support and advocacy of multi-threading.

References

Goetz, B. 2006, Java Concurrency in Practice (New York: Addison-Wesley Professional), 1st ed.